Cross-Sell on the Web Extended

STUDENT GUIDE

PEGA Academy

Mission: Cross-Sell on the Web Extended

Product: Pega Customer Decision Hub™ 8.7

URL: https://academy.pega.com/mission/cross-sell-web-extended/v4

Date: 16 December 2021

Contents

Business use case: Cross-sell on the web extended....4

  Business use case cross-sell on the web extended....5

Creating and understanding decision strategies....10

  Decision strategies....11

  Decision strategy canvas....16

  Creating a decision strategy....24

  Decision strategy execution....30

Creating engagement strategies using customer credit score....41

  Customer credit score using a scorecard....42

  Building a scorecard to calculate the creditscore....47

  Creating a decision strategy....51

  Using a Scorecard in a Decision Strategy....57

Creating eligibility rules using customer risk segments....63

  Creating customer risk segments using a decision table....64

Leveraging predictive models....74

  Predictive models drive predictions....75

  Arbitrating across business issues....80

  Using predictions in engagement strategies....82

Creating parameterized predictors using a sub strategy....88

  Creating parameterized predictors....89

Business use case: Cross-sell on the web extended

Description

Next-Best-Action Designer guides you through the creation of a core Next-Best-Action strategy for your business. Learn how to customize the core strategy by creating decision strategies from scratch that extend Next-Best-Action Designer capabilities.

Learning objectives

- Explain when to consider creating decision strategies from scratch

Business use case cross-sell on the web extended

Introduction

Next-Best-Action Designer guides you through the creation of a core Next-Best-Action strategy for your business. Learn how to customize the core strategy by creating decision strategies from scratch that extend Next-Best-Action Designer capabilities.

Transcript

This video describes several use cases where decision strategies are used to extend Next-Best-Action Designer capabilities.

U+ is a retail bank. The bank is leveraging its website as a marketing channel to improve 1-to-1 customer engagement, drive sales, and deliver Next-Best-Actions in real-time.

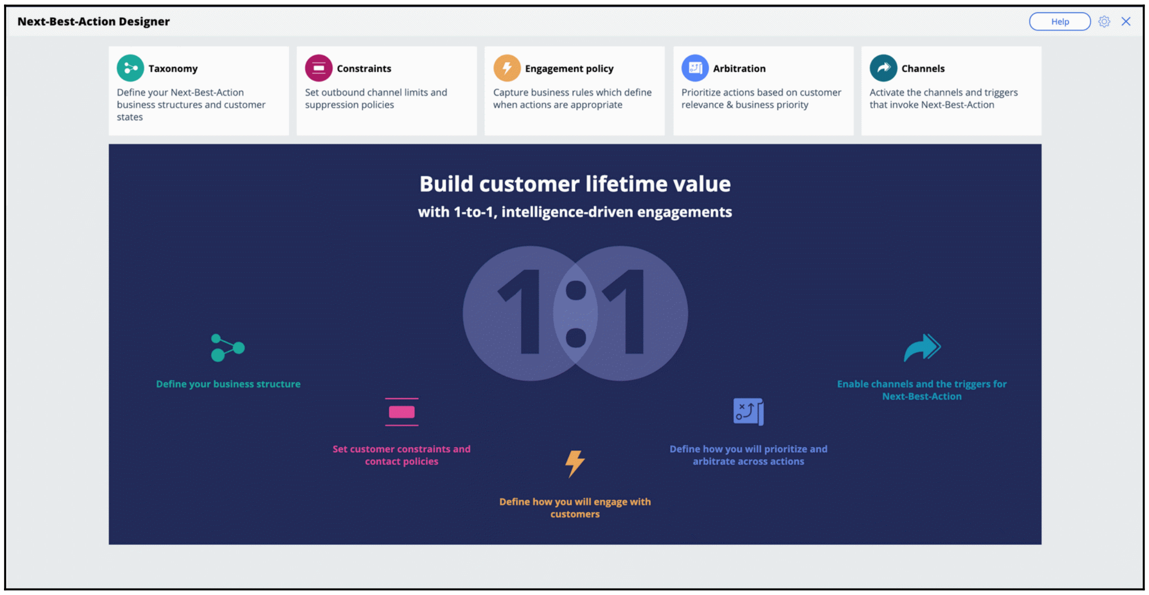
The bank is using the Pega Customer Decision Hub™ to recommend more relevant banner ads to its customers when they visit their personal portal. In the start-up phase, U+ successfully implemented all their use cases using Next-Best-Action Designer.

Next-Best-Action Designer guides you through the creation of a Next-Best-Action strategy for your business.

With the Next-Best-Action Designer user interface you define high-level business rules and AI controls, which the system uses to configure the underlying Next-Best-Action strategy framework.

This framework leverages best practices to automatically generate Next-Best-Action decision strategies at the enterprise level.

These decision strategies are a combination of the business rules and AI models that form the core of the Pega Customer Decision Hub, which determines the personalized set of Next-Best-Actions for each customer.


Next-Best-Action Designer
Help
Taxonomy
Define your Next-Best-Action business structures and customer states
Constraints
Set outbound channel limits and suppression policies
Engagement policy
Capture business rules which define when actions are appropriate
Arbitration
Prioritize actions based on customer relevance & business priority
Channels
Activate the channels and triggers that invoke Next-Best-Action
Build customer lifetime value
with 1-to-1, intelligence-driven engagements
1:1
Define your business structure
Set customer constraints and contact policies
Define how you will engage with customers
Define how you will prioritize and arbitrate across actions
Enable channels and the triggers for Next-Best-Action

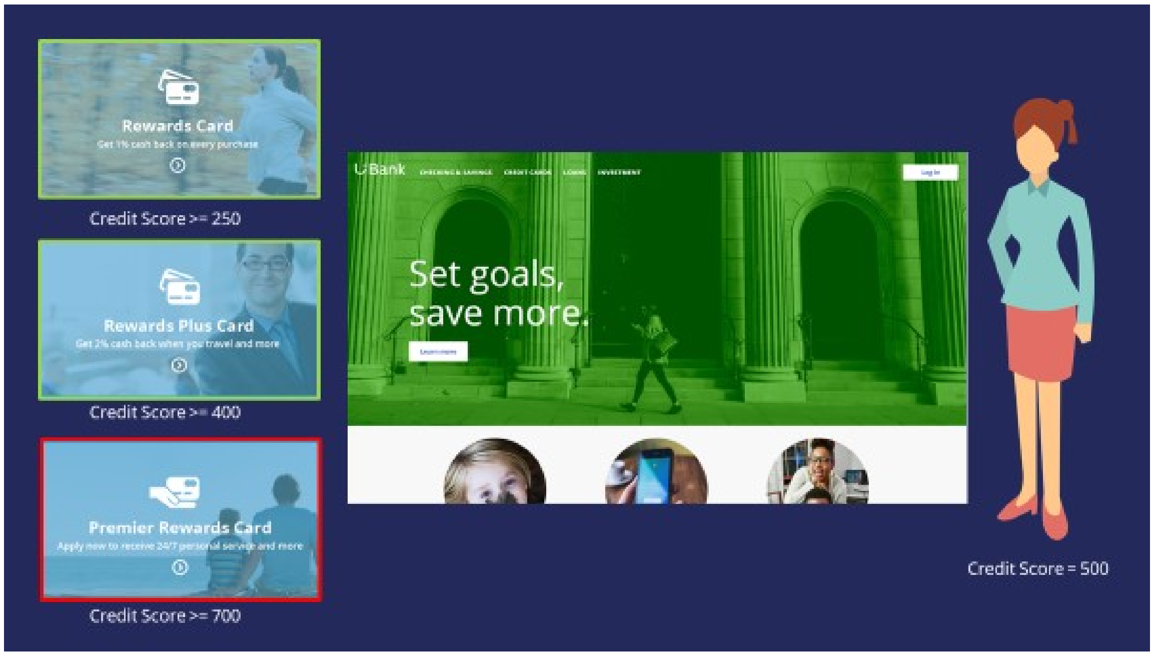
U+ Bank wants to implement additional use cases to meet new business requirements. These use cases require U+ bank to extend Next-Best-Action Designer capabilities.

Adding suitability rules based on customer credit score
Adding eligibility rules based on customer risk segments
Adding more tracking time periods
Testing strategies en masse with simulations

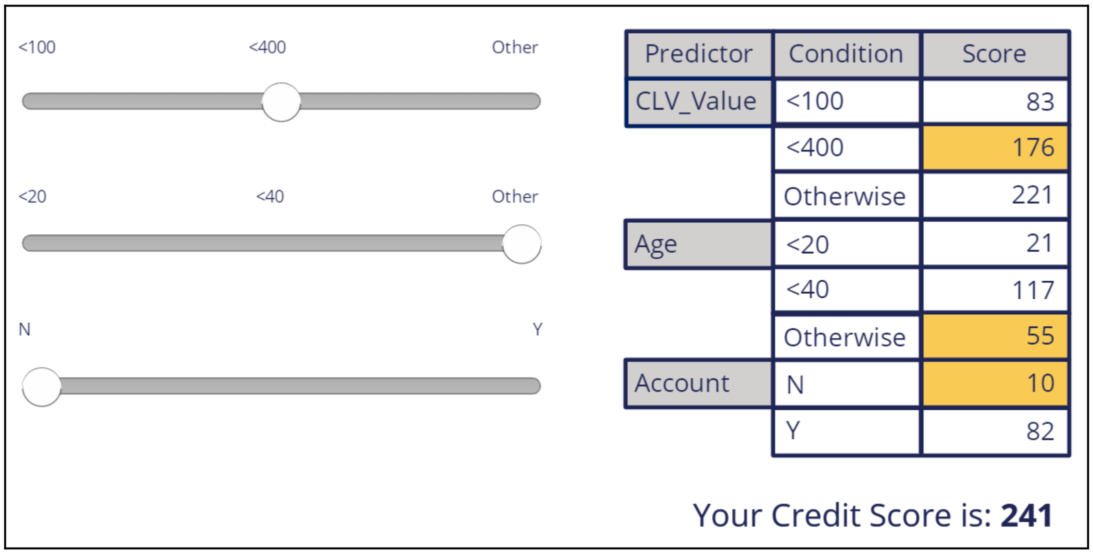
The first use case is to create suitability rules based on a customer's credit score. The bank wants to determine whether or not a customer is suitable for an offer based on their credit score. In this scenario, the credit score is not available, it needs to be computed in real-time based on customer profile information.

For example, when customer Barbara logs in, U+ bank only wants to present her with the Rewards and Rewards Plus offers, not the Premier Rewards offer. Barbara's credit score is

500. This makes her unsuitable for Premier Rewards, which is only suitable for customers with a credit score over 700.



To determine suitability, the bank wants to calculate a customer's credit score using a scorecard. A scorecard is used to assign importance to pieces of data for use in a calculation. A scorecard uses a subset of customer property values divided into ranges and assigns scores to each range to compute a final score.

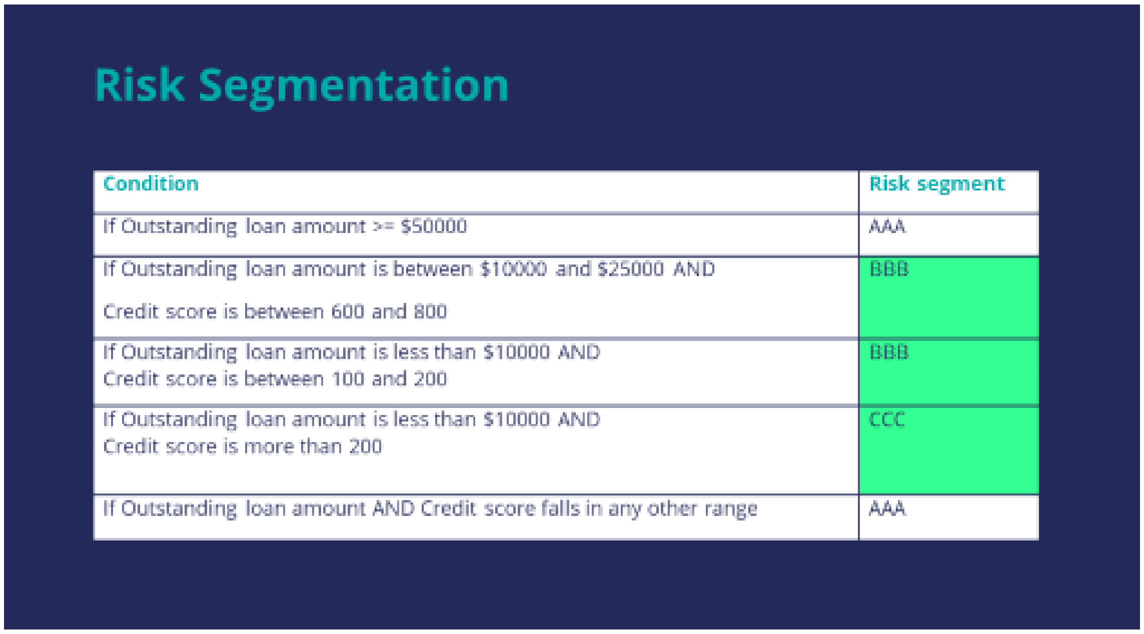

<100
<400
Other
<20
<40
Other
N
Y

Predictor	Condition	Score
CLV_Value	<100	83
	<400	176
	Otherwise	221
Age	<20	21
	<40	117
	Otherwise	55
Account	N	10
	Y	82

Your Credit Score is: **241**

To implement this, you use a special Suitability condition in Next-Best-Action Designer. The Suitability condition uses a decision strategy that references a Scorecard rule. The Scorecard rule is used to determine the customer's credit score.

In the next use case, the bank has introduced some strict regulations for which they need new eligibility rules. U+ does not want to offer credit cards to customers whom they classify as 'high risk'. Customers are divided into risk segments from AAA to CCC based on their outstanding loan amount and credit score. In the beginning, only customers in the risk segments BBB and CCC will be eligible for credit cards.



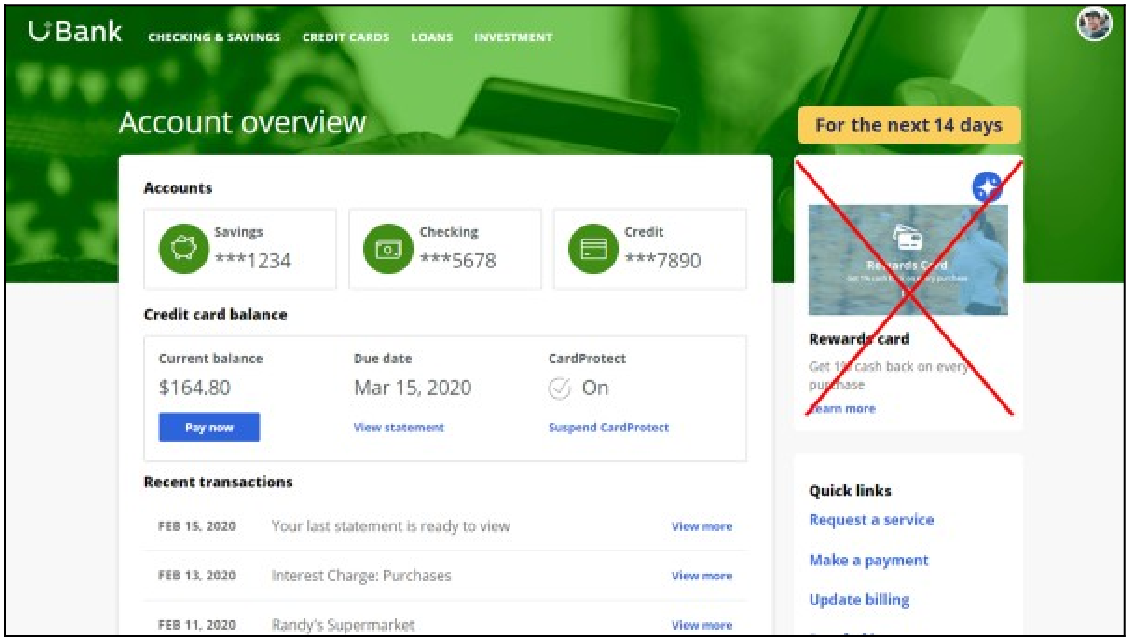
Risk Segmentation

Condition	Risk segment
If Outstanding loan amount >= $50000	AAA
If Outstanding loan amount is between $10000 and $25000 AND Credit score is between 600 and 800	BBB
If Outstanding loan amount is less than $10000 AND Credit score is between 100 and 200	BBB
If Outstanding loan amount is less than $10000 AND Credit score is more than 200	CCC
If Outstanding loan amount AND Credit score falls in any other range	AAA

To implement this, you use a special Eligibility condition in Next-Best-Action Designer. The Eligibility condition leverages a decision strategy that uses the scorecard to determine the customer's credit score, which it passes as an argument to a decision table. The decision table then determines which risk segment the customer is in.

The next use case is about adding more time periods for tracking customer behavior.

In this use case, the bank does not want to show the same offer to customers who have clicked on it three times in the last 14 days. By default, the Pega Customer Decision Hub tracks customer responses for a period of 7 or 30 days.


UBank CHECKING & SAVINGS CREDIT CARDS LOANS INVESTMENT

Account overview

For the next 14 days

Accounts

Savings ***1234

Checking ***5678

Credit ***7890

Credit card balance

Current balance	Due date	CardProtect
$164.80	Mar 15, 2020	On
Pay now	View statement	Suspend CardProtect

Recent transactions

FEB 15, 2020	Your last statement is ready to view	View more
FEB 13, 2020	Interest Charge: Purchases	View more
FEB 11, 2020	Randy's Supermarket	View more

~~Rewards card~~

~~Get 1% cash back on every purchase~~

~~Learn more~~

Quick links

Request a service

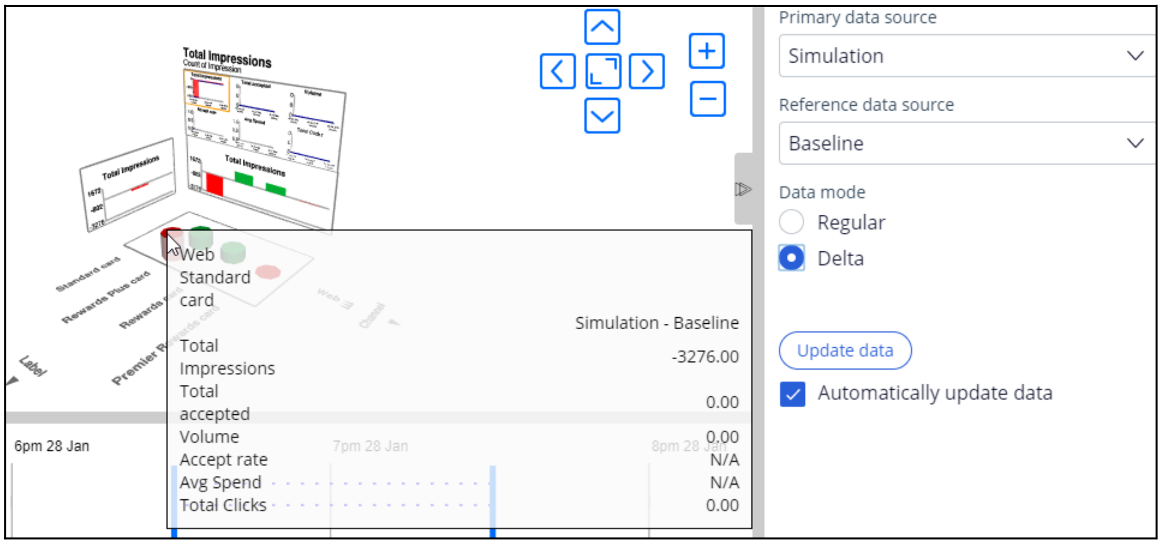
Make a payment

Update billing

This use case requires an additional tracking time period of 14 days.

To add a new tracking time period you have to extend the decision strategies that are used to implement contact policies. This requires creating a new Interaction History Summary rule which tracks customer responses for 14 days.

Finally, while the bank is implementing these various changes, they want to understand what the impact will be on their Next-Best-Actions. Therefore, U+ Bank would like to run some simulations to test the effects of the changes. The results of the simulations can be analyzed using Visual Business Director.



Web Standard card

	Simulation - Baseline
Total Impressions	-3276.00
Total accepted	0.00
Volume	0.00
Accept rate	N/A
Avg Spend	N/A
Total Clicks	0.00

Primary data source: Simulation

Reference data source: Baseline

Data mode: Regular [✓ Delta]

Update data

☑ Automatically update data

In summary, this video has shown you the various use cases U+ Bank would like to implement by extending Next-Best-Action Designer capabilities in this phase of the project.

Creating and understanding decision strategies

Description

Next-Best-Action Designer provides a guided and intuitive UI to bootstrap your application development with proven best practices that generate the underlying strategies for you. These strategies can be customized using designated extension points or by building decision strategies from scratch, depending on the business requirement.

Learning objectives

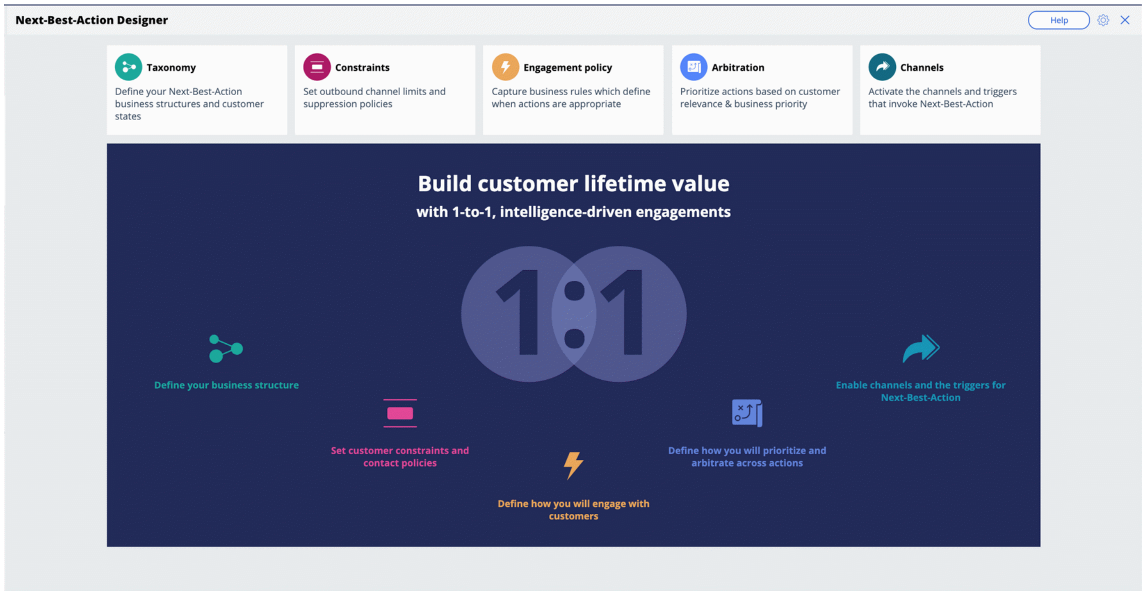
- Describe how decision strategies are used in the Next-Best-Action strategy framework
- Explain the decision strategy canvas and its building blocks
- Create decision strategies from scratch
- Explain what's going on inside each component when a decision strategy is executed

Decision strategies

Transcript

Next-Best-Action Designer guides you through the creation of a Next-Best-Action strategy for your business. Its intuitive interface, proven best practices and sophisticated underlying decisioning technology enable you to automatically deliver personalized customer experiences across inbound, outbound and paid channels.

The Next-Best-Action Designer user interface allows you to easily define, manage and monitor Next-Best-Actions.



Taxonomy
Define your Next-Best-Action business structures and customer states

Constraints
Set outbound channel limits and suppression policies

Engagement policy
Capture business rules which define when actions are appropriate

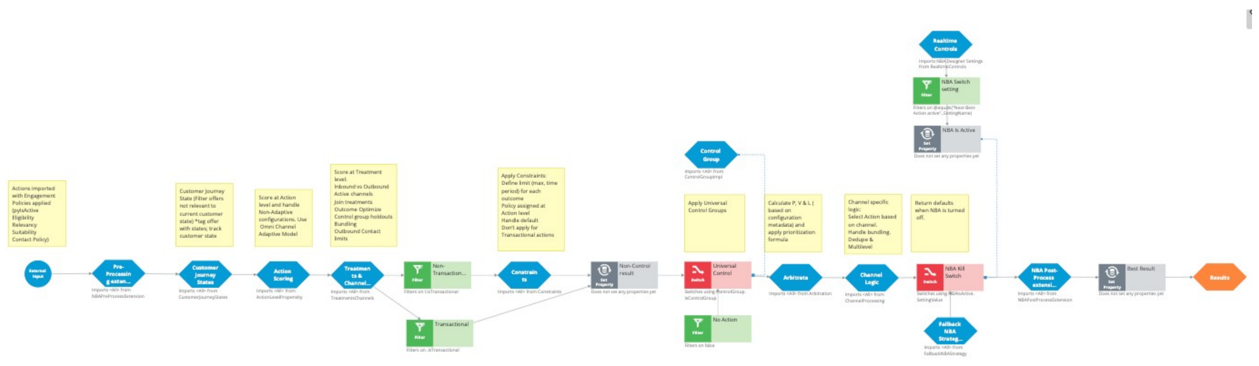
Arbitration
Prioritize actions based on customer relevance & business priority

Channels
Activate the channels and triggers that invoke Next-Best-Action

Build customer lifetime value
with 1-to-1, intelligence-driven engagements

1:1
Define your business structure
Set customer constraints and contact policies
Define how you will engage with customers
Define how you will prioritize and arbitrate across actions
Enable channels and the triggers for Next-Best-Action

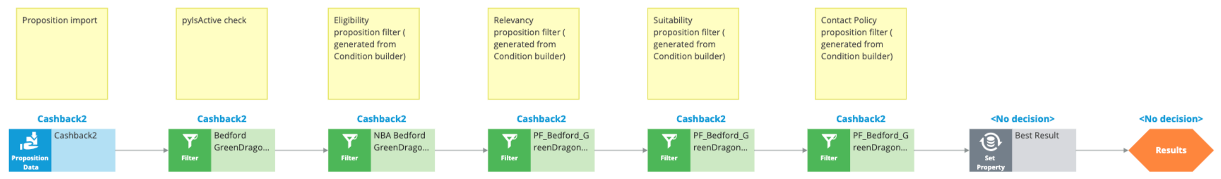
As you use the Next-Best-Action Designer user interface to define strategy criteria, the system uses these criteria to create the Next-Best-Action Strategy framework. This framework leverages best practices to generate Next-Best-Action decision strategies at the enterprise level. These decision strategies are a combination of the business rules and AI models that form the core of the Pega Centralized Decision Hub, which determines the personalized set of Next-Best-Actions for each customer.


Actions imported with Engagement Policies applied (pyIsActive Eligibility Relevancy Suitability Contact Policy)
External Input
Pre-Processing exten...
Imports <All> from NBAPreProcessExtension
Customer Journey State (Filter offers not relevant to current customer state) *tag offer with states; track customer state
Customer Journey States
Imports <All> from CustomerJourneyStates
Score at Action level and handle Non-Adaptive configurations. Use Omni Channel Adaptive Model
Action Scoring
Imports <All> from ActionLevelPropensity
Score at Treatment level: Inbound vs Outbound Active channels Join treatments Outcome Optimize Control group holdouts Bundling Outbound Contact limits
Treatments & Channel...
Imports <All> from TreatmentsChannels
Filter
Non-Transaction...
Filters on !.IsTransactional
Filter
Transactional
Filters on .IsTransactional
Apply Constraints: Define limit (max, time period) for each outcome Policy assigned at Action level Handle default Don't apply for Transactional actions
Constraints
Imports <All> from Constraints
Set Property
Non-Control result
Does not set any properties yet
Control Group
Imports <All> from ControlGroupImpl
Apply Universal Control Groups
Switch
Universal Control
Switches using ControlGroup.IsControlGroup
Filter
No Action
Filters on false
Calculate P, V & L (based on configuration metadata) and apply prioritization formula
Arbitrate
Imports <All> from Arbitration
Channel specific logic: Select Action based on channel. Handle bundling, Dedupe & Multilevel
Channel Logic
Imports <All> from ChannelProcessing
Realtime Controls
Imports NBA Designer Settings from RealtimeControls
Filter
NBA Switch setting
Filters on @equals("Next-Best-Action active",.SettingName)
Set Property
NBA Is Active
Does not set any properties yet
Return defaults when NBA is turned off.
Switch
NBA Kill Switch
Switches using NBAIsActive.SettingValue
Fallback NBA Strateg...
Imports <All> from FallbackNBAStrategy
NBA Post-Process extensi...
Imports <All> from NBAPostProcessExtension
Set Property
Best Result
Does not set any properties yet
Results

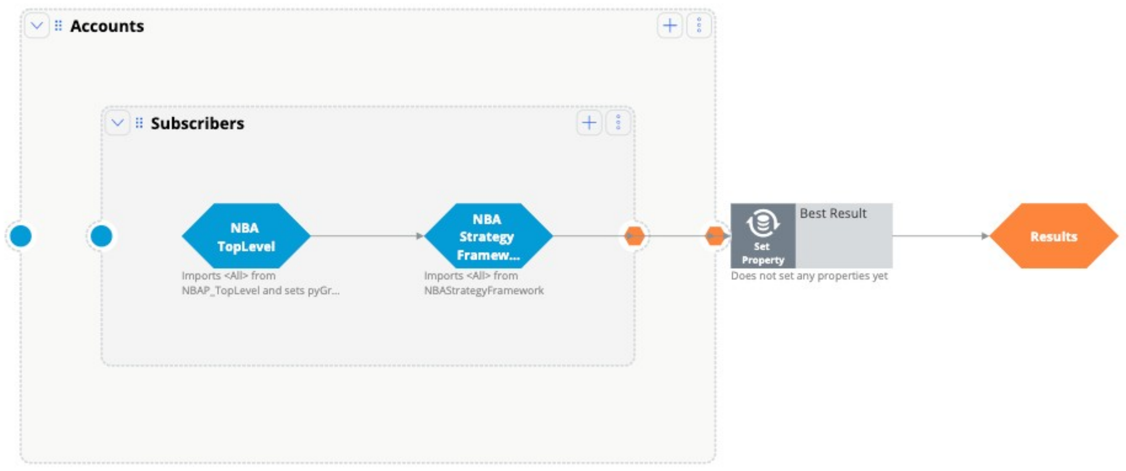
If you want to modify the strategy later, you can do that from Next-Best-Action Designer's simple and transparent interface.

The strategy framework is applied to all relevant Actions and Treatments after you define a Trigger in the Next-Best-Action Designer **Channels** tab.

Each Trigger generates a strategy that first imports the Actions from the appropriate level of the business structure and then applies the Eligibility, Relevance and Suitability rules.

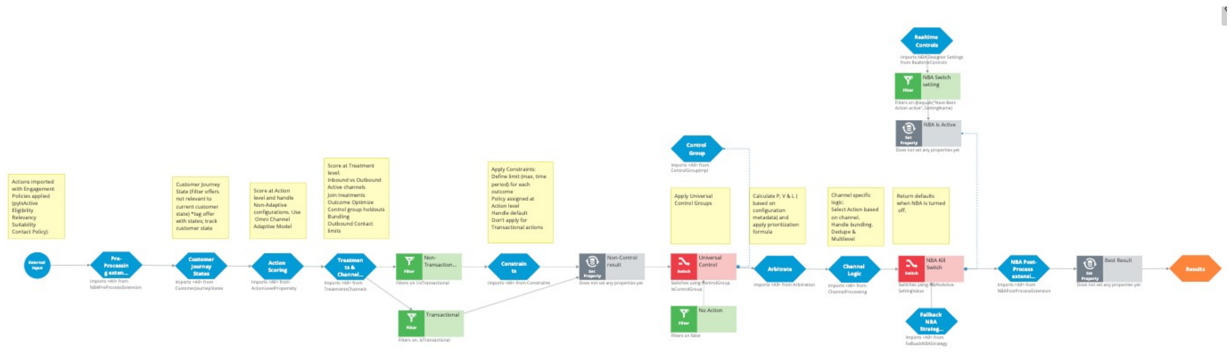

Proposition import
pyIsActive check
Eligibility proposition filter (generated from Condition builder)
Relevancy proposition filter (generated from Condition builder)
Suitability proposition filter (generated from Condition builder)
Contact Policy proposition filter (generated from Condition builder)
Cashback2
Proposition Data
Cashback2
Cashback2
Filter
Bedford GreenDrago...
Cashback2
Filter
NBA Bedford GreenDrago...
Cashback2
Filter
PF_Bedford_GreenDragon...
Cashback2
Filter
PF_Bedford_GreenDragon...
Cashback2
Filter
PF_Bedford_GreenDragon...
<No decision>
Set Property
Best Result
<No decision>
Results

The strategy then passes these results to the strategy framework for processing.

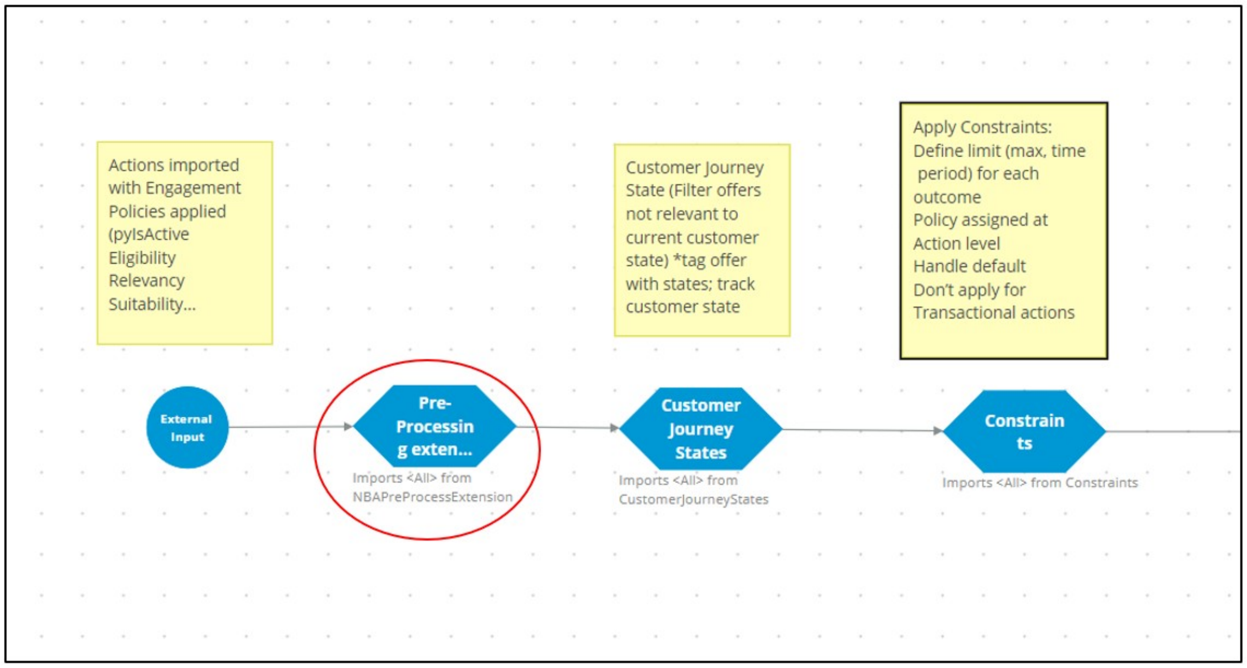

Accounts
Subscribers
NBA TopLevel
Imports <All> from NBAP_TopLevel and sets pyGr...
NBA Strategy Framew...
Imports <All> from NBAStrategyFramework
Set Property
Best Result
Does not set any properties yet
Results

There are several extension points within the framework. An extension point is an empty rule or activity that is intended to be overridden to meet the specific needs of the application. When building an implementation of the current framework, the decision strategy designers must override the empty activity with a functioning interface to their customer master file.

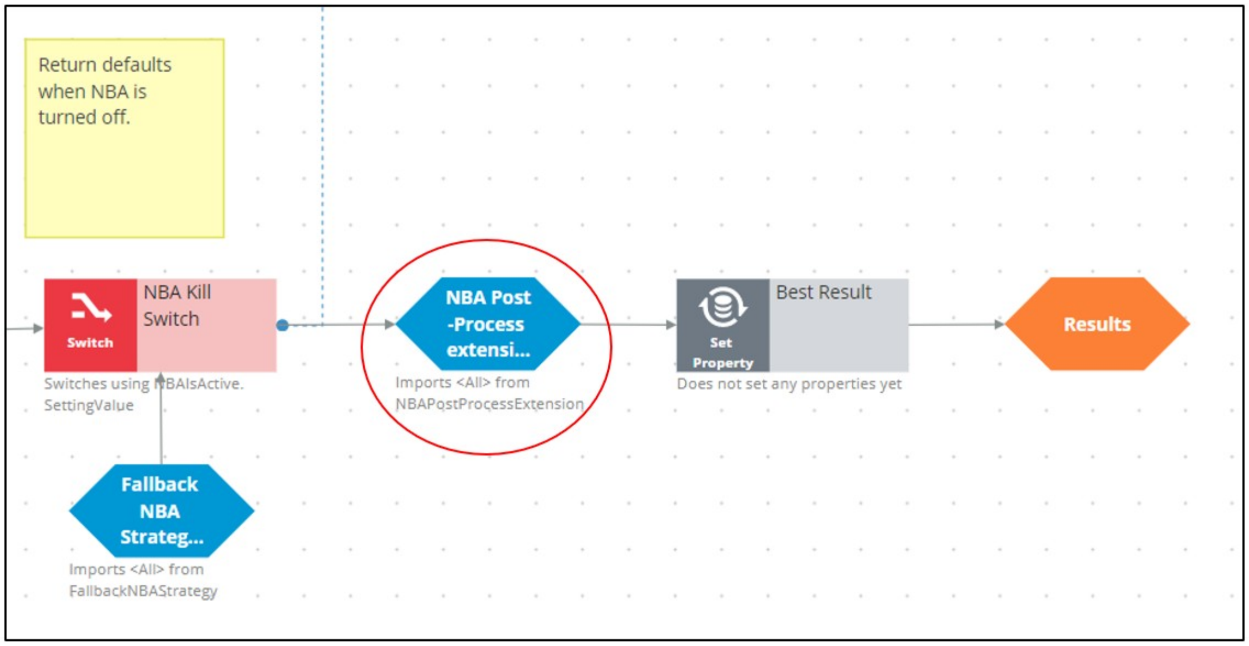
This is the NBA framework strategy when applied to each of the Actions.


Actions imported with Engagement Policies applied (pyIsActive Eligibility Relevancy Suitability Contact Policy)
Customer Journey State (Filter offers not relevant to current customer state) *tag offer with states; track customer state
Score at Action level and handle Non-Adaptive configurations. Use Omni Channel Adaptive Model
Score at Treatment level: Inbound vs Outbound Active channels Join treatments Outcome Optimize Control group holdouts Bundling Outbound Contact limits
Apply Constraints: Define limit (max, time period) for each outcome Policy assigned at Action level Handle default Don't apply for Transactional actions
External Input
Pre-Processin g exten...
Imports <All> from NBAPreProcessExtension
Customer Journey States
Imports <All> from CustomerJourneyStates
Action Scoring
Imports <All> from ActionLevelPropensity
Treatmen ts & Channel...
Imports <All> from TreatmentsChannels
Filter
Non-Transaction...
Filters on !.IsTransactional
Filter
Transactional
Filters on .IsTransactional
Constrain ts
Imports <All> from Constraints
Set Property
Non-Control result
Does not set any properties yet
Control Group
Imports <All> from ControlGroupImpl
Apply Universal Control Groups
Calculate P, V & L (based on configuration metadata) and apply prioritization formula
Channel specific logic: Select Action based on channel. Handle bundling. Dedupe & Multilevel
Return defaults when NBA is turned off.
Realtime Controls
Imports NBA Designer Settings from RealtimeControls
Filter
NBA Switch setting
Filters on @equals("Next-Best-Action active",.SettingName)
Set Property
NBA Is Active
Does not set any properties yet
Switch
Universal Control
Switches using ControlGroup. isControlGroup
Filter
No Action
Filters on false
Arbitrate
Imports <All> from Arbitration
Channel Logic
Imports <All> from ChannelProcessing
Switch
NBA Kill Switch
Switches using NBAIsActive. SettingValue
Fallback NBA Strateg...
Imports <All> from FallbackNBAStrategy
NBA Post-Process extensi...
Imports <All> from NBAPostProcessExtension
Set Property
Best Result
Does not set any properties yet
Results

The first component within the strategy framework is an extension point for any Action pre-processing you might need to perform.


Actions imported with Engagement Policies applied (pyIsActive Eligibility Relevancy Suitability...
Customer Journey State (Filter offers not relevant to current customer state) *tag offer with states; track customer state
Apply Constraints: Define limit (max, time period) for each outcome Policy assigned at Action level Handle default Don't apply for Transactional actions
External Input
Pre-Processin g exten...
Imports <All> from NBAPreProcessExtension
Customer Journey States
Imports <All> from CustomerJourneyStates
Constrain ts
Imports <All> from Constraints

The last functional component within the strategy framework is another extension point for any post-processing that must be performed.


Return defaults when NBA is turned off.
NBA Kill Switch
Switch
Switches using NBAIsActive. SettingValue
Fallback NBA Strateg...
Imports <All> from FallbackNBAStrategy
NBA Post -Process extensi...
Imports <All> from NBAPostProcessExtension
Set Property
Best Result
Does not set any properties yet
Results

Similarly, there are many other extension points such as the outbound limits extension points and business value extension points.

To ensure upgradeability, avoid overriding any part of the framework that is not a designated extension point.

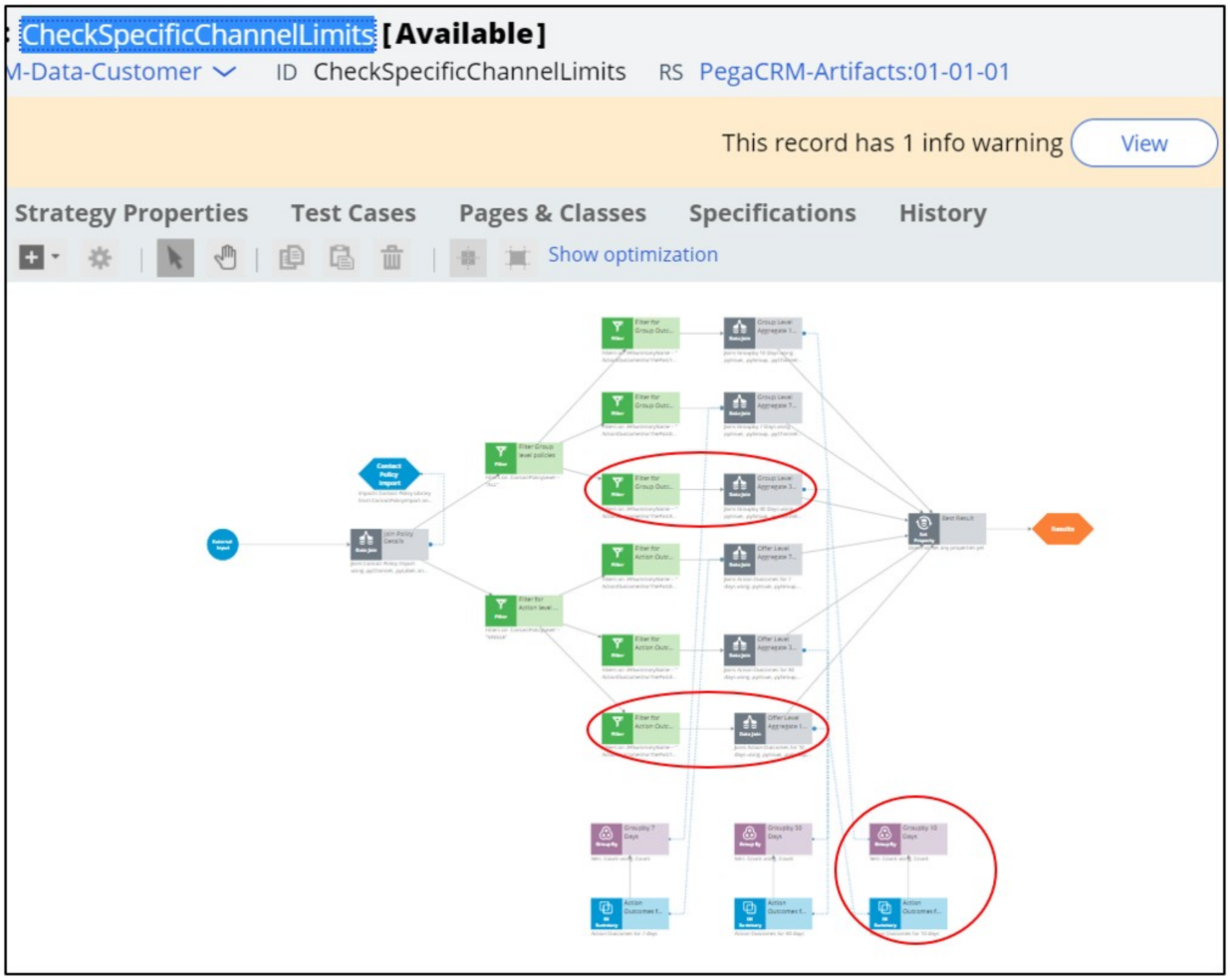
Also, the generated framework has some extension points where you can create strategies.

For example, while configuring values for Arbitration, you can specify a business value for an Action, or you can use a strategy to calculate the value. This can be done by adding a strategy to the existing framework.

Similarly, in defining the engagement rules, you can use a new strategy as a definition instead of an existing condition. Strategy designers can create such strategies from scratch using the decision strategy canvas.

Or, while defining the suppression rules, you can add a strategy to define new suppression rule limits instead of the existing 7 or 30 days.

For example, in the screenshot below, the CheckSpecificChannelLimits rule has been extended to have a 15-day limit:


: CheckSpecificChannelLimits [Available]
M-Data-Customer ID CheckSpecificChannelLimits RS PegaCRM-Artifacts:01-01-01
This record has 1 info warning View
Strategy Properties Test Cases Pages & Classes Specifications History
Show optimization

In conclusion, the NBA Designer provides a guided and intuitive UI to bootstrap your application development with proven best practices. NBA designer generates the underlying strategies for you, which can be extended using existing values in the designated extension points or by building decision strategies from scratch, depending on the business requirement.

Decision strategy canvas

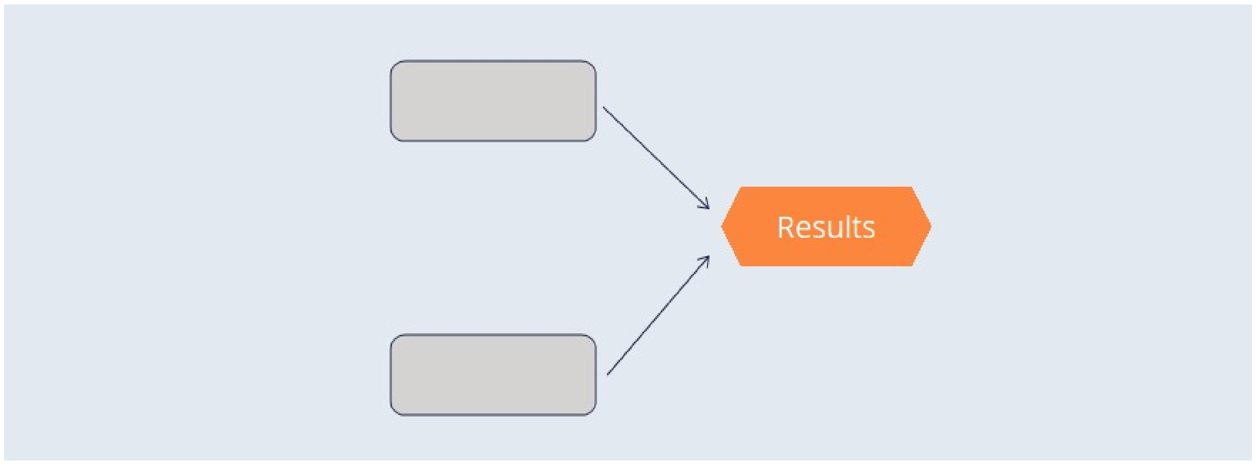
Introduction

Decision strategies drive the next best action and comprise a unit of reasoning represented by decision components. You use the Proposition Data component to import actions into a strategy canvas. The sequence of the components in the canvas determines which action is selected for a customer.

Click the Play button to learn more about decision strategies.

Screen1: U+ business scenario

U+, a telecom organization, wants to promote two new phones in the contact center: iPhone and Galaxy. Click the + icons to learn more about the elements of a decision strategy that is created for this requirement.



Decision Components: A decision strategy is comprised of building blocks called decision components. You can add and connect components to implement the business requirements.

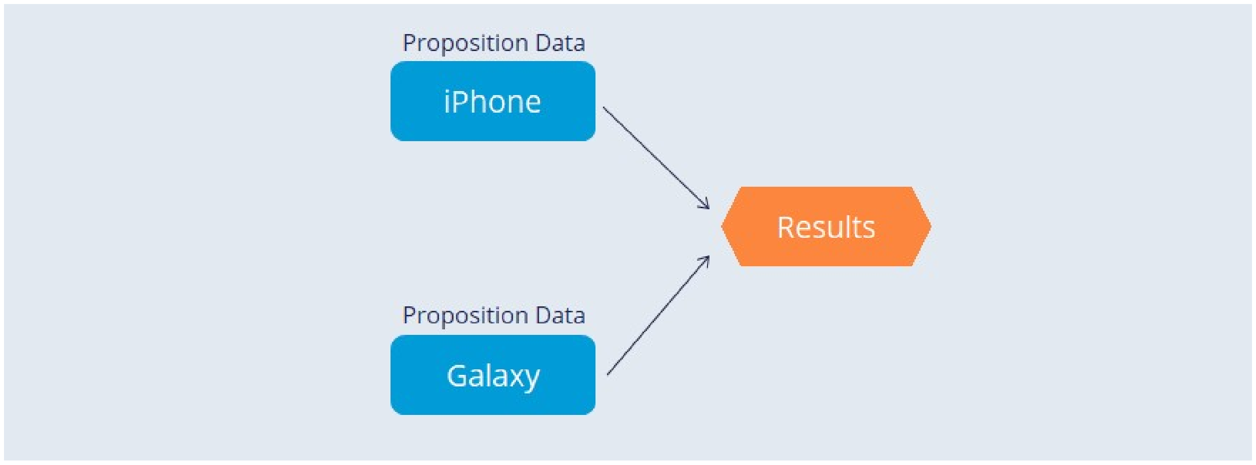
Arrows: An important element of the strategy canvas is the arrow. An arrow connects two decision components. A solid line means the data is copied from one component to another.

Strategy Canvas: In Pega, business users visually design decision strategies on what is known as a strategy canvas.

Screen2: Proposition Data component

The Proposition Data decision component imports the properties of an action. The result of this component is a flat list of all the properties.

Click the + icons on the proposition components to examine the components' results.



iPhone: This Proposition Data component outputs the Price and the Cost properties of the iPhone action.

Name: iPhone

Price: 150

Cost: 100

Galaxy: This Proposition Data component outputs the Price and the Cost properties of the Galaxy action.

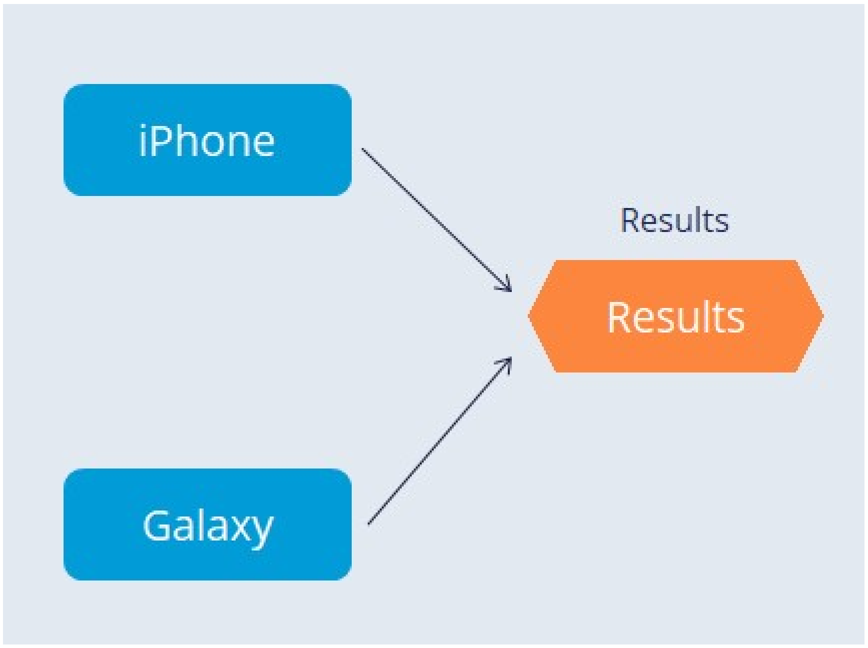
Name: Galaxy

Price: 250

Cost: 150

Screen3: Results component

Another decision component is the Results component. Each strategy always contains one Results component, which defines the output of the decision strategy.



How many actions do you think this strategy outputs?

0

1

2

Feedback: Because both iPhone and Galaxy are connected to the Results component with a solid arrow line, this decision strategy outputs two actions.

Dynamic pricing

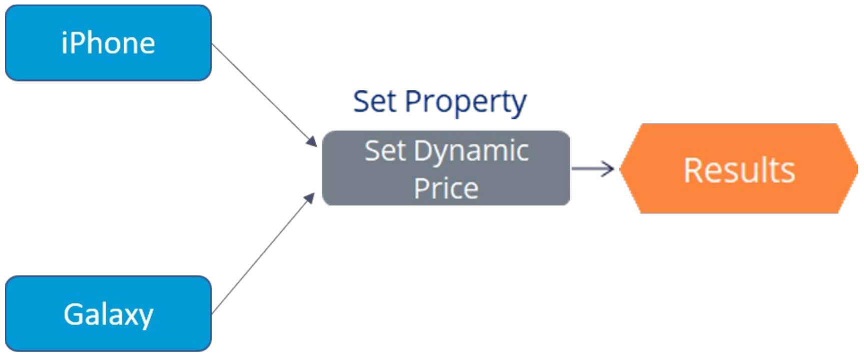
U+ Bank wants a dynamic Price for all offered actions. If the Customer value of a customer is higher than 60, the bank wants to offer a 10% discount to the customer.

To meet the new requirement, you must enhance the existing strategy to set the value of the Price based on Customer value. Changing the Price dynamically based on the Customer value makes the pricing customer-centric.

Set Property component

The Set Property component is used to dynamically alter the value of an action property based on a customer property. You use this component to set values to properties that are output by the strategy.

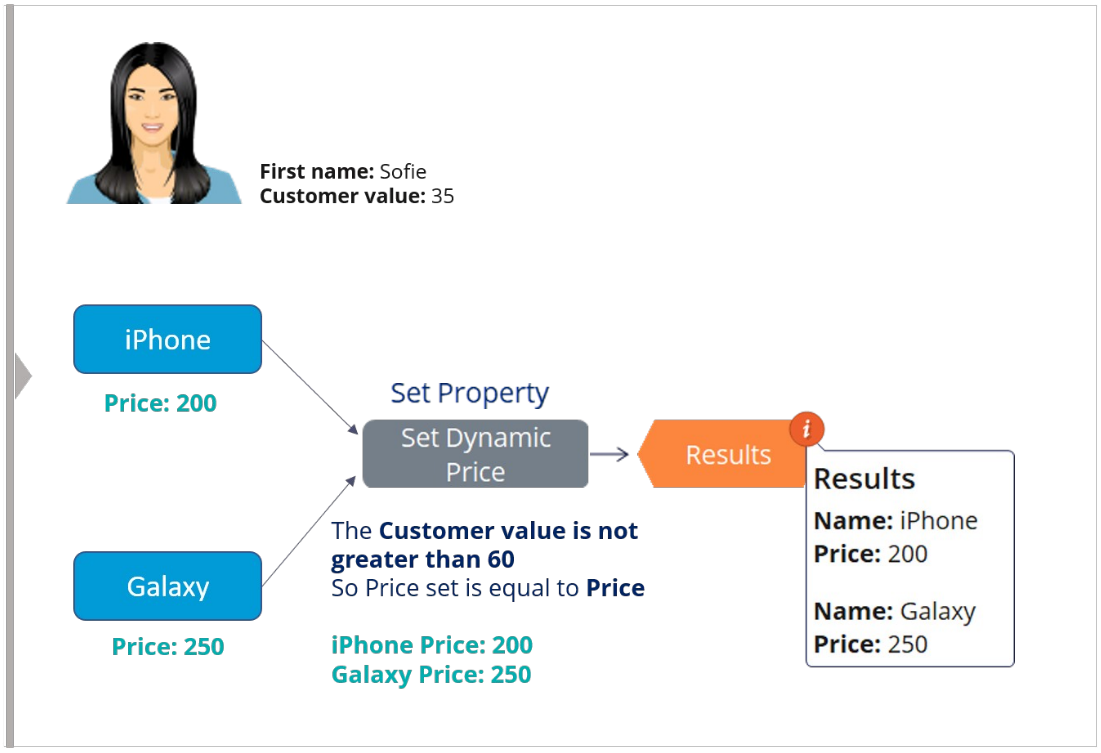
You can set properties to a constant or calculated value.



Example

Consider two customers: Sofie and Lily with customers value 35 and 65 respectively.

In the center of the following image, slide the vertical line to see how Sofie and Lily's Customer value affects the Price of the action offered to them.

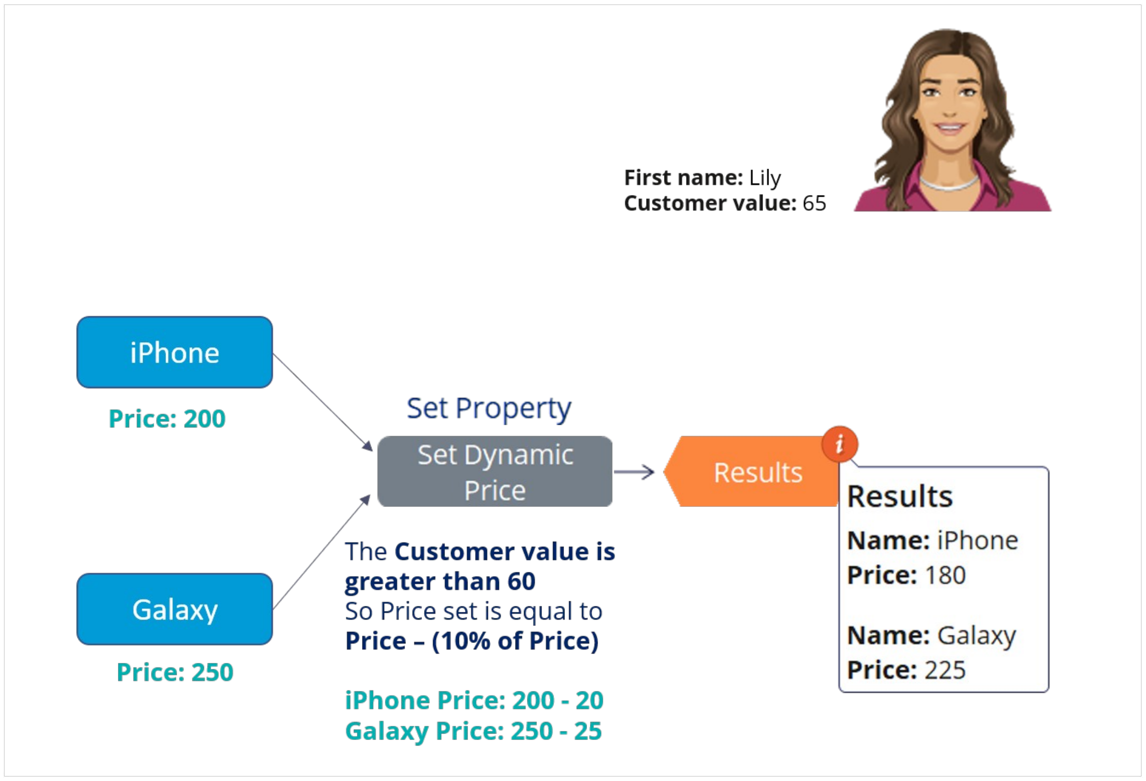


The **Customer value is not greater than 60**
So Price set is equal to **Price**

iPhone Price: 200
Galaxy Price: 250

Results
Name: iPhone
Price: 200

Name: Galaxy
Price: 250



The **Customer value is greater than 60**
So Price set is equal to
Price – (10% of Price)

iPhone Price: 200 - 20
Galaxy Price: 250 - 25

Results
Name: iPhone
Price: 180

Name: Galaxy
Price: 225

Action ranking

U+ wants to offer the most profitable action to its customers.

To enhance this strategy based on the new requirement, you need a new decision component that can rank the actions based on Profit and select the highest ranked action.

Profit is calculated based on Price and Cost action properties.

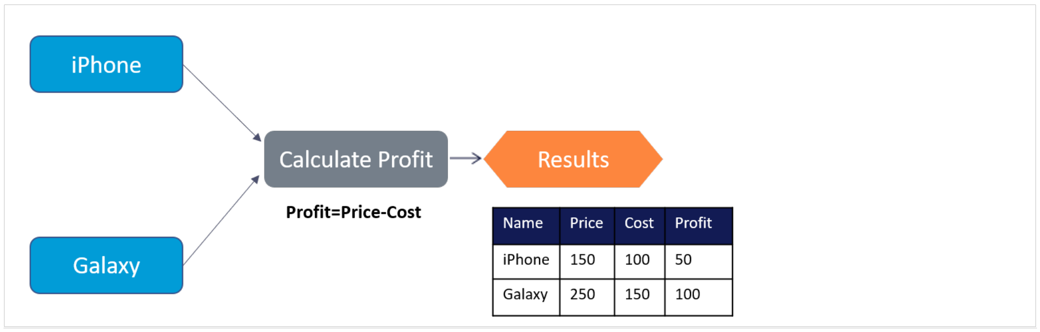
Prioritize component

The Prioritize component is a decision strategy component used to rank actions. The Prioritize component is also used to select the top 1, top 2, or arbitrary top-n actions.

Click the Play icon to learn more about action ranking in detail with the help of a sample strategy.

Screen 1:

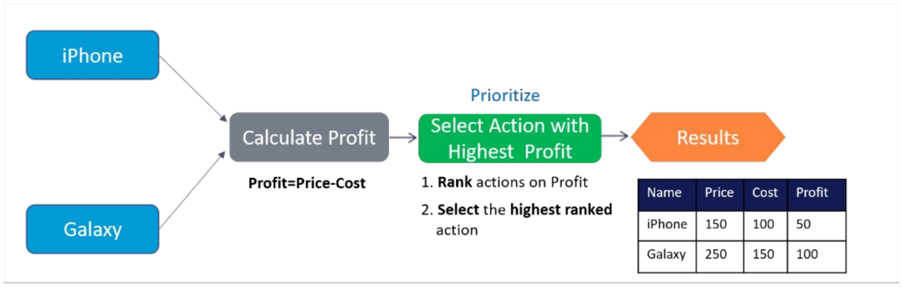
The initial strategy outputs price, cost and the profit calculated by the Set Property component. Click Add Prioritize to add the Prioritize component to the strategy.



Name	Price	Cost	Profit
iPhone	150	100	50
Galaxy	250	150	100

Screen 2:

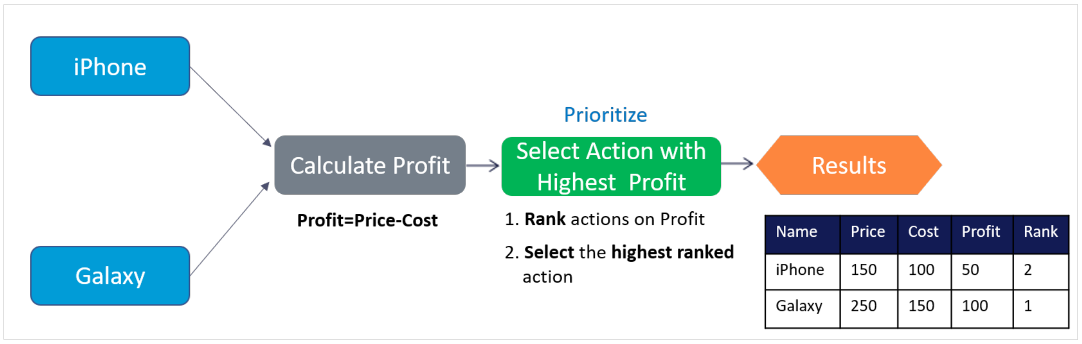
The Prioritize decision component can perform two operations: rank actions based on an expression, and select the highest ranked action. Click Rank actions to see how the component rank the actions.


iPhone
Galaxy
Calculate Profit
Profit=Price-Cost
Prioritize
Select Action with Highest Profit
1. Rank actions on Profit
2. Select the highest ranked action
Results

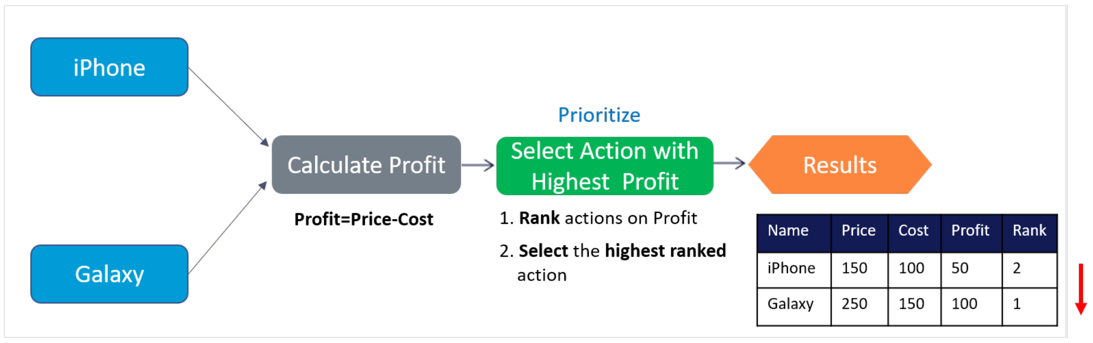
Name	Price	Cost	Profit
iPhone	150	100	50
Galaxy	250	150	100

Screen 3:

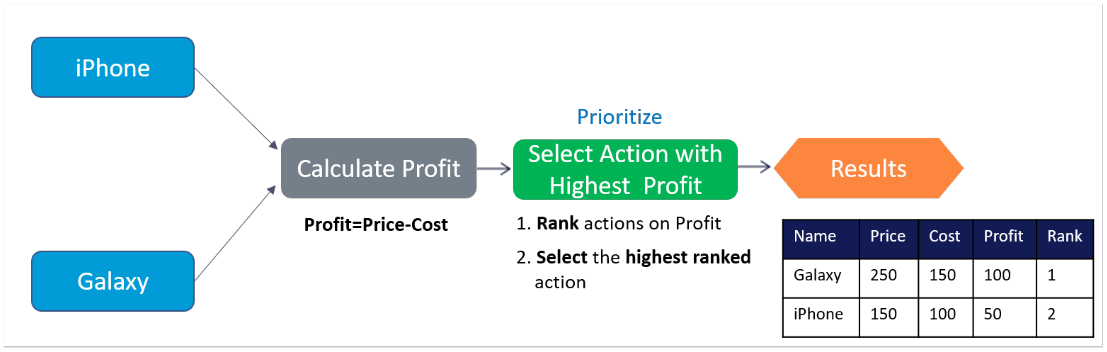
Once the actions are ranked, the prioritize component selects the highest ranked action. Click Select highest to see the action selected.


iPhone
Galaxy
Calculate Profit
Profit=Price-Cost
Prioritize
Select Action with Highest Profit
1. Rank actions on Profit
2. Select the highest ranked action
Results

Name	Price	Cost	Profit	Rank
iPhone	150	100	50	2
Galaxy	250	150	100	1


iPhone
Galaxy
Calculate Profit
Profit=Price-Cost
Prioritize
Select Action with Highest Profit
1. Rank actions on Profit
2. Select the highest ranked action
Results

Name	Price	Cost	Profit	Rank
iPhone	150	100	50	2
Galaxy	250	150	100	1

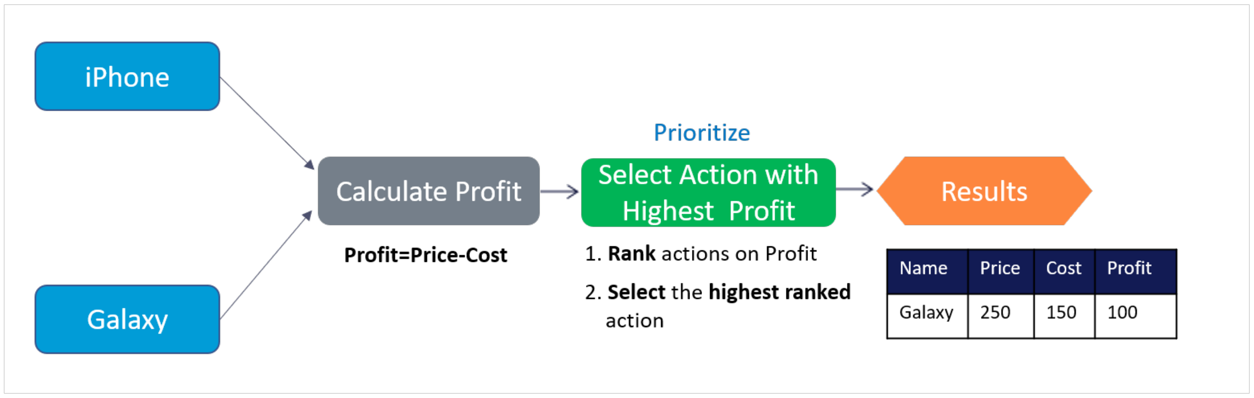

iPhone
Galaxy
Calculate Profit
Profit=Price-Cost
Prioritize
Select Action with Highest Profit
1. Rank actions on Profit
2. Select the highest ranked action
Results

Name	Price	Cost	Profit	Rank
Galaxy	250	150	100	1
iPhone	150	100	50	2

Select highest

Screen 4:

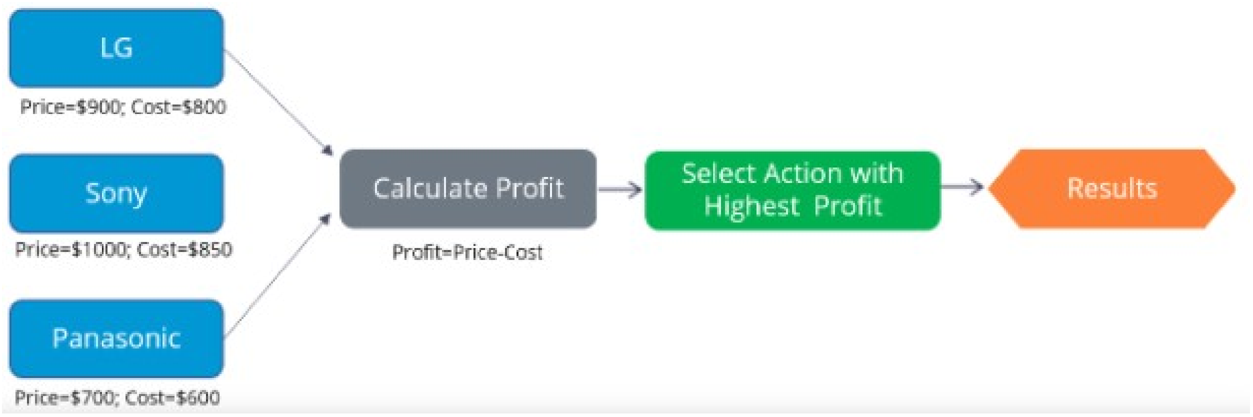
You can see that the output of the result component is the highest selected action: Galaxy with a profit of 100.


iPhone
Galaxy
Calculate Profit
Profit=Price-Cost
Prioritize
Select Action with Highest Profit
1. Rank actions on Profit
2. Select the highest ranked action
Results

Name	Price	Cost	Profit
Galaxy	250	150	100

Quiz

Examine this strategy and then answer the following question to check your knowledge on action ranking.



What does the Results component of the strategy contain?

Sony with profit 150

LG with profit 100

Panasonic with profit 50

Feedback: The Prioritize decision component ranks the actions and selects the highest ranked action. Hence, the Results component of the strategy contains Sony with a profit of 150.

Creating a decision strategy

Introduction

Decision Strategies drive Next-Best-Action. They comprise a unit of reasoning represented by decision components. How these components combine determines which action will be selected for a customer: the Next-Best-Action. Learn the type of decision components and how they are used to create decision strategies. Gain hands-on experience designing and executing your own Next-Best-Action decision strategy.

Transcript

This demo will show you how to create a new decision strategy.

It will also describe three important decision components and the types of properties available for use in expressions during strategy building.

In this demo you will build a Next-Best-Label strategy. The Next-Best-Label strategy is a sample strategy, used to illustrate the mechanics of a decision strategy.

Start by creating a new strategy from scratch.

Decision strategies output actions, utilizing the so-called Strategy-Results class.

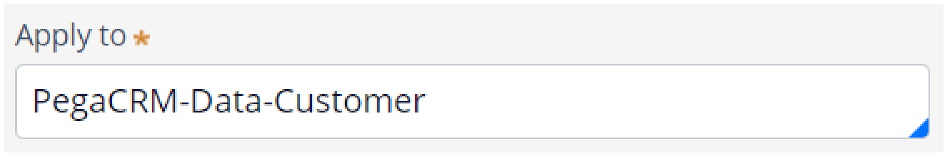
The Strategy-Results class limits the output of the strategy to the actions contained in the Business issue and Group.

The strategy you build will select a Label action from a set of predefined actions. The Label action selected will be the one with the lowest printing cost.

Notice that the complete definition of the Next-Best-Label strategy needs to include a reference to the PegaCRM-Data-Customer class.

This is the ‘Apply to’ class and it indicates the context of the strategy.

It ensures that from within the strategy, you have access to customer-related properties such as Age, Income, Address, Name, etc.


Apply to *

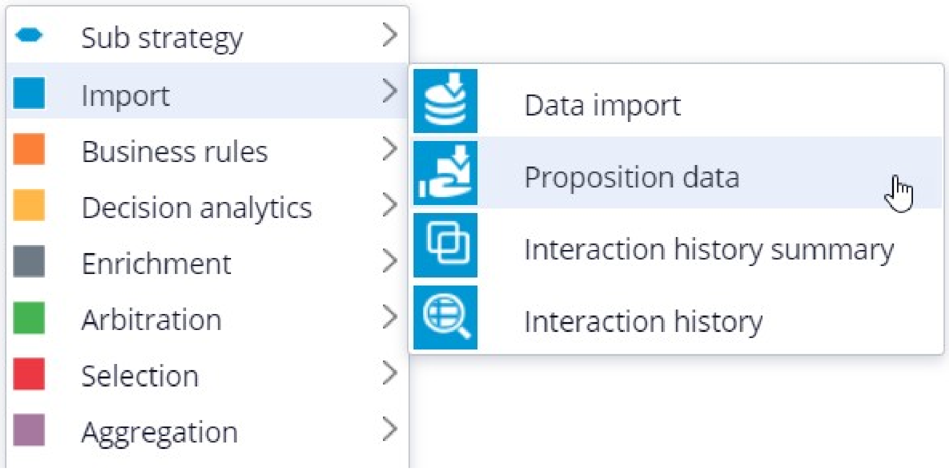
PegaCRM-Data-Customer

You can now start building the strategy. Right-click on the canvas to get the Context menu, which shows all component categories.

The first component to add is an Import component.

By expanding the Import category, you can see the Import component types available.

In this case you need a Proposition Data component to define the actions that will be considered by the strategy.



Now you need to configure the component. First, right-click to open the Proposition Data properties panel.

Notice that the Business issue and Group are grayed out.

This cannot be changed because the Enablement Business issue and Labels Group have already been selected for this decision strategy.

By default, the strategy will import all actions within that Group, unless you select a specific action.

For this component, you only want to import the Green Label, so let's select that.

Selecting the action from the drop-down menu automatically gives the component the appropriate name.

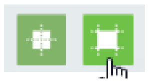
The description, which will appear under the component on the canvas, will also be generated automatically.

If you want to create your own description, you can do so by clicking the 'Use custom' radio button.

Now you want to import a second action into the strategy. You can use the Copy and Paste buttons to quickly add more Proposition Data components to the canvas.

You can use Alignment Snapping and Grid Snapping for easy placement of the components.

By turning these off, you can place a component anywhere on the canvas, but it makes it more difficult to align the shapes.



Now you need to add the next component in the strategy, which is an Enrichment component called Set Property.

You can add this component to the canvas by selecting it from the component menu.

Next, connect it to the Proposition Data components.

Ultimately, the result of this strategy should be the Label action with the lowest printing cost.

This printing cost is the sum of a base printing cost, which is specific to each label, and a variable cost, which depends on the number of letters.

The Set Property component is where you will calculate the printing cost for each of the actions.

The information in the ‘Source components’ tab is populated automatically by the Proposition Data components connected to this component.

Notice that the Black Label action is in the first row.

On the Target tab you can add properties for which values need to be calculated.

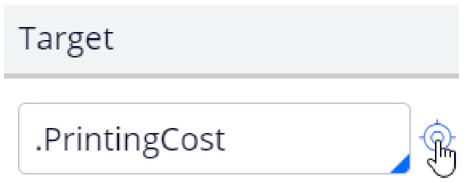
Click ‘Add Item’ to create the equation that will calculate the printing cost for each of the components.

Begin by setting the Target property to ‘dot’ PrintingCost.

In Pega, all inputs begin with a dot. This is called the dot-operator and it means that you are going to use a strategy property.

The PrintingCost property is a new strategy property that does not yet exist.

To create the new PrintingCost strategy property, click on the icon next to the Target field.



By default, the property type is Text. In Pega, there are various types supported. In this case, the PrintingCost is a numeric value, so change its type to Decimal.

Next, you need to make PrintingCost equal to the calculation you create. To create the calculation, click on the icon next to the Source field.

Using the Expression builder, you can create all sorts of complex calculations, but in this use case, the computation is very basic.

PrintingCost should equal BaseCost + 5 * LetterCount.

To access the BaseCost you type a dot. Notice that when you type the dot, a list of available and relevant strategy properties appears.

This not only makes it easy to quickly find the property names you're looking for; it also avoids spelling mistakes.

In a decision strategy, you have two categories of properties available to use in Expressions.

The first category contains the strategy properties, which can be one of two types.

An Action property is defined in the Action form. Examples are the BaseCost and LetterCount properties you are using here.

These properties have a value defined in the Action form and are available in the decision strategy via the Proposition Data component.

The property values can be overridden in the decision strategy but will often be used as read only.

The second type of strategy property is a calculation like the one you just created, PrintingCost. Such calculations are often created and set in the decision strategy.

These types of properties are either used as transient properties, for temporary calculations, or for additional information you want the strategy to output.

The second category contains properties from the strategy context, also called customer properties.

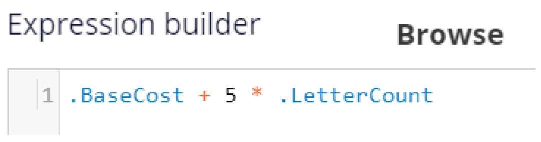
Suppose you want to use a customer property in your Expression, such as Age or Income.

In that case, you would have to type the prefix 'Customer dot', instead of just dot.

This is the list of available properties from the strategy context, also known as Customer properties.

For now, you calculate the printing cost for each action that does not use customer properties.

Finalize the Expression.

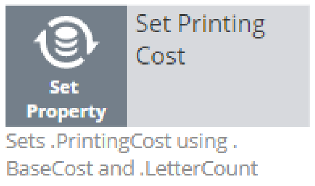


```
.BaseCost + 5 * .LetterCount
```

Even though you used the dot-operator to build your Expression, it's best practice to validate it, so click Test.

If the Expression isn't valid, you will receive an error message on screen.

On the canvas, you can see the automatically generated description for the component: Sets PrintingCost using BaseCost and LetterCount.



Sets .PrintingCost using .BaseCost and .LetterCount

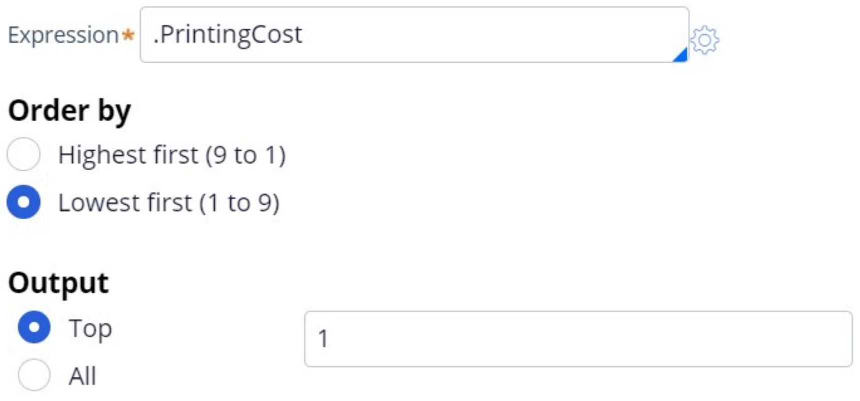
Now you want to ensure that the actions will be prioritized based on the lowest printing cost. So, you need to add the Prioritize component from the Arbitration category.

The prioritization can either be based on an existing property, or it can be based on an equation. Let's select an existing property using the dot construct.

Here you can select the order in which the top actions are presented. Since you are interested in the lowest printing costs, configure it accordingly.

You can also select the number of actions that will be returned by the strategy.

If you want to output only one label, select Top 1 here.



Expression* .PrintingCost

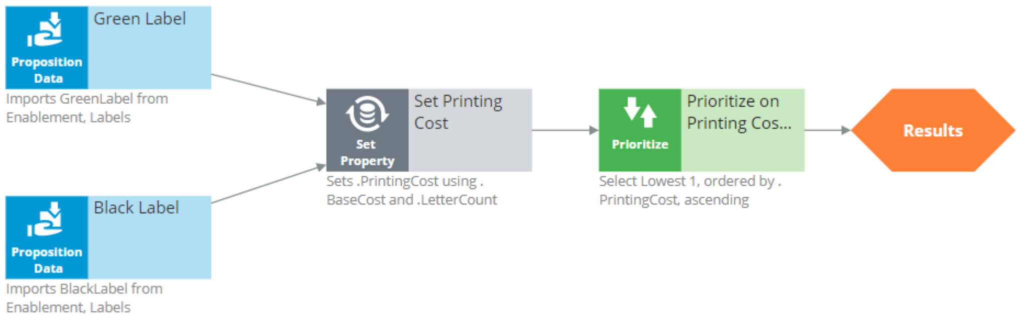
Order by

Highest first (9 to 1)
[✓ Lowest first (1 to 9)]

Output

[✓ Top] 1
All

Now you can connect the components and save the strategy.


Green Label
Proposition Data
Imports GreenLabel from Enablement, Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement, Labels
Set Printing Cost
Set Property
Sets .PrintingCost using .BaseCost and .LetterCount
Prioritize on Printing Cos...
Prioritize
Select Lowest 1, ordered by .PrintingCost, ascending
Results

To test the strategy, first check it out. Then, expand the right-hand side test panel and click 'Save & Run' to examine the results.

You can view results for any of the components by selecting that component.

If more than one action is present, each one is presented as a Page.

For the Set Property component, the Results contain a page for the Black Label and one for the Green Label.

For the Black Label the PrintingCost is 70.

For the Green Label the PrintingCost is 60.

On the canvas, you can show values for strategy properties such as Printing Cost.

For this exercise, you execute this strategy against a Data Transform called UseCase1.

If you open UseCase1, you can see the customer data the strategy uses when you run it.

To test the strategy on a different use case, you can create a Data Transform with different properties.

You can also select a Data Set that points to an actual live database table.

This demo has concluded. What did it show you?

- How to create a decision strategy from scratch.
- How to configure Proposition Data, Set Property and Prioritize decision components.
- How to build expressions in strategies.
- The two categories of properties available for expressions.
- How to test a decision strategy using a use case stored in a data transform.

Decision strategy execution

Introduction

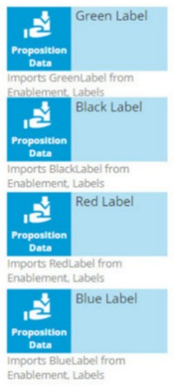
Using Pega Decision Management, you do not need to be an expert in programming, math or data science to design and execute sophisticated decision strategies that engage your customers throughout the customer journey. With its highly intuitive graphical canvas, Pega Decision Management enables you to easily embed Pega or third-party predictive models into your decision strategies. The result is customer-centric interactions that improve the customer experience while increasing customer value, retention and response rates.

Transcript

This demo explains what's going on inside each component when a Decision Strategy is executed.

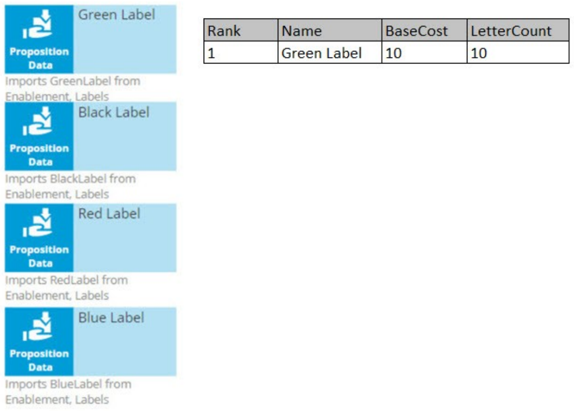
For example, what happens 'under the covers' when a Filter component is executed, and how does it interact with the components around it?

In the interest of keeping it simple, this example is limited to four actions. In reality, decision strategies will involve many more actions than that.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Proposition Data
Red Label
Imports RedLabel from Enablement, Labels
Proposition Data
Blue Label
Imports BlueLabel from Enablement, Labels

Here are our 4 actions: 'Green Label', 'Black Label', 'Red Label' and 'Blue Label'; they are represented by a Data Import or, more specifically, a Proposition Data component.

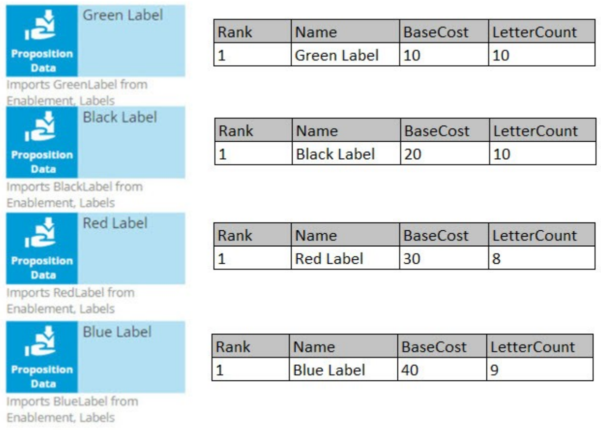
In this example, the Proposition Data components import three data properties for each action: Name, BaseCost and LetterCount.


Green Label
Proposition Data
Imports GreenLabel from Enablement, Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement, Labels
Red Label
Proposition Data
Imports RedLabel from Enablement, Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement, Labels

Rank	Name	BaseCost	LetterCount
1	Green Label	10	10

The first action's Name is Green Label, its BaseCost is 10, and its LetterCount is 10.

Likewise, the other actions have a Name, BaseCost and LetterCount.


Green Label
Proposition Data
Imports GreenLabel from Enablement, Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement, Labels
Red Label
Proposition Data
Imports RedLabel from Enablement, Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement, Labels

Rank	Name	BaseCost	LetterCount
1	Green Label	10	10

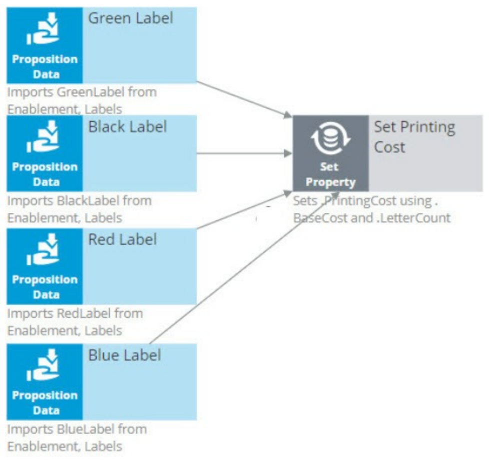
Rank	Name	BaseCost	LetterCount
1	Black Label	20	10

Rank	Name	BaseCost	LetterCount
1	Red Label	30	8

Rank	Name	BaseCost	LetterCount
1	Blue Label	40	9

One property is automatically populated for you; this is the Rank. We will come back to this later, but notice that, as separate components, each action has a Rank of 1.

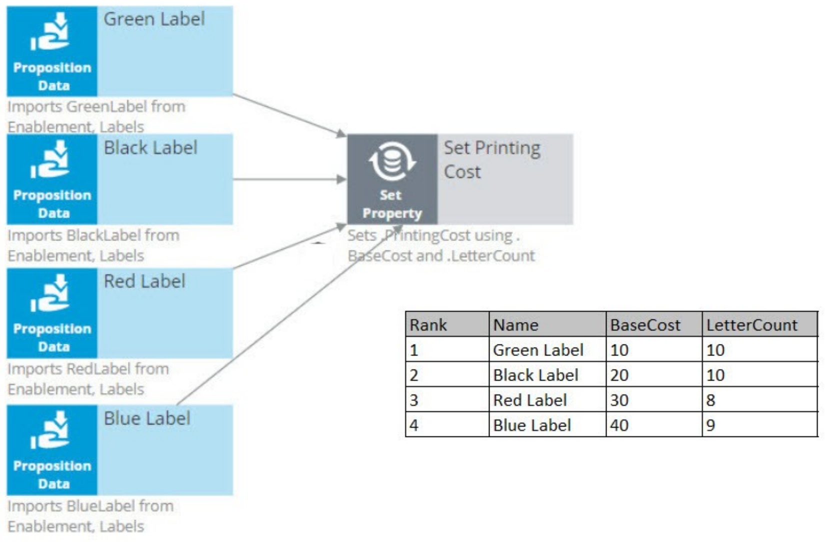
On the strategy canvas, components are connected by drawing arrows from component to component. So, what do these arrows mean exactly?


Green Label
Proposition Data
Imports GreenLabel from Enablement. Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement. Labels
Red Label
Proposition Data
Imports RedLabel from Enablement. Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement. Labels
Set Printing Cost
Set Property
Sets .PrintingCost using .BaseCost and .LetterCount

Well, when you draw an arrow, what happens is that, at runtime, all information in the component you're drawing the arrow from is available as a data source to the component you're drawing the arrow to.

So now, the Name, BaseCost and LetterCount for all of the actions are available in a single Set Property component.

The only data element that changes is the row number, or as we call it in the strategies, the Rank. In each decision component, the Rank value is automatically computed.


Green Label
Proposition Data
Imports GreenLabel from Enablement. Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement. Labels
Red Label
Proposition Data
Imports RedLabel from Enablement. Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement. Labels
Set Printing Cost
Set Property
Sets .PrintingCost using .BaseCost and .LetterCount

Rank	Name	BaseCost	LetterCount
1	Green Label	10	10
2	Black Label	20	10
3	Red Label	30	8
4	Blue Label	40	9

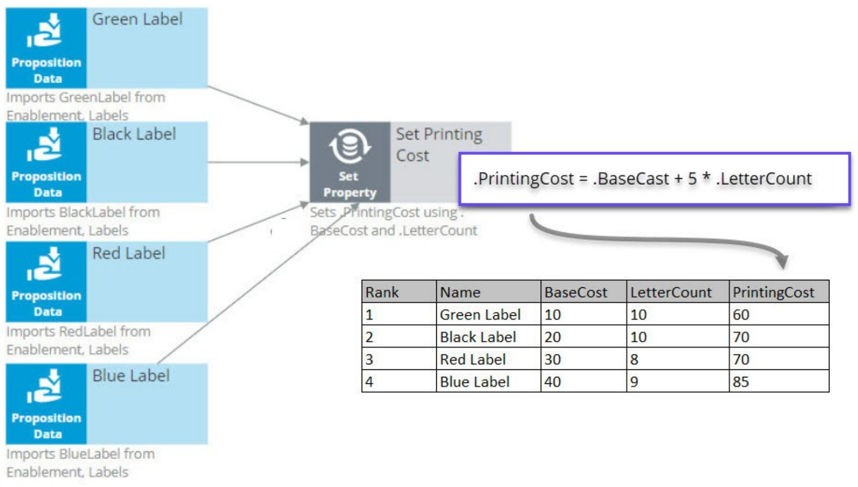
In the Set Property component, the Rank is determined by the order in which the actions are received by the component.

As a result, in this instance, the Green Label action has a Rank of 1, Black has a Rank of 2, Red has a Rank of 3, and Blue has a Rank of 4.

Ultimately, you want to select the best Label action. That is the Label with the lowest printing cost.

The printing cost of a Label is the sum of the BaseCost and a variable cost based on the LetterCount.

You configure the Set Property component to compute the printing cost of each Label action.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Proposition Data
Red Label
Imports RedLabel from Enablement, Labels
Proposition Data
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
.PrintingCost = .BaseCast + 5 * .LetterCount

Rank	Name	BaseCost	LetterCount	PrintingCost
1	Green Label	10	10	60
2	Black Label	20	10	70
3	Red Label	30	8	70
4	Blue Label	40	9	85

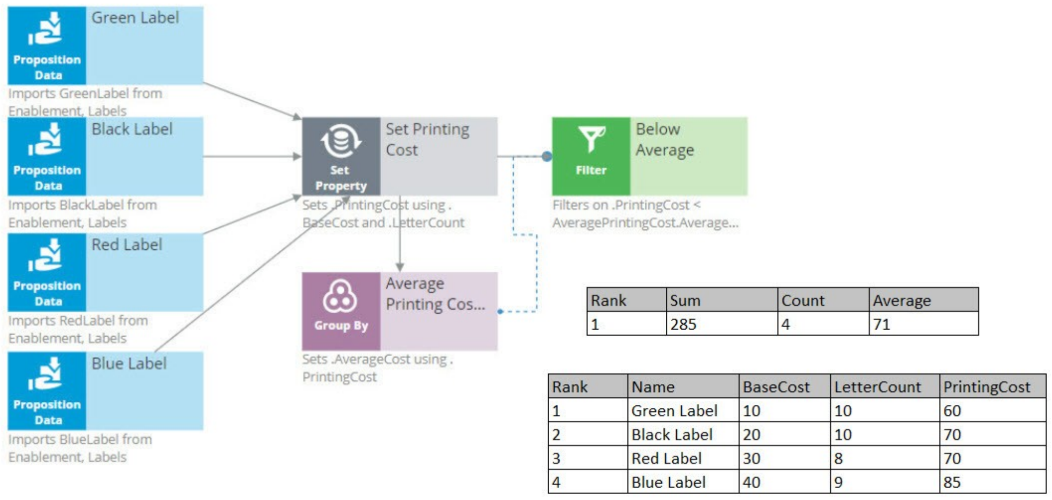
Because we are combining the data in our four Proposition Data components into one Set Property component, we only need to add one PrintingCost property to the new component, and it automatically computes the printing cost for all four actions.

For the Green Label action, PrintingCost equals a BaseCost of 10 plus 5 times the LetterCount of 10 which equals 60.

Similarly, the PrintingCost for the Black and Red Label actions is 70, and for the Blue Label action is 85.

Now, let's say the business rule is to select only Label actions with a printing cost lower than the average printing cost of all labels. For this requirement we use a 'Group by'/Filter component combination.

A 'Group by' component offers essential aggregation capabilities, like Sum and Count, that are used in many decision strategies. We will use it to calculate the average printing cost.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Black Label
Imports BlackLabel from Enablement, Labels
Red Label
Imports RedLabel from Enablement, Labels
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...

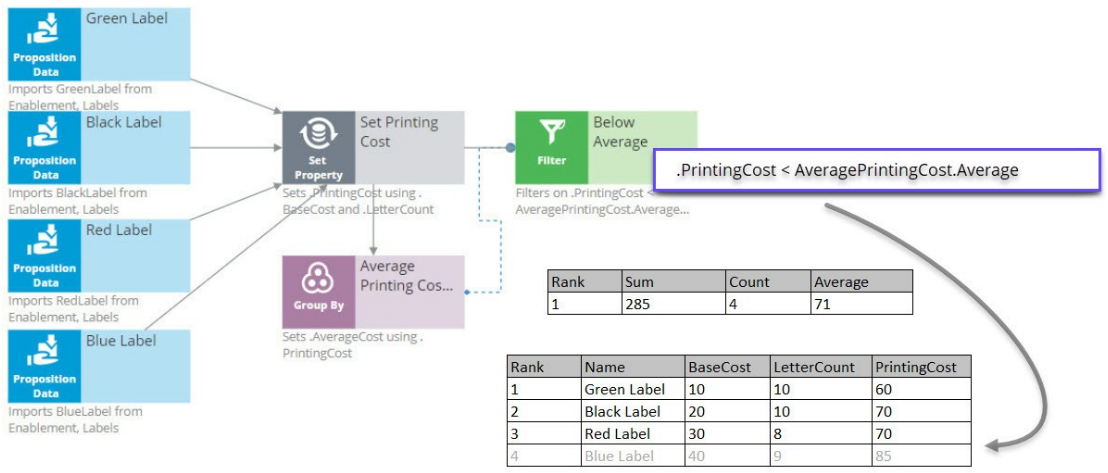
Rank	Sum	Count	Average
1	285	4	71

Rank	Name	BaseCost	LetterCount	PrintingCost
1	Green Label	10	10	60
2	Black Label	20	10	70
3	Red Label	30	8	70
4	Blue Label	40	9	85

Again, we have our set of actions, each with their own specific PrintingCost value. The ‘Group by’ component combines all actions into one row. How does that work?

Well, it sums the PrintingCost values for all the actions, it counts the actions, and it calculates the average printing cost by dividing the summed printing cost by the count.

In this example, the sum of the PrintingCost values is 285, and the count of the actions is 4, so the average printing cost is 71.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Black Label
Imports BlackLabel from Enablement, Labels
Red Label
Imports RedLabel from Enablement, Labels
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
.PrintingCost < AveragePrintingCost.Average

Rank	Sum	Count	Average
1	285	4	71

Rank	Name	BaseCost	LetterCount	PrintingCost
1	Green Label	10	10	60
2	Black Label	20	10	70
3	Red Label	30	8	70
4	Blue Label	40	9	85

Now that you have calculated the average printing price using a ‘Group by’ component, configure the Filter component to filter out actions that have a printing cost equal to or higher than this average.

So far in this strategy, we’ve seen only the solid line arrows, which copy information from one component to another. But now we also see a dotted line arrow.

This tells us that a component refers to information in another component.

Here, the Filter component is referencing the average printing cost that exists inside the Aggregation component. This is an important capability to understand.

The Filter component filters out actions when the printing cost for that action is equal to or above the average printing cost and propagates the other actions.

First, via the solid arrow, the filter looks at the actions sourced from the Set Property component.

Then, it applies the filter condition, which references the average printing cost in the 'Group by' component via the dotted arrow.

The Filter Condition in the Filter component is the Expression: 'dot PrintingCost is smaller than AveragePrintingCost dot Average'.

By using this ComponentName dot Property construct, any decision component can be referenced by any other component by name.

Important to note that the Filter component lets actions through when the condition Expression evaluates to **true** and filters out actions when the condition Expression is not met.

When you refer to a component, you always refer to the first element in the component, the one with Rank 1.

In this case, you are referring to the one and only row in the 'Group by' component, which naturally has Rank 1.

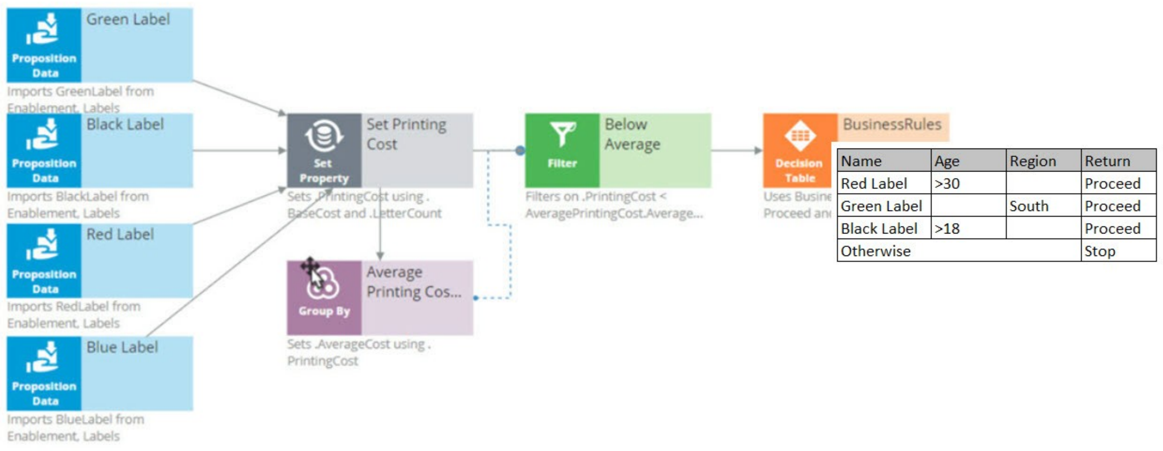
The Rank 1 average equals 71 in the 'Group by' component. This means that the filter will allow Label actions through that have a printing cost lower than 71.

By this standard, the printing cost of the Blue Label action is too high, so it is filtered out. The printing cost of the other Label actions are below 71, so they survive.

The result is that the table contains three surviving actions: Green Label with Rank 1, Black Label with Rank 2, and Red Label with Rank 3.

The next component is a Decision Table. A Decision Table in Pega is an artifact that can be used to implement business requirements in table format.

In a Decision Table, the business rules are represented by a set of conditions and a set of Return values.


Green Label
Proposition Data
Imports GreenLabel from Enablement. Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement. Labels
Red Label
Proposition Data
Imports RedLabel from Enablement. Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement. Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
Decision Table
BusinessRules
Uses Busine
Proceed an

Name	Age	Region	Return
Red Label	>30		Proceed
Green Label		South	Proceed
Black Label	>18		Proceed
Otherwise			Stop

The Decision Table receives information about the remaining actions via the solid arrow from the Filter component.

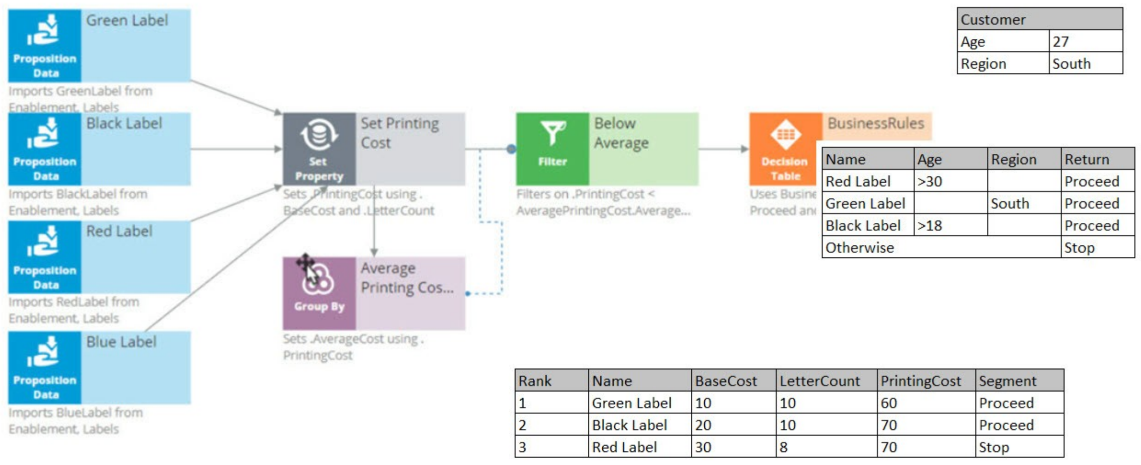
The business criteria say that the Red Label action can be offered if the customer's age is over 30 and they are from any region. If these criteria are met, the Return value is 'Proceed'.

The Decision Table also says that the Green Label action can be offered to anyone in the Southern region. So, if the Region value is South, the Return value for Green is 'Proceed'.

The Black Label action can be offered to anyone over the age of 18.

But in all other cases, or, Otherwise, no Label action meets the criteria, and the Return value is 'Stop'.

As an example, consider a customer with Age 27 and Region South.


Green Label
Proposition Data
Imports GreenLabel from Enablement. Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement. Labels
Red Label
Proposition Data
Imports RedLabel from Enablement. Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement. Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
Decision Table
BusinessRules
Uses Busine
Proceed an

Customer	
Age	27
Region	South

Name	Age	Region	Return
Red Label	>30		Proceed
Green Label		South	Proceed
Black Label	>18		Proceed
Otherwise			Stop

Rank	Name	BaseCost	LetterCount	PrintingCost	Segment
1	Green Label	10	10	60	Proceed
2	Black Label	20	10	70	Proceed
3	Red Label	30	8	70	Stop

Now, the Decision Table applies the business criteria for each action against the customer information and returns a value. The value returned by a Decision Table is also called a Segment.

The Decision Table checks the Green Label action with Rank 1 first, and in this case, it can proceed because the customer's Region is South.

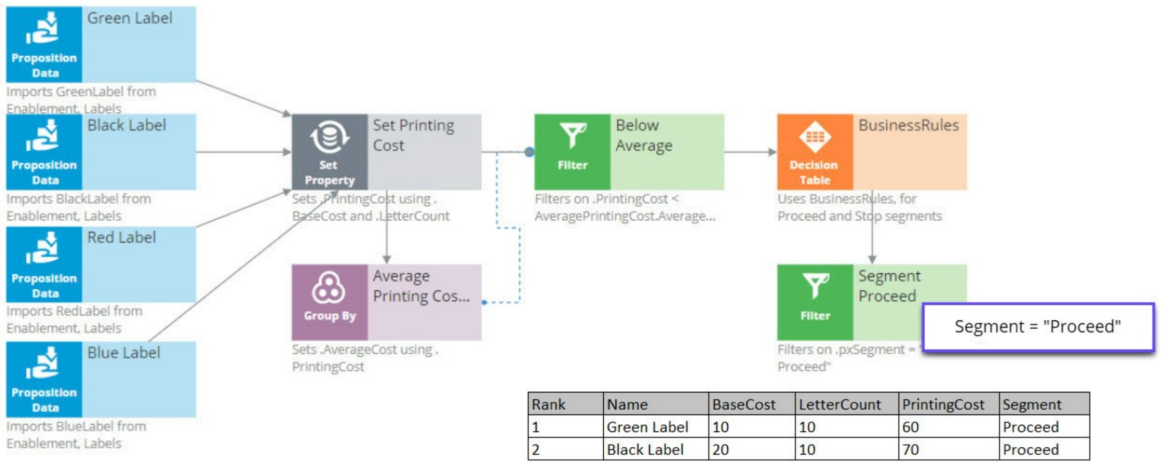
Next, it looks at the Black action and sees that the criteria for Black is that the customer's age is greater than 18. This customer is 27.

Black doesn't care about the Region, so the Segment value for the Black action is 'Proceed'.

Finally, it looks at the Red action, and the Age criteria don't match up, so the Segment value for Red is 'Stop'.

The result of the component is that you get a new segmentation column that flags which of the actions comply with the business rules.

You're now going to filter out the actions that do not match the business rules. This happens in the 'Segment Proceed' Filter component.



Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Proposition Data
Red Label
Imports RedLabel from Enablement, Labels
Proposition Data
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
Decision Table
BusinessRules
Uses BusinessRules, for Proceed and Stop segments
Filter
Segment Proceed
Filters on .pxSegment = Proceed"
Segment = "Proceed"

Rank	Name	BaseCost	LetterCount	PrintingCost	Segment
1	Green Label	10	10	60	Proceed
2	Black Label	20	10	70	Proceed

Again, via the solid arrow, the strategy copies the data over from the Decision Table component into the Filter component.

Now each action has a Rank, Name, BaseCost, LetterCount, PrintingCost and Segment. The filter condition is applied to this data.

The filter condition says: allow this action through if the Segment value equals 'Proceed'.

What this Filter component now does is go through the list of actions to find the actions with value 'Proceed' in their Segment property.

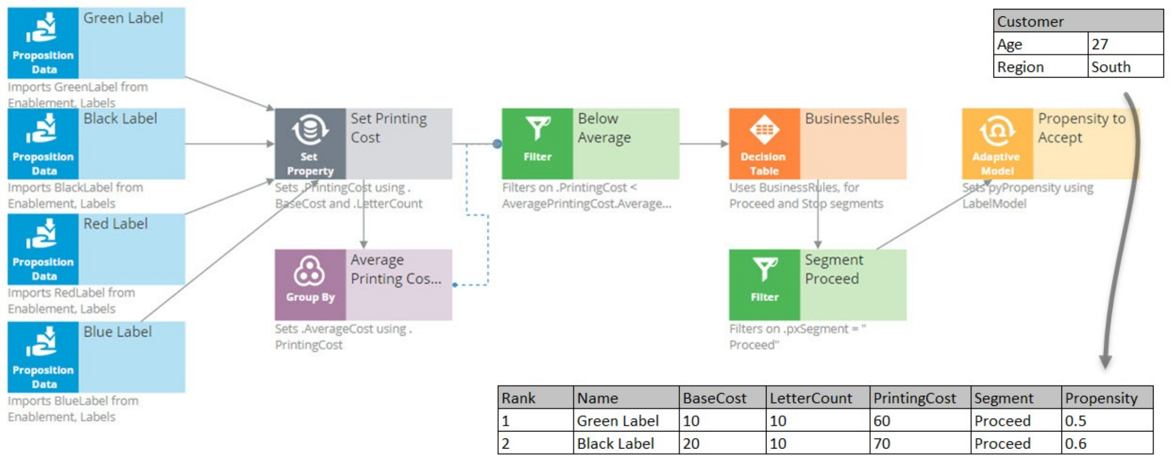
First is the Green Label. Green is allowed through, which means its properties will be available in the new component.

Then the Black Label. It is also allowed through because it also has ‘Proceed’ in its Segment property.

But the Red Label action is not allowed through, because Red has ‘Stop’ in its Segment property. Therefore, Red is not part of the output.

The strategy so far has selected two of our original actions, Green and Black.

Now, in the Adaptive Model component, you will use predictive analytics to determine the propensity of each of the remaining actions.


Green Label
Proposition Data
Imports GreenLabel from Enablement, Labels
Black Label
Proposition Data
Imports BlackLabel from Enablement, Labels
Red Label
Proposition Data
Imports RedLabel from Enablement, Labels
Blue Label
Proposition Data
Imports BlueLabel from Enablement, Labels
Set Printing Cost
Set Property
Sets .PrintingCost using .BaseCost and .LetterCount
Average Printing Cos...
Group By
Sets .AverageCost using .PrintingCost
Below Average
Filter
Filters on .PrintingCost < AveragePrintingCost.Average...
BusinessRules
Decision Table
Uses BusinessRules, for Proceed and Stop segments
Segment Proceed
Filter
Filters on .pxSegment = "Proceed"
Propensity to Accept
Adaptive Model
Sets .pyPropensity using LabelModel

Customer	
Age	27
Region	South

Rank	Name	BaseCost	LetterCount	PrintingCost	Segment	Propensity
1	Green Label	10	10	60	Proceed	0.5
2	Black Label	20	10	70	Proceed	0.6

Propensity is the probability that a customer will accept an action, or, their likelihood of interest in it.

In order to calculate the propensity, we use an Adaptive Model component. The referenced model is configured to monitor customer characteristics such as Age and Region.

In this case our test customer has an Age of 27 and is from the South Region.

Again, just to keep it simple, we are using a model that makes predictions based on only this information. In reality, models will take into account many more properties.

The Adaptive Model determines the propensity.

First, we supply the action and the customer profile to the Adaptive Model, and the model says: ‘Oh, it’s the Green Label action; we have some evidence that young people like the Green Label action, but people from the South don’t like it.’

Combining both factors, we get an overall propensity of 0.5 for the Green Label action.

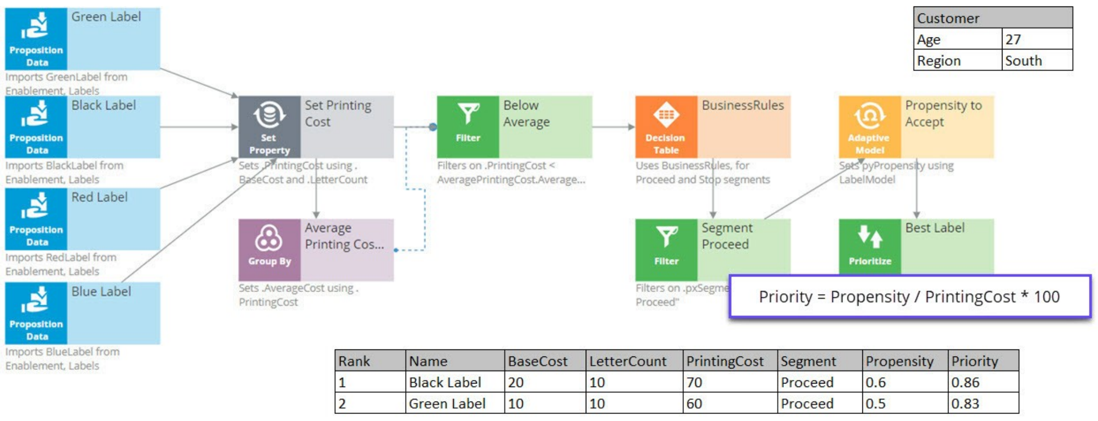
For the Black Label action, the likelihood turns out to be 0.6.

After consulting the Adaptive Model, the Propensity to Accept component sets the Propensity property value for each action.

Remember, the propensity is always a number between zero and 1.

It shows something along the lines of, half of the customers that are like this customer accepted the Green Label action in the past, and 3 out of 5 customers like this customer accepted the Black action last month.

The next component in our chain, called Best Label, is the Prioritize component. This component determines the priority of each action and ranks them. Let’s see how this works.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Proposition Data
Red Label
Imports RedLabel from Enablement, Labels
Proposition Data
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using .PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
Decision Table
BusinessRules
Uses BusinessRules, for Proceed and Stop segments
Filter
Segment Proceed
Filters on .pxSegme
Proceed"
Adaptive Model
Propensity to Accept
Sets pyPropensity using LabelModel
Prioritize
Best Label
Priority = Propensity / PrintingCost * 100

Customer	
Age	27
Region	South

Rank	Name	BaseCost	LetterCount	PrintingCost	Segment	Propensity	Priority
1	Black Label	20	10	70	Proceed	0.6	0.86
2	Green Label	10	10	60	Proceed	0.5	0.83

A key element of this component is the priority Expression, which calculates a priority value for each action. According to this Expression, the higher the value, the higher the priority and rank.

In this case, the priority calculation weighs likelihood of acceptance in its equation: ‘Propensity divided by PrintingCost times 100’.

When performing this calculation on the Black Label action, we can see that it has a PrintingCost of 70 and a Propensity of 0.6, therefore its Priority is 0.86.

The Green Label action has a lower PrintingCost and a lower Propensity, resulting in a Priority of 0.83.

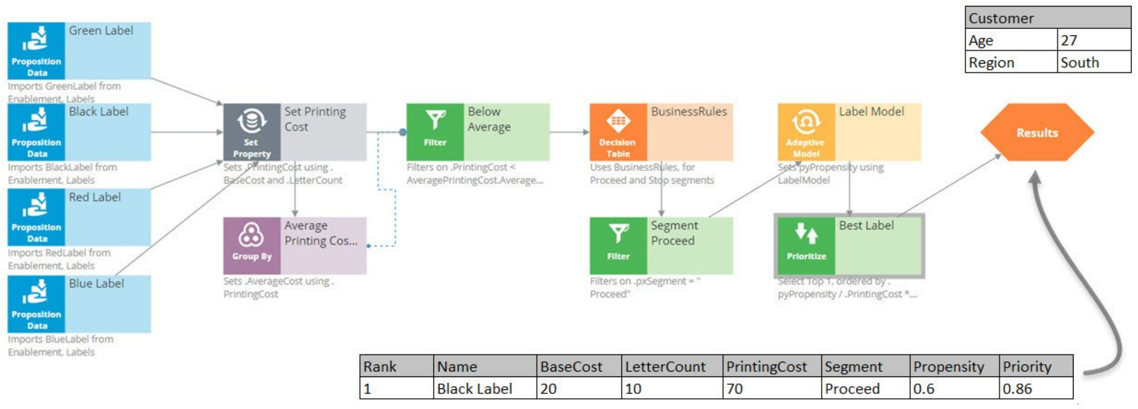
Because 0.86 is higher than 0.83, the Black Label action is now ranked number one.

So, even though the printing cost of the Black Label action is higher than that of the Green Label action, the Black Label action still comes out on top.

In this case, the Priority component reversed the Ranks of the two actions. Black is now the primary action and Green is the secondary action.

The same Prioritization component is also configured to output only the top action.

Therefore, it filters out the Green action altogether, and at the end of our strategy chain, the Black Label is left as our best action.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Proposition Data
Red Label
Imports RedLabel from Enablement, Labels
Proposition Data
Blue Label
Imports BlueLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using . BaseCost and .LetterCount
Group By
Average Printing Cos...
Sets .AverageCost using . PrintingCost
Filter
Below Average
Filters on .PrintingCost < AveragePrintingCost.Average...
Decision Table
BusinessRules
Uses BusinessRules, for Proceed and Stop segments
Filter
Segment Proceed
Filters on .pxSegment = " Proceed"
Adaptive Model
Label Model
Sets pyPropensity using LabelModel
Prioritize
Best Label
Select Top 1, ordered by . pyPropensity / .PrintingCost *...
Results

Customer	
Age	27
Region	South

Rank	Name	BaseCost	LetterCount	PrintingCost	Segment	Propensity	Priority
1	Black Label	20	10	70	Proceed	0.6	0.86

Creating engagement strategies using customer credit score

Description

A scorecard is a mathematical model that determines a score value based on a set of conditions and a combiner function. The conditions either use customer properties directly, or an expression based on them. The output of a scorecard is the raw score plus a segment, which is obtained by defining a set of cut off values that create a set of score ranges. Learn how a scorecard can be used to determine the customer credit score, and how the credit score can be used in a decision strategy to define suitability rules.

Learning Objectives

- Create a scorecard
- Use the segmentation result and the score value of a scorecard in a decision strategy
- Use a scorecard to determine group-level and action-level suitability

Customer credit score using a scorecard

Introduction

A scorecard is a mathematical model that determines a score value based on a set of conditions and a combiner function. These conditions can either use customer properties directly, or an expression based on them. The output of a scorecard is the raw score and a segment, which is obtained by defining a set of cut-off values that create a set of score ranges.

Transcript

This video explains how a scorecard can be used to determine the customer credit score.

Let’s consider a typical loan application scenario for U+ Bank.

The bank wants to automate the home loan requests process by analyzing a customer's credit score based on customer profile information and categorizing it into the right credit score level using a scorecard.

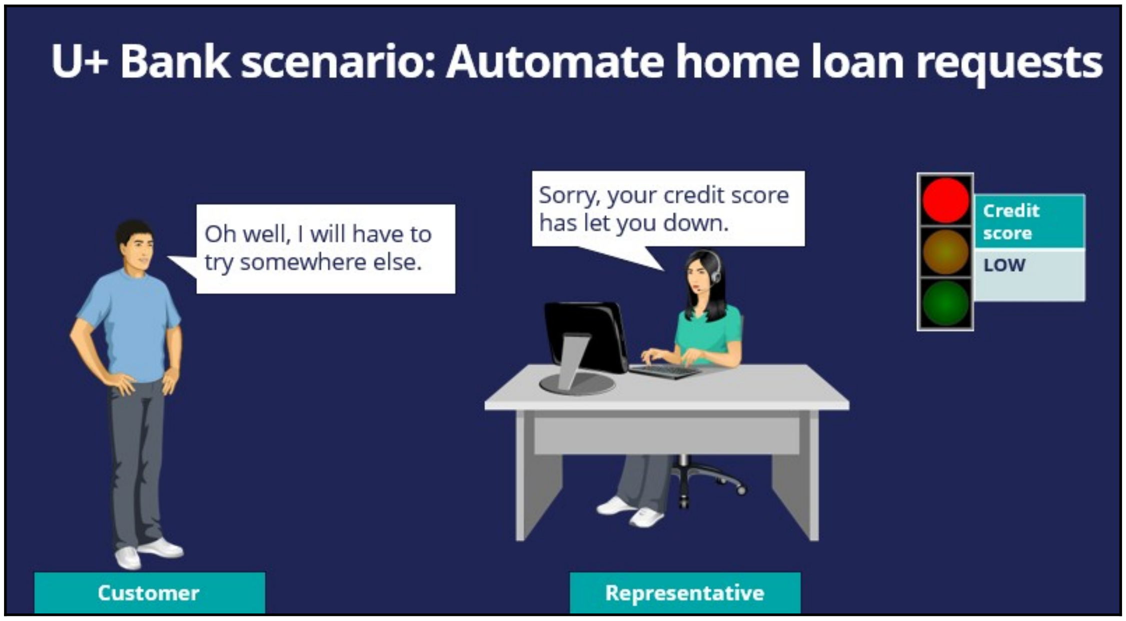
Tom is a U+ Bank customer who visits the bank to apply for a home loan. Iris is a U+ bank loan representative who is responsible for processing loan requests.

Iris gets Tom’s profile information. Based on his customer data, Iris calculates his credit score using a scorecard.

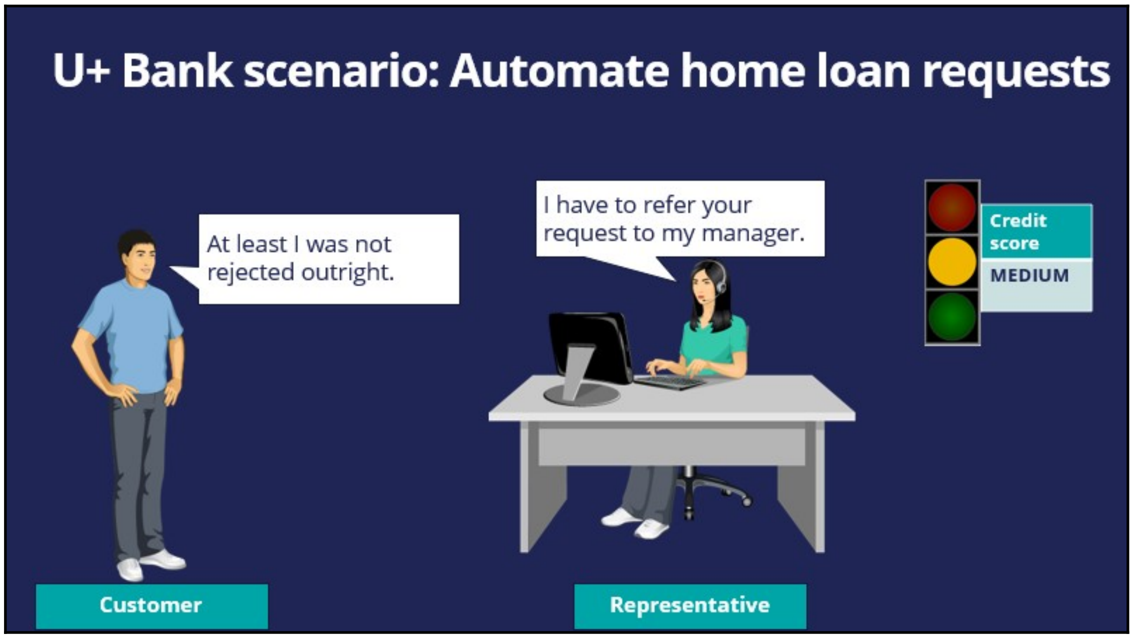
Depending on Tom’s credit score, a decision is made on his loan application.

The bank has already defined three credit score levels: **Low**, **Medium** and **High**.

For example, if Tom’s credit score is **LOW**, he is not qualified to get a loan, and his application is automatically refused.

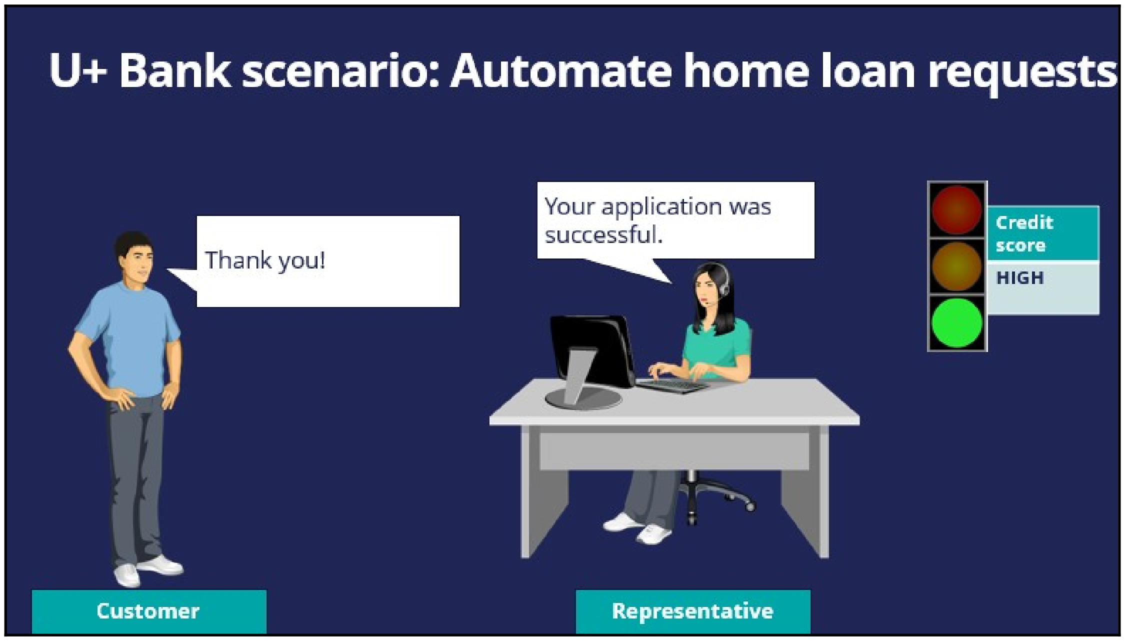

U+ Bank scenario: Automate home loan requests
Oh well, I will have to try somewhere else.
Sorry, your credit score has let you down.
Credit score
LOW
Customer
Representative

If Tom's credit score is **MEDIUM**, then his application needs to be reviewed further by manager.

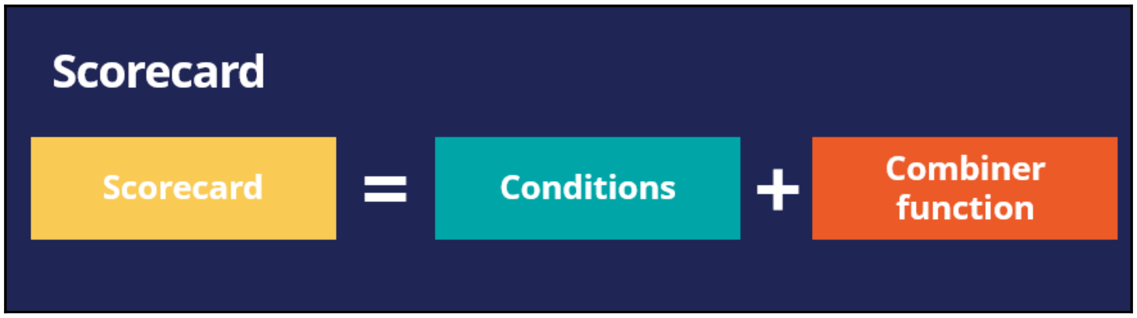

U+ Bank scenario: Automate home loan requests
At least I was not rejected outright.
I have to refer your request to my manager.
Credit score
MEDIUM
Customer
Representative

If Tom's credit score is **HIGH,** then he is automatically eligible to get the loan right away.

Let's now learn about the scorecard and how it works by considering the U+ Bank loan application scenario.


U+ Bank scenario: Automate home loan requests
Thank you!
Your application was successful.
Credit score
HIGH
Customer
Representative

A scorecard is a mathematical model that computes a score and outputs a segmentation result based on one or more conditions and a combiner function.



Let's take a look at an example of a scorecard U+ bank could use. Assume that in order to process home loan requests, the bank has identified **three predictors** that can be used in the scorecard calculation. The predictors are **Customer lifetime value, Age,** and **Account**.

Each predictor has some **conditions** and scores associated with those conditions.

The scorecard enables the bank to assign a **score** to a value or a range of values the predictor can have.

For example, If the **Customer lifetime value** is less than 100, a score of 83 is assigned. If the value is between 100 and 399, a score of 221 is assigned; otherwise a score of 350 is assigned.

If the customers’ Age is less than 20, a score of 21 is assigned. If their Age is between 20 and 39, a score of 217 is assigned; otherwise a score of 55 is assigned.

In a similar way, if a customer has an account with U+ bank, a score of 82 is assigned. Otherwise, a score of 10 is assigned.

The combiner function is used to combine the total sum of score values between conditions.

Now that the bank has identified **predictors** and assigned a **score** to a range of values the predictor has, let’s see the results of the scorecard.

The output of a scorecard is a **score** and a **segment result**.

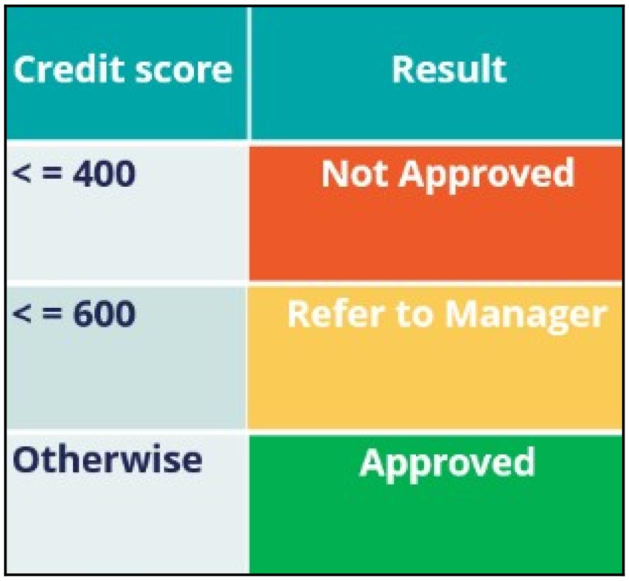
First, you will see how to calculate a customer’s credit score using the scorecard.

Let’s consider an example in which the customer lifetime value is 350, the customer’s age is 35, and the customer has a U+ Bank account.

Since the customer lifetime value is 350, he gets a score of 221. For his age, he gets a score of 217, and since he has a U+ bank account, he gets a score of 82. The sum of these three individual scores is 520, representing his final credit score.

The scorecard outputs a **score** as well as a **segment result.**

To process loan requests, the bank has defined three result values. If the credit score is less than or equal to 400, the result is **Not Approved.** If the credit score value is between 401 and 600, the result is **Refer to Manager.** If the credit score is higher than 600, the result is **Approved**. So, the higher a customer’s credit score, the better his/her chances of getting a loan approved.



Credit score	Result
< = 400	Not Approved
< = 600	Refer to Manager
Otherwise	Approved

Let’s consider the profiles of three customers applying for home loans.

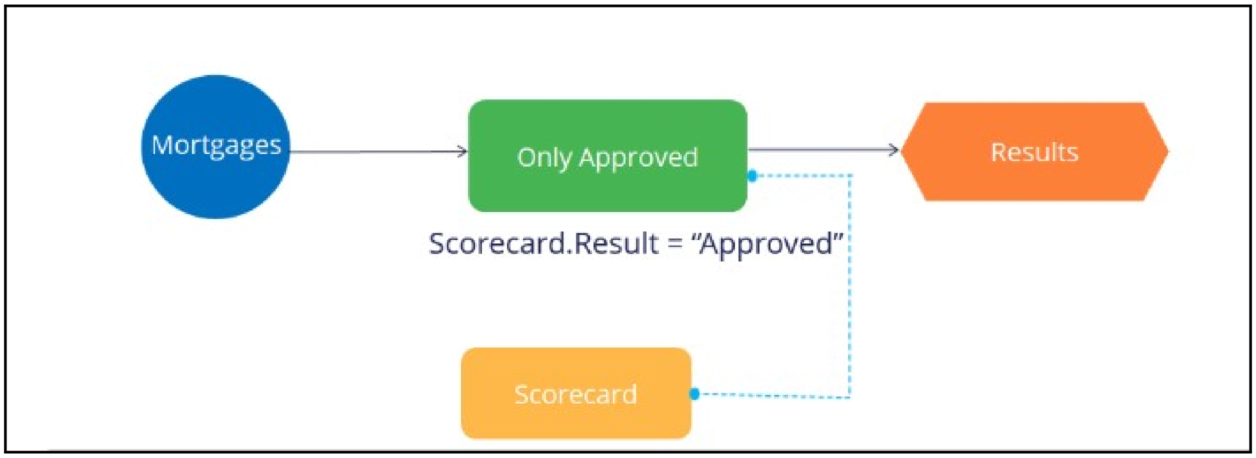
Customer A has a customer lifetime value of 150, their age is 18, and they don’t have a U+ Bank account. Therefore, their credit score is 252, and the segment result is **Not Approved**, because their credit score value is less than 400. This means they do not qualify for a loan.

Customer B has a different profile, and their credit score is 415, so the segment result is **Refer to Manager**, because their credit score value is between 401 and 600. This means their application needs further review.

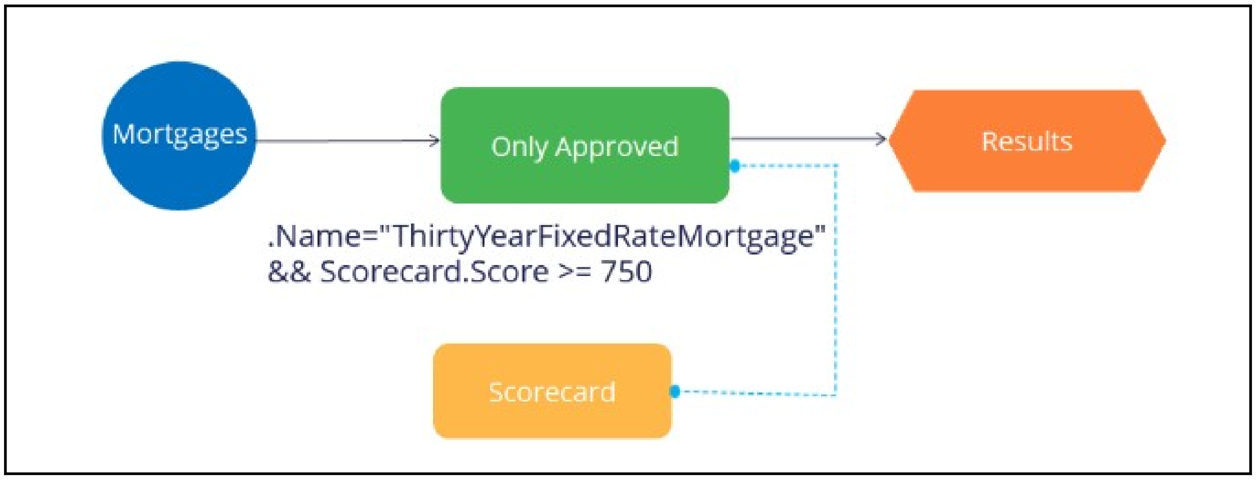
Customer C has a credit score of 649, which is higher than 600, so he is eligible to get a loan right away.

Now that you learned about the scorecard and its results, let’s see how to use the scorecard segmentation in a decision strategy.

To use a Scorecard in a decision strategy, use the **Scorecard** component. The **Scorecard** component outputs the result, in this case **Not Approved, Refer to Manager**, or **Approved**, which you can use in your decision strategy.



The scorecard also outputs the **raw score value**, this score value can be used directly in the decision strategy.


Mortgages
Only Approved
Results
.Name="ThirtyYearFixedRateMortgage" && Scorecard.Score >= 750
Scorecard

In summary, the scorecard is a mathematical model that computes a score and outputs a segmentation result based on a set of conditions and a combiner function. The scorecard's score and segmentation result can be used directly in a decision strategy using the Scorecard decision component.

Building a scorecard to calculate the creditscore

Introduction

Scorecard rules are referenced in decision strategies through the scorecard component. Learn how to create scorecard-specific rules to derive decision results from several factors.

Get detailed insight into how scores are calculated by testing scorecard logic using the rule form. Use the test results to view score explanations for all predictors used in the calculation so you can validate and refine the current scorecard design or troubleshoot potential issues.

Transcript

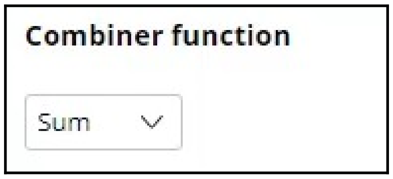
U+ bank wants to build a scorecard that calculates the credit score of a customer to determine if the customer is suitable for a mortgage or not.

The bank has identified three customer properties to use in the scorecard calculation. These are the HasCards (a property that indicates if a customer already has a credit card or not), Income, and Age.

To build a scorecard, navigate to the scorecards landing page and create a scorecard.

Enter a short description for the scorecard.

The Combiner function enables you to select a method for combining scores.



Combiner function

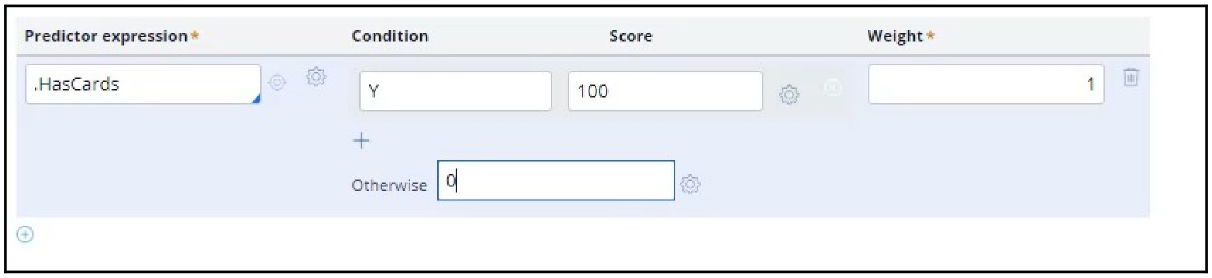
Sum

In this scenario, the credit score is the sum of scores attributed to each customer property, so use the Sum method.

The scorecard enables you to assign a score to a value or a range of values the property can have.

For example, the credit card indicator, HasCards property can have the values “Y” or “N”.

If a customer has a credit card with U+ bank, assign a score of 100. Otherwise, assign a score of 0.



Predictor expression*	Condition	Score	Weight*
.HasCards	Y	100	1
	Otherwise	0	

You may also sub-divide values of numeric properties into ranges and assign a score to each range.

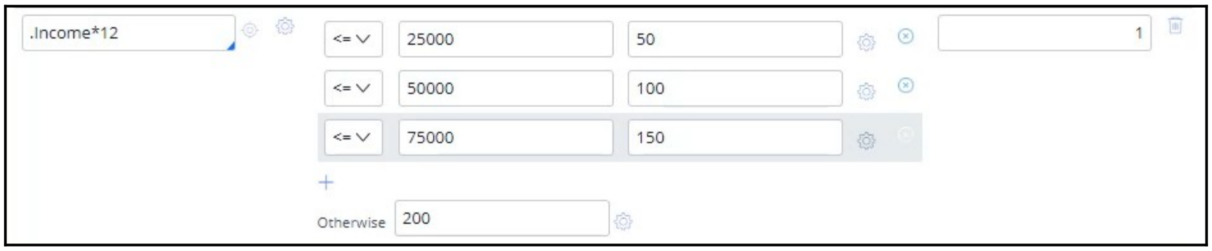
The next property is Income. In this case you want to split the values for yearly income into three ranges and assign a score to each range.

The data model contains an Income property, which contains the monthly income of the customer. The scorecard allows you to use this property to build the expression to compute the customer’s yearly income.

Once the expression is created, you can start assigning scores to each range. If the customer's yearly income is less than or equal to 25,000, assign a score of 50.

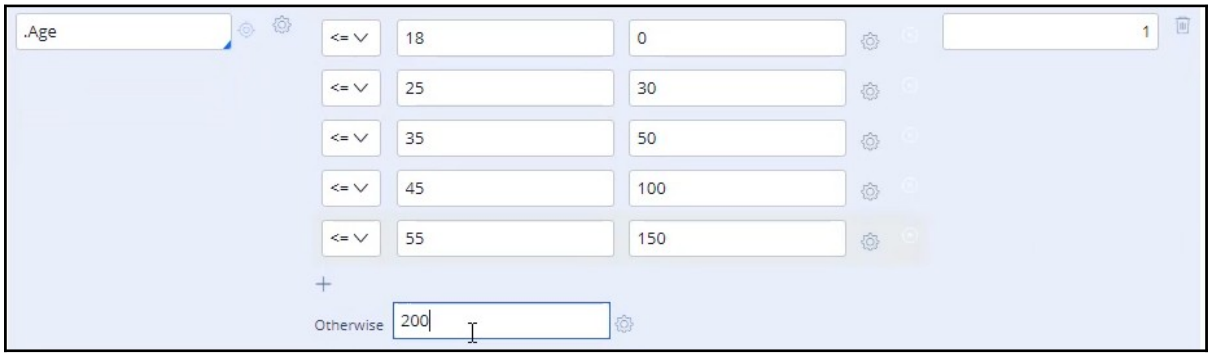
If their yearly income is between 25,001 and 50,000, assign a score of 100; and if their income is between 50,001 and 75,000, assign a score of 150.

If their income is higher than 75,000, assign a score of 200, using the **Otherwise** setting.



.Income*12	<=	25000	50	1
	<=	50000	100	
	<=	75000	150	
	Otherwise		200	

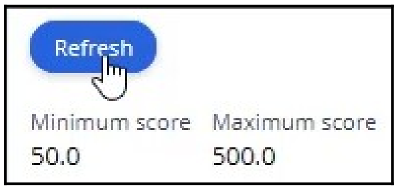
In a similar way, split the values for Age into five ranges and assign a score to each range. The higher the range, the higher the score you assign.



.Age	<=	18	0	1
	<=	25	30	
	<=	35	50	
	<=	45	100	
	<=	55	150	
	Otherwise	200		

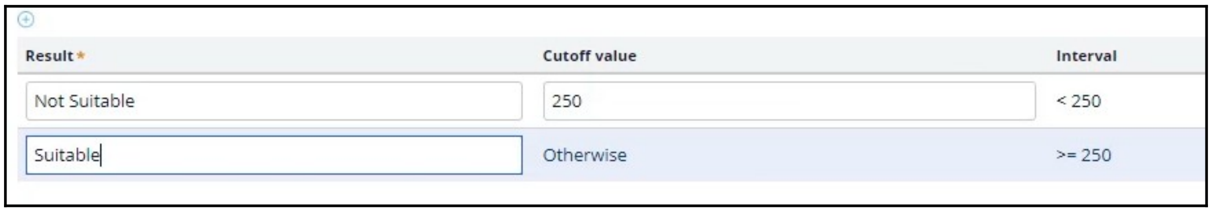
Once the Scorecard is defined, you can define the results of the scorecard calculation.

When you refresh, the scorecard calculates the Minimum and Maximum scores.



Minimum score	Maximum score
50.0	500.0

You may define two or more Result values based on the score. In this scenario, you want the scorecard to return a Result value of 'Not Suitable' if the score is less than 250. Otherwise, the Result value is 'Suitable'.



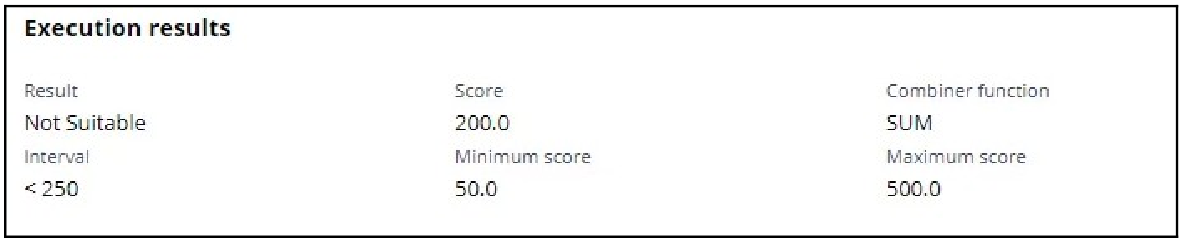
Result *	Cutoff value	Interval
Not Suitable	250	< 250
Suitable	Otherwise	>= 250

Save the configuration.

Now, run the scorecard and verify the results.

You can either fill in the property values, or you can test the result for a predefined test case using a data transform. For example, select the customer Troy.

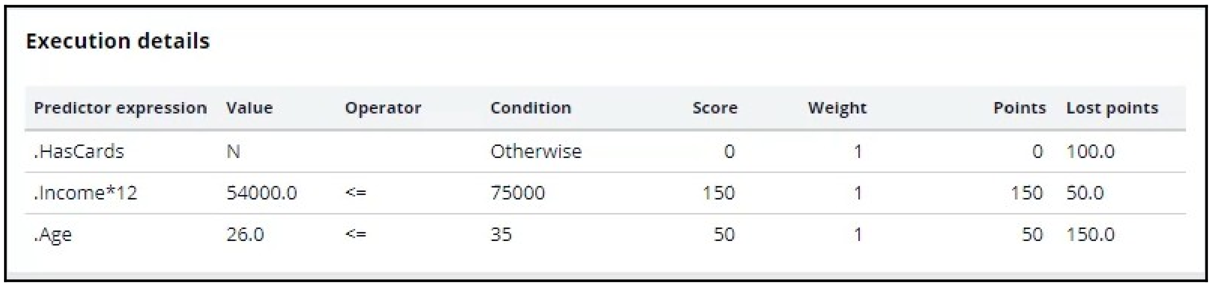
The end result for Troy is that he is Not Suitable for a mortgage, as his credit score is 200, which is lower than the set threshold.



Execution results

Result	Score	Combiner function
Not Suitable	200.0	SUM
Interval	Minimum score	Maximum score
< 250	50.0	500.0

You can see the **Execution details** for Troy, which show how his credit score is computed. Since he has no cards, he gets a score of 0; for his income, he gets a score of 150; and for his age, he gets a score of 50. That's 200 in total.

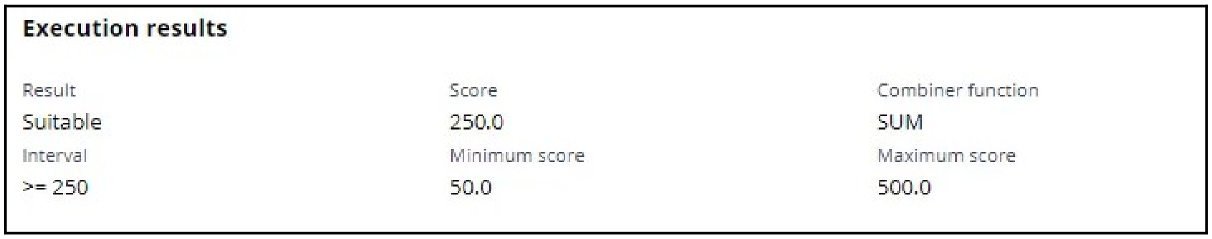


Execution details

Predictor expression	Value	Operator	Condition	Score	Weight	Points	Lost points
.HasCards	N		Otherwise	0	1	0	100.0
.Income*12	54000.0	<=	75000	150	1	150	50.0
.Age	26.0	<=	35	50	1	50	150.0

Now, test the results for customer Robert.

Robert is suitable for a mortgage, as his credit score is 250.



Execution results

Result	Score	Combiner function
Suitable	250.0	SUM
Interval	Minimum score	Maximum score
>= 250	50.0	500.0

The newly created scorecard is now available on the Scorecards landing page.

This demo has concluded. What did it show you?

- How to create a scorecard.

- How to assign a score to categorical and numerical property values.

- How to use expressions in scorecards.

- How to define scorecard outputs.

- How to test scorecards.

Creating a decision strategy

Introduction

Decision Strategies drive Next-Best-Action. They comprise a unit of reasoning represented by decision components. How these components combine determines which action will be selected for a customer: the Next-Best-Action. Learn the type of decision components and how they are used to create decision strategies. Gain hands-on experience designing and executing your own Next-Best-Action decision strategy.

Transcript

This demo will show you how to create a new decision strategy.

It will also describe three important decision components and the types of properties available for use in expressions during strategy building.

In this demo you will build a Next-Best-Label strategy. The Next-Best-Label strategy is a sample strategy, used to illustrate the mechanics of a decision strategy.

Start by creating a new strategy from scratch.

Decision strategies output actions, utilizing the so-called Strategy-Results class.

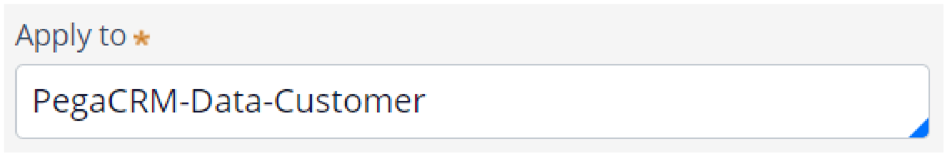
The Strategy-Results class limits the output of the strategy to the actions contained in the Business issue and Group.

The strategy you build will select a Label action from a set of predefined actions. The Label action selected will be the one with the lowest printing cost.

Notice that the complete definition of the Next-Best-Label strategy needs to include a reference to the PegaCRM-Data-Customer class.

This is the ‘Apply to’ class and it indicates the context of the strategy.

It ensures that from within the strategy, you have access to customer-related properties such as Age, Income, Address, Name, etc.



Apply to *

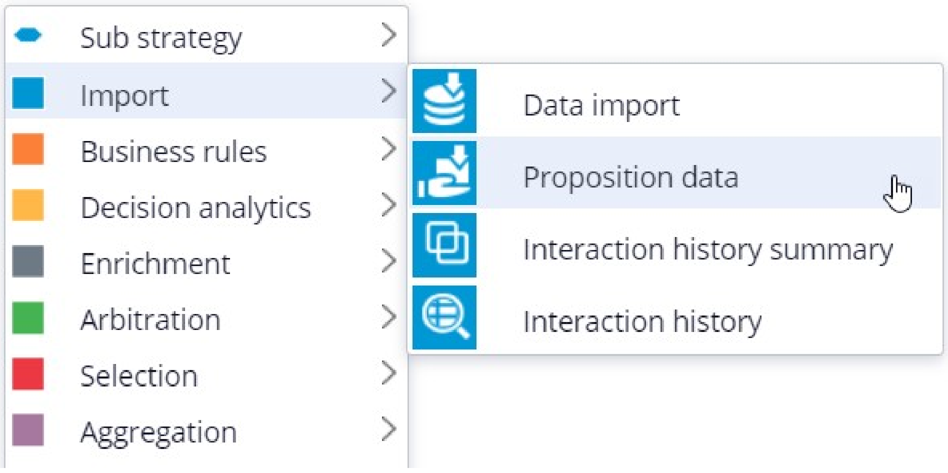
PegaCRM-Data-Customer

You can now start building the strategy. Right-click on the canvas to get the Context menu, which shows all component categories.

The first component to add is an Import component.

By expanding the Import category, you can see the Import component types available.

In this case you need a Proposition Data component to define the actions that will be considered by the strategy.



Now you need to configure the component. First, right-click to open the Proposition Data properties panel.

Notice that the Business issue and Group are grayed out.

This cannot be changed because the Enablement Business issue and Labels Group have already been selected for this decision strategy.

By default, the strategy will import all actions within that Group, unless you select a specific action.

For this component, you only want to import the Green Label, so let's select that.

Selecting the action from the drop-down menu automatically gives the component the appropriate name.

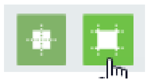
The description, which will appear under the component on the canvas, will also be generated automatically.

If you want to create your own description, you can do so by clicking the 'Use custom' radio button.

Now you want to import a second action into the strategy. You can use the Copy and Paste buttons to quickly add more Proposition Data components to the canvas.

You can use Alignment Snapping and Grid Snapping for easy placement of the components.

By turning these off, you can place a component anywhere on the canvas, but it makes it more difficult to align the shapes.



Now you need to add the next component in the strategy, which is an Enrichment component called Set Property.

You can add this component to the canvas by selecting it from the component menu.

Next, connect it to the Proposition Data components.

Ultimately, the result of this strategy should be the Label action with the lowest printing cost.

This printing cost is the sum of a base printing cost, which is specific to each label, and a variable cost, which depends on the number of letters.

The Set Property component is where you will calculate the printing cost for each of the actions.

The information in the 'Source components' tab is populated automatically by the Proposition Data components connected to this component.

Notice that the Black Label action is in the first row.

On the Target tab you can add properties for which values need to be calculated.

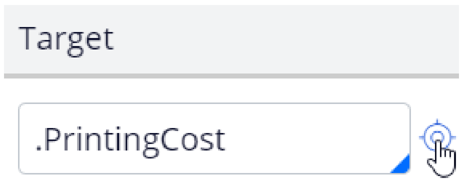
Click 'Add Item' to create the equation that will calculate the printing cost for each of the components.

Begin by setting the Target property to 'dot' PrintingCost.

In Pega, all inputs begin with a dot. This is called the dot-operator and it means that you are going to use a strategy property.

The PrintingCost property is a new strategy property that does not yet exist.

To create the new PrintingCost strategy property, click on the icon next to the Target field.



Target

.PrintingCost

By default, the property type is Text. In Pega, there are various types supported. In this case, the PrintingCost is a numeric value, so change its type to Decimal.

Next, you need to make PrintingCost equal to the calculation you create. To create the calculation, click on the icon next to the Source field.

Using the Expression builder, you can create all sorts of complex calculations, but in this use case, the computation is very basic.

PrintingCost should equal BaseCost + 5 * LetterCount.

To access the BaseCost you type a dot. Notice that when you type the dot, a list of available and relevant strategy properties appears.

This not only makes it easy to quickly find the property names you're looking for; it also avoids spelling mistakes.

In a decision strategy, you have two categories of properties available to use in Expressions.

The first category contains the strategy properties, which can be one of two types.

An Action property is defined in the Action form. Examples are the BaseCost and LetterCount properties you are using here.

These properties have a value defined in the Action form and are available in the decision strategy via the Proposition Data component.

The property values can be overridden in the decision strategy but will often be used as read only.

The second type of strategy property is a calculation like the one you just created, PrintingCost. Such calculations are often created and set in the decision strategy.

These types of properties are either used as transient properties, for temporary calculations, or for additional information you want the strategy to output.

The second category contains properties from the strategy context, also called customer properties.

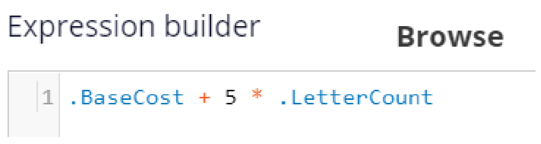
Suppose you want to use a customer property in your Expression, such as Age or Income.

In that case, you would have to type the prefix 'Customer dot', instead of just dot.

This is the list of available properties from the strategy context, also known as Customer properties.

For now, you calculate the printing cost for each action that does not use customer properties.

Finalize the Expression.

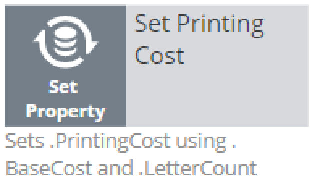


```
.BaseCost + 5 * .LetterCount
```

Even though you used the dot-operator to build your Expression, it's best practice to validate it, so click Test.

If the Expression isn't valid, you will receive an error message on screen.

On the canvas, you can see the automatically generated description for the component: Sets PrintingCost using BaseCost and LetterCount.



Sets .PrintingCost using .BaseCost and .LetterCount

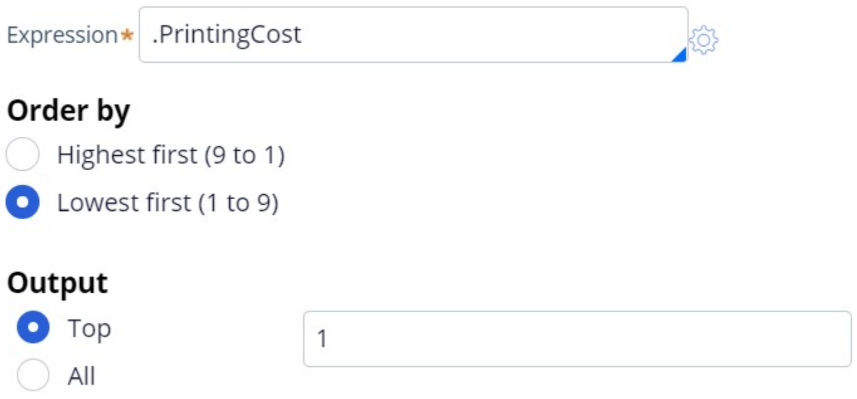
Now you want to ensure that the actions will be prioritized based on the lowest printing cost. So, you need to add the Prioritize component from the Arbitration category.

The prioritization can either be based on an existing property, or it can be based on an equation. Let's select an existing property using the dot construct.

Here you can select the order in which the top actions are presented. Since you are interested in the lowest printing costs, configure it accordingly.

You can also select the number of actions that will be returned by the strategy.

If you want to output only one label, select Top 1 here.



Expression* .PrintingCost

Order by

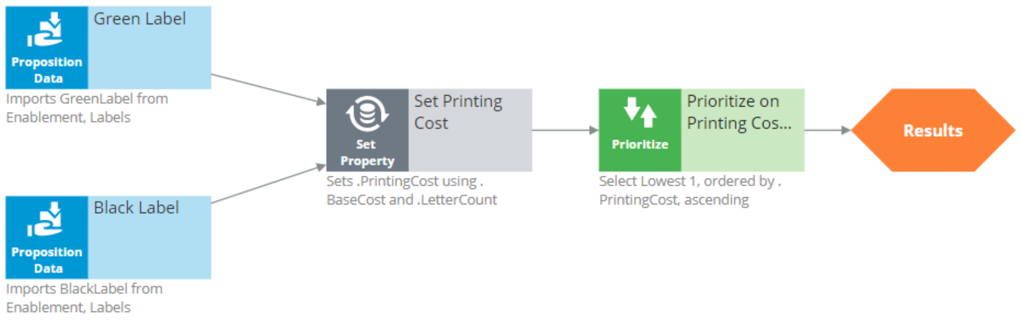
Highest first (9 to 1) [✓ Lowest first (1 to 9)]

Output

[✓ Top] 1

All

Now you can connect the components and save the strategy.


Proposition Data
Green Label
Imports GreenLabel from Enablement, Labels
Proposition Data
Black Label
Imports BlackLabel from Enablement, Labels
Set Property
Set Printing Cost
Sets .PrintingCost using .BaseCost and .LetterCount
Prioritize
Prioritize on Printing Cos...
Select Lowest 1, ordered by .PrintingCost, ascending
Results

To test the strategy, first check it out. Then, expand the right-hand side test panel and click 'Save & Run' to examine the results.

You can view results for any of the components by selecting that component.

If more than one action is present, each one is presented as a Page.

For the Set Property component, the Results contain a page for the Black Label and one for the Green Label.

For the Black Label the PrintingCost is 70.

For the Green Label the PrintingCost is 60.

On the canvas, you can show values for strategy properties such as Printing Cost.

For this exercise, you execute this strategy against a Data Transform called UseCase1.

If you open UseCase1, you can see the customer data the strategy uses when you run it.

To test the strategy on a different use case, you can create a Data Transform with different properties.

You can also select a Data Set that points to an actual live database table.

This demo has concluded. What did it show you?

- How to create a decision strategy from scratch.
- How to configure Proposition Data, Set Property and Prioritize decision components.
- How to build expressions in strategies.
- The two categories of properties available for expressions.
- How to test a decision strategy using a use case stored in a data transform.

Using a Scorecard in a Decision Strategy

Introduction

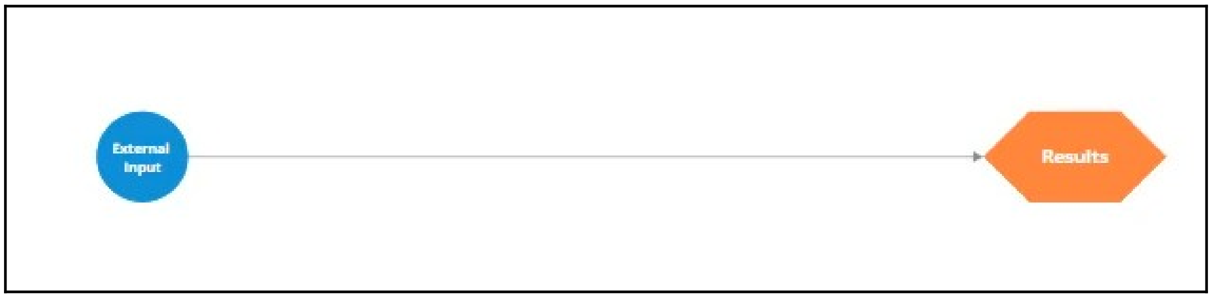
Learn how to use the segmentation result and the score value from a scorecard in a decision strategy. Learn how suitability rules can be defined to reflect the bank's requirements using a decision strategy.

Transcript

Currently, U+ Bank is doing cross-sell on the web by showing various credit cards to its customers.

The bank wants to implement a new requirement: All credit cards are suitable only for customers who have a credit score greater than or equal to 250.

To implement this requirement, use an Engagement Strategy.



U+ already created a Scorecard that computes the customer credit score. To use the Scorecard in the decision strategy, add a Scorecard component to the canvas and configure it.

Select the Scorecard model **DeterminеCreditScore**, which U+ already created**.**

If you open this Scorecard, you can see how the credit score is computed. Note that the 3 customer properties are used and a score is assigned to the value or range of values the property can have.

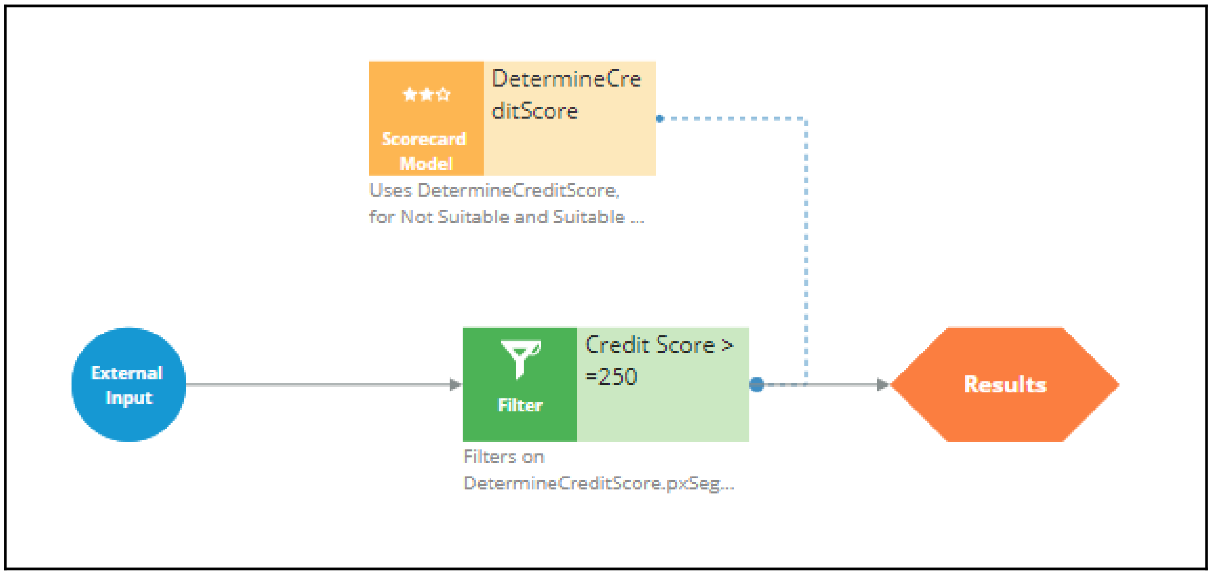
You can also see the segment results. The Scorecard returns value **Not Suitable** if the credit score is less than 250. Otherwise, it outputs the result **Suitable**.

To implement the new requirement, use a Filter component to express the condition under which the Actions are suitable.

You want this strategy to output something only if the result of the scorecard is “Suitable”. The result of the Scorecard is stored in the pxSegment property. Therefore, set the Filter condition to DetermineCreditScore.pxSegment==”Suitable”.

If the result of the Scorecard is Not Suitable, this strategy has no results.

Note that to reference a property from a component that is not connected to the Filter component, use the <ComponentName>.<PropertyName> construct.

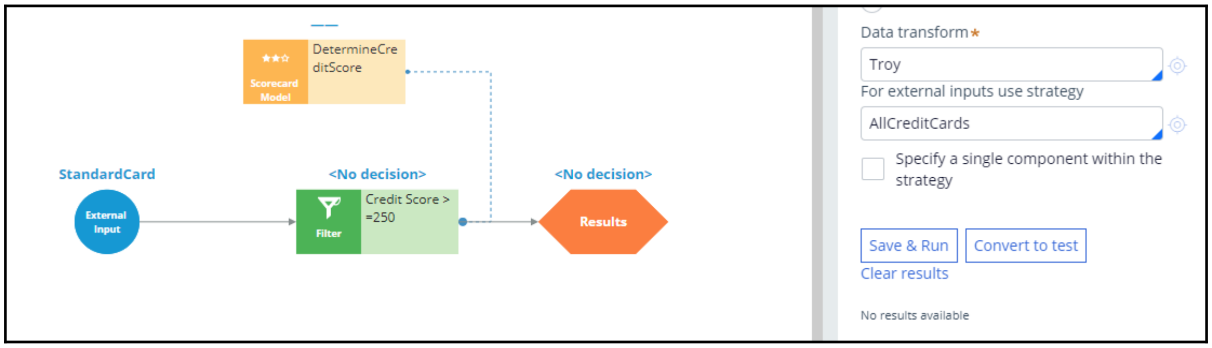

DetermineCreditScore
Scorecard Model
Uses DetermineCreditScore, for Not Suitable and Suitable ...
External Input
Filter
Credit Score >=250
Filters on DetermineCreditScore.pxSeg...
Results

Note that the usage of the External Inputs component also gives you the ability to create more complex conditions, where Action attributes are also used.

Save the configurations.

Now, test the strategy using the customer profiles, Troy and Robert.
For external inputs, consider all credit cards available.

The result of the Scorecard is: **Not Suitable**. Therefore, the strategy did not output any results for Troy.

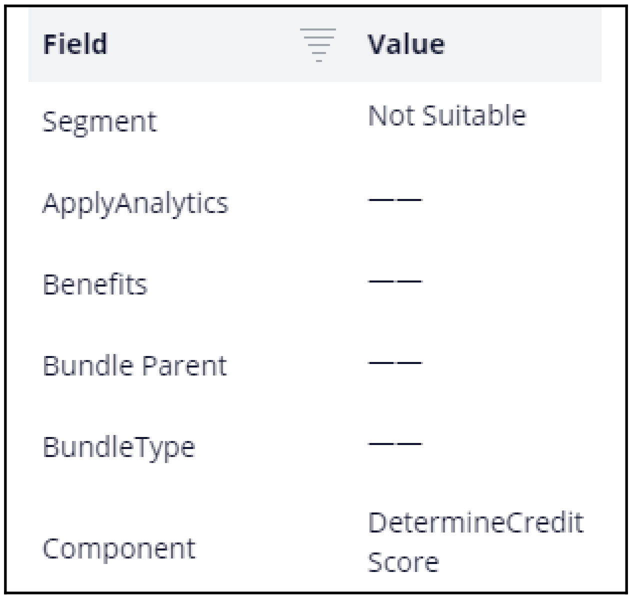


Data transform* Troy

For external inputs use strategy AllCreditCards

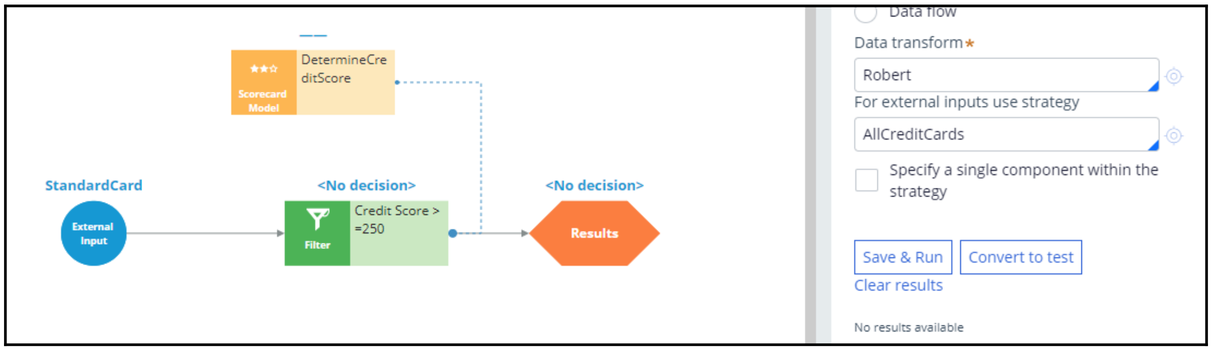
Specify a single component within the strategy

No results available



Field	Value
Segment	Not Suitable
ApplyAnalytics	——
Benefits	——
Bundle Parent	——
BundleType	——
Component	DetermineCredit Score

Now, repeat the test to verify results for the Robert data transform.



Data flow

Data transform*

Robert

For external inputs use strategy

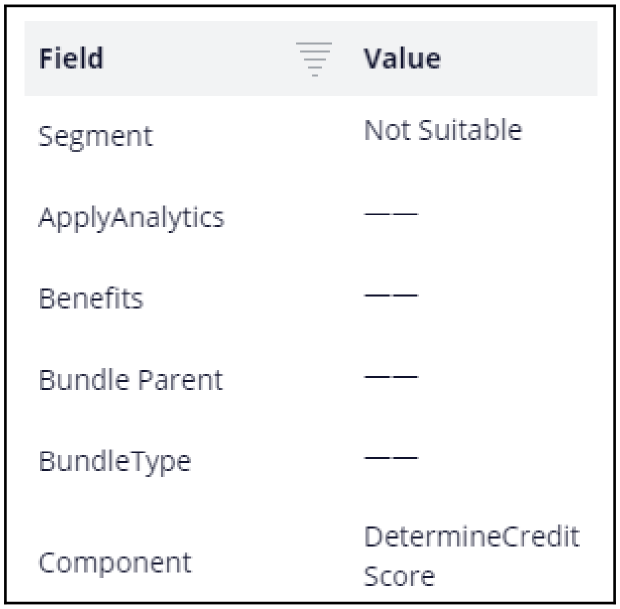
AllCreditCards

Specify a single component within the strategy

Save & Run | Convert to test

Clear results

No results available

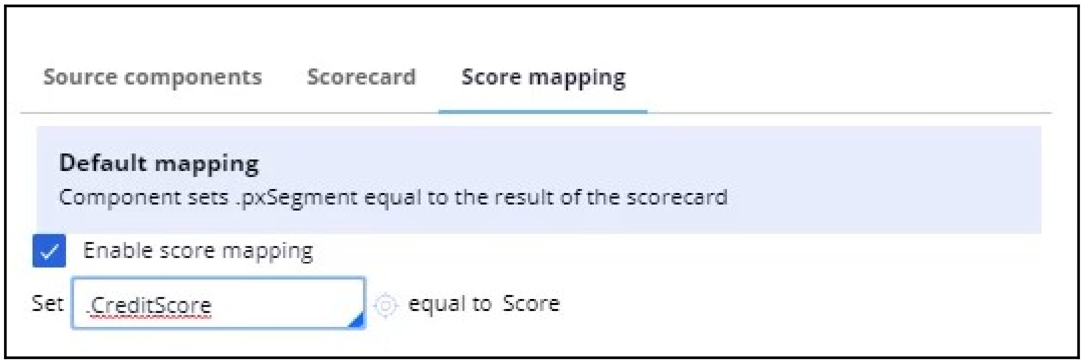


Field	Value
Segment	Not Suitable
ApplyAnalytics	——
Benefits	——
Bundle Parent	——
BundleType	——
Component	DetermineCredit Score

The strategy is also not outputting any results for Robert, as the Segment value is **Not Suitable**.

Now, assume the credit score value for customers is already computed and presented in the Customer data model. However, this value is not set for certain customers, in which case you want to use the credit score determined by the Scorecard itself.

To make these adjustments, start by opening the Scorecard component and mapping the score computed by the Scorecard to the CreditScore property.



Default mapping
Component sets .pxSegment equal to the result of the scorecard

✓ Enable score mapping

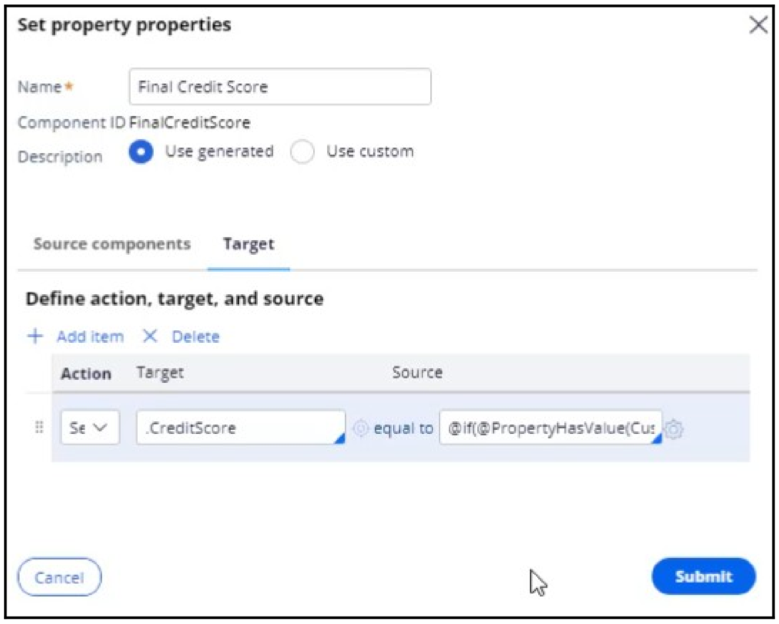
Set .CreditScore equal to Score

Then, add a Set Property component to determine the actual value of the credit score, given that the credit score is already available for certain customers.

Use 'Add item' to set the CreditScore to either the available credit score value from the Customer data model, or to the value computed by the Scorecard.

Define the Expression as *if(@PropertyHasValue(Customer.CreditScore), Customer.CreditScore, .CreditScore).*

If set, this Expression will result in the credit score from the Customer data model. If not, the credit score value will be computed by the Scorecard.


Set property properties

Name*	Final Credit Score
Component ID	FinalCreditScore
Description	[✓ Use generated] Use custom

Source components | Target

Define action, target, and source

+ Add item ✕ Delete

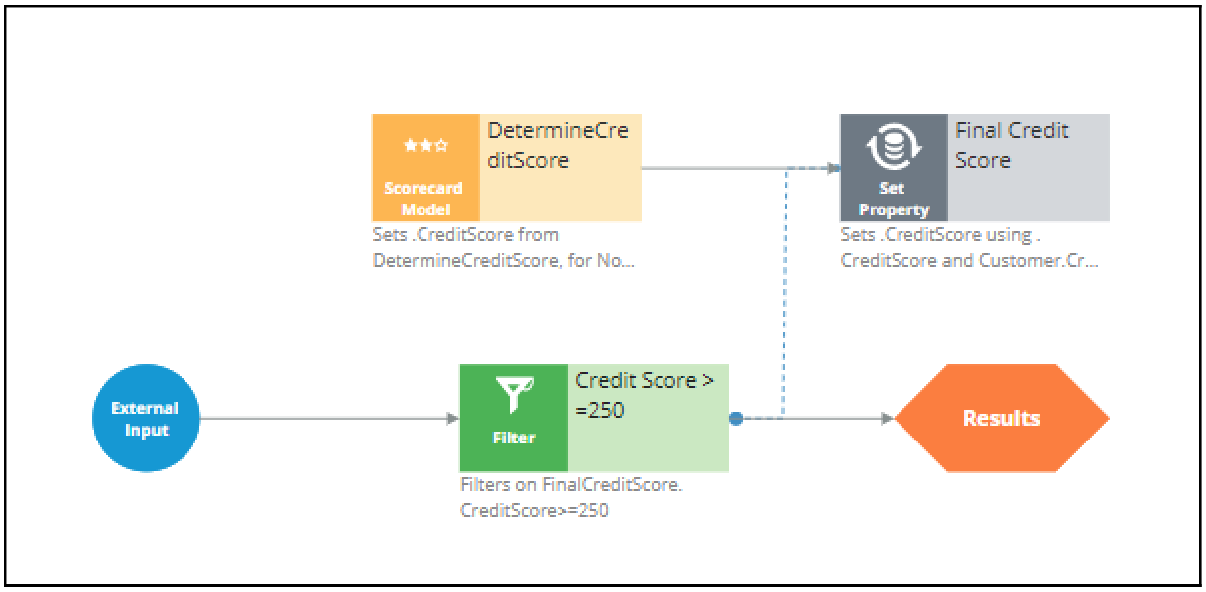
Action	Target		Source
Se	.CreditScore	equal to	@if(@PropertyHasValue(Cus

Once the Set Property component is configured, open the **Filter** component properties to modify the Filter condition. In this scenario, the bank has decided to present various credit cards to suitable customers who have credit scores greater than or equal to 250.

So, open the 'Expression builder' to modify the condition as *FinalCreditScore.CreditScore>=250*.

Now, connect the Scorecard Model component to the Set Property component to ensure the CreditScore value determined by the Scorecard is copied to the Set Property component.

Save the configurations.


Scorecard Model
DetermineCreditScore
Sets .CreditScore from DetermineCreditScore, for No...
Set Property
Final Credit Score
Sets .CreditScore using .CreditScore and Customer.Cr...
External Input
Filter
Credit Score >=250
Filters on FinalCreditScore.CreditScore>=250
Results

Re-test the strategy for the Troy and Robert data transforms.

Note that the strategy is not outputting any results for Troy, as his credit score, 200, is less than 250.

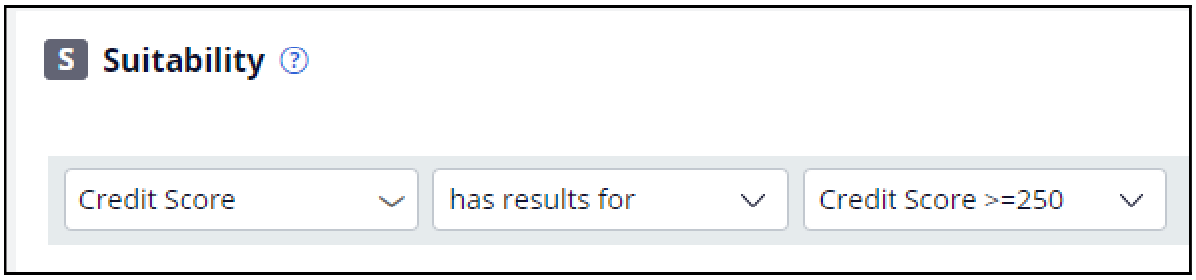
For Robert, the strategy is outputting results.

If you open the Robert data transform, note that the CreditScore in the data model is set to 600.

The Set Property component picks up the credit score value available in the data model (that is, CreditScore = 600) and not the value computed by the Scorecard (CreditScore = 200). That is why the strategy is outputting results for Robert.

Since you have completed configuring the strategy, check it in, so that the strategy will be available to the U+ bank website.

You can now configure this strategy in the Next-Best-Action Designer Engagement Policy as a suitability condition for the Group-level Suitability rules.


Suitability

Credit Score | has results for | Credit Score >=250

Select the strategy. The condition is: *Credit Score has results for the Credit Score >=250.*

Saving this completes all the required configurations.

On the U+ bank website, if you log in as Troy, notice that no credit card offer is displayed. This is because no credit cards are suitable for Troy.

Now, if you log in as Robert, notice that the credit card offer is displayed.

This is because credit cards are suitable for Robert.

This demo has concluded. What did it show you?

- How to use the segmentation result and the score value of a Scorecard in a decision strategy.

- How to use the *PropertyHasValue* function to check if a Customer property has value or not.

Creating eligibility rules using customer risk segments

Description

Decision tables enable you to better adjust to the variables in your business processes. Learn to create decision tables that help you derive a value that has one of a few possible outcomes, where each outcome can be detected by a test condition. Decision tables list two or more rows, each containing test conditions, optional actions, and a result.

Learning Objectives

- Create a decision table
- Use the outcome of a decision table in a decision strategy

Creating customer risk segments using a decision table

Business scenario

U+ Bank is currently doing cross-sell on the web by showing various credit cards to its customers. The bank already uses a customer's credit score to determine their suitability for a credit card.

The new eligibility rules require customers to be divided into risk segments varying from AAA to CCC. To start, customers in the risk category BB- and CCC are not eligible for credit cards. Customers from all other risk segments are eligible.

The risk segments are determined by two parameters: **Outstanding loan amount** and the customer **Credit score**.

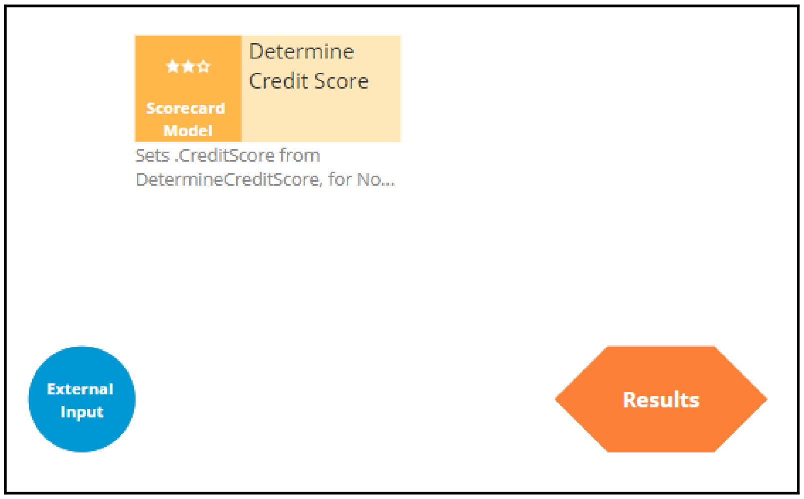
Condition	Risk Segment
If **Outstanding loan amount** < $10000 AND **Credit score** is > 600	AAA
If **Outstanding loan amount** between $10000 and $25000 AND **Credit score** is > 600	AAA-
If **Outstanding loan amount** between $25000 and $50000 AND **Credit score** is > 600	AA
If **Outstanding loan amount** > $50000 AND **Credit score** is > 600	AA-
If **Outstanding loan amount** < $25000 AND **Credit score** is between 400 and 600	BBB
If **Outstanding loan amount** between $25000 and $50000 AND **Credit score** is between 400 and 600	BBB-
If **Outstanding loan amount** > $50000 AND **Credit score** is between 400 and 600	BB-
If **Credit score** is between 200 and 400	CCC
If **Outstanding loan amount** AND **Credit score** falls in any other range	CCC

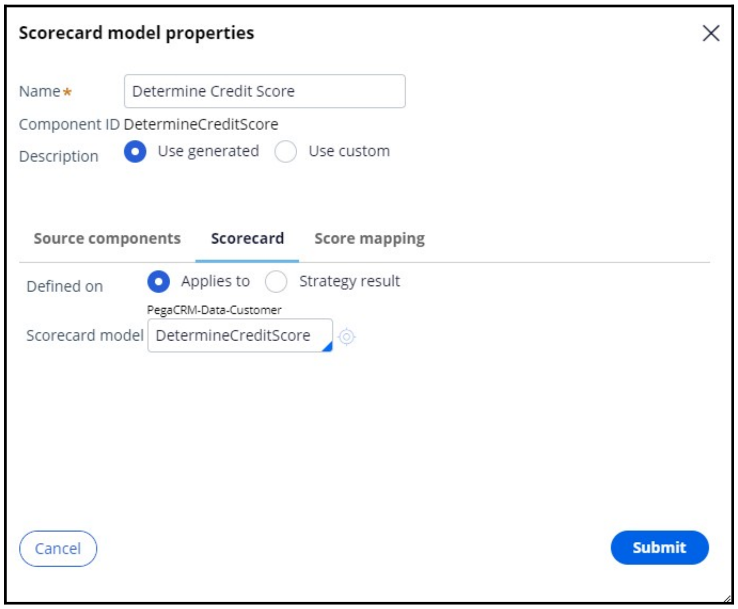
To implement the new bank regulations, use an Engagement Strategy to:

1. Calculate credit score.
2. Define risk segment.
3. Filter credit cards based on risk segment.

Calculate credit score

U+ has already created a Scorecard, **DetermineCreditScore**, which computes the customer's credit score. You can use the Scorecard in the decision strategy by adding a Scorecard component to the canvas.





Name* Determine Credit Score

Component ID DetermineCreditScore

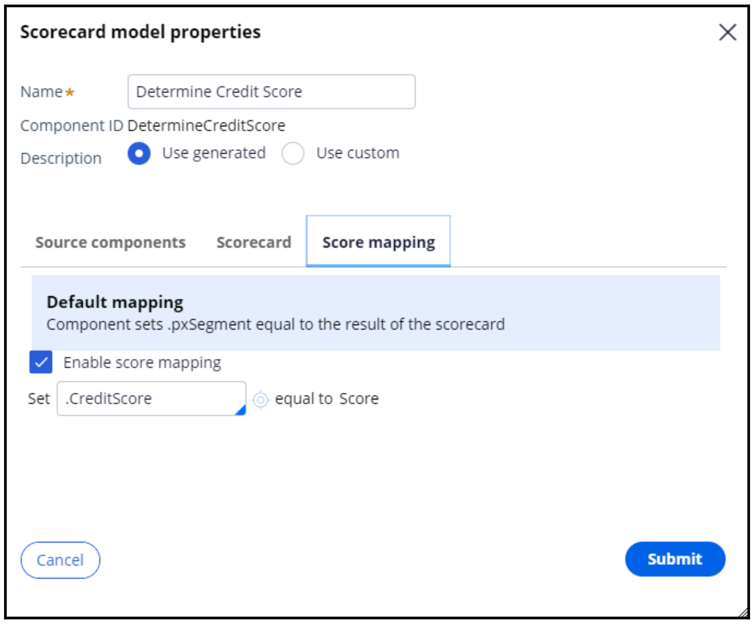
Description [✓ Use generated] Use custom

Defined on [✓ Applies to] Strategy result

PegaCRM-Data-Customer

Scorecard model DetermineCreditScore

Since you need the raw score, enable **Score mapping** and set the Score value in the Credit Score property.


Scorecard model properties

Name* Determine Credit Score

Component ID DetermineCreditScore

Description [✓ Use generated] Use custom

Source components | Scorecard | Score mapping

Default mapping
Component sets .pxSegment equal to the result of the scorecard

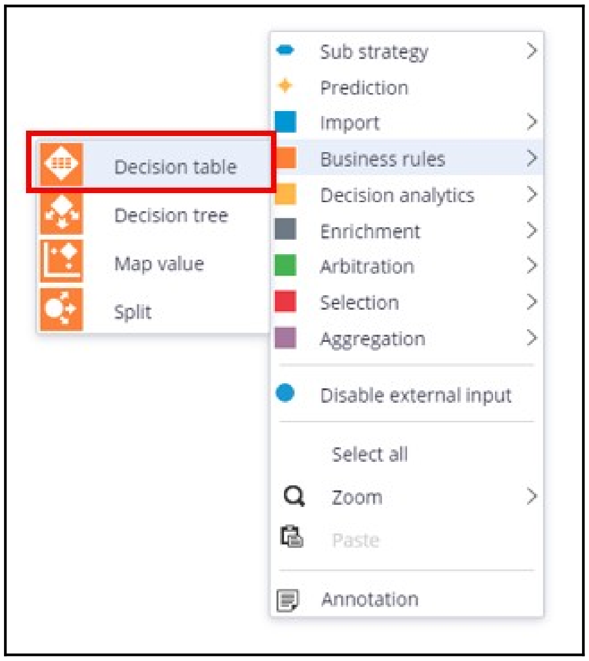
[✓] Enable score mapping

Set .CreditScore equal to Score

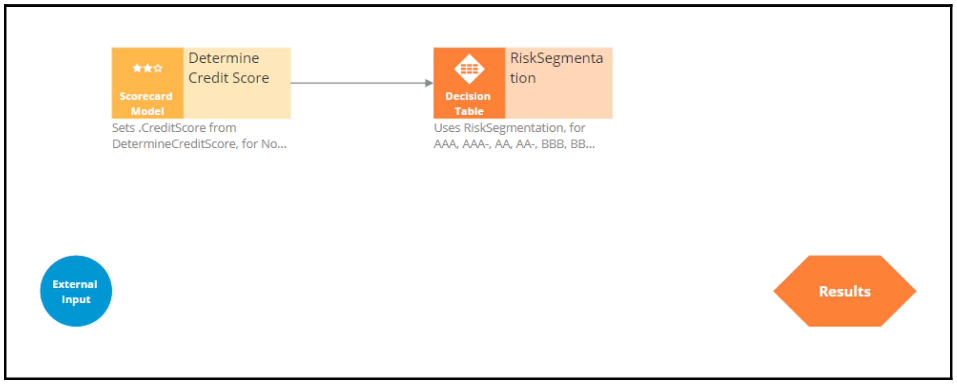
Cancel | Submit

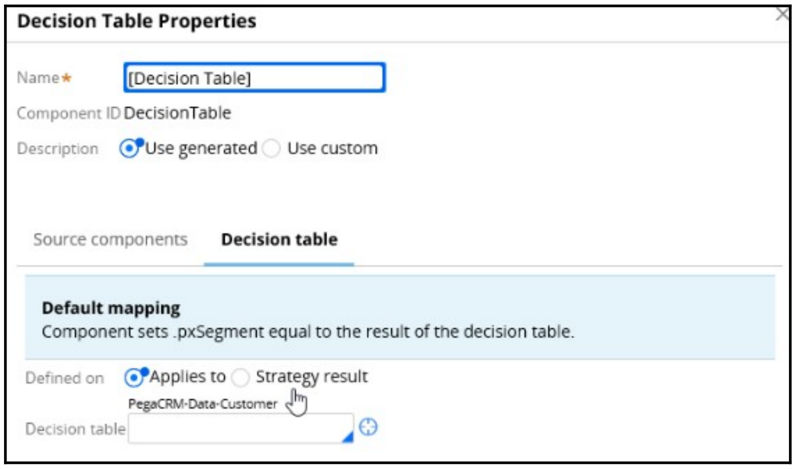
Define risk segment

To implement the risk segmentation requirements, use a **Decision table** component from the **Business rules** category, which outputs the customer risk segment.



The Decision Table can be defined on the Strategy Result, SR class, or a Customer class. The choice often depends on where the properties used in the Decision Table are located. In this case the outstanding loan amount is a Customer property and the Credit Score calculation is an SR property, so both options are valid.


Scorecard Model
Determine Credit Score
Sets .CreditScore from DetermineCreditScore, for No...
Decision Table
RiskSegmentation
Uses RiskSegmentation, for AAA, AAA-, AA, AA-, BBB, BB...
External Input
Results



Decision Table Properties

Name* [Decision Table]

Component ID DecisionTable

Description [✓ Use generated] Use custom

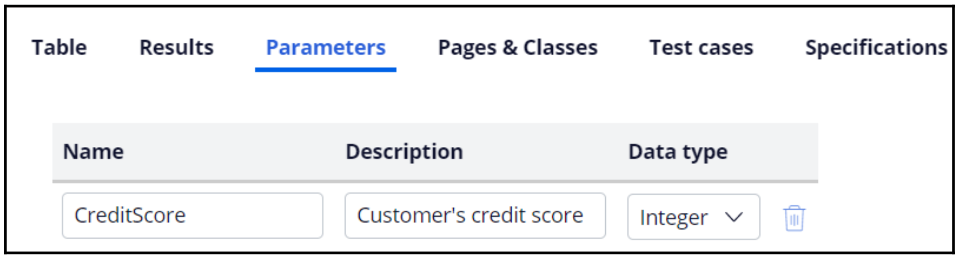
Source components | **Decision table**

Default mapping
Component sets .pxSegment equal to the result of the decision table.

Defined on [✓ Applies to] Strategy result
PegaCRM-Data-Customer

Decision table

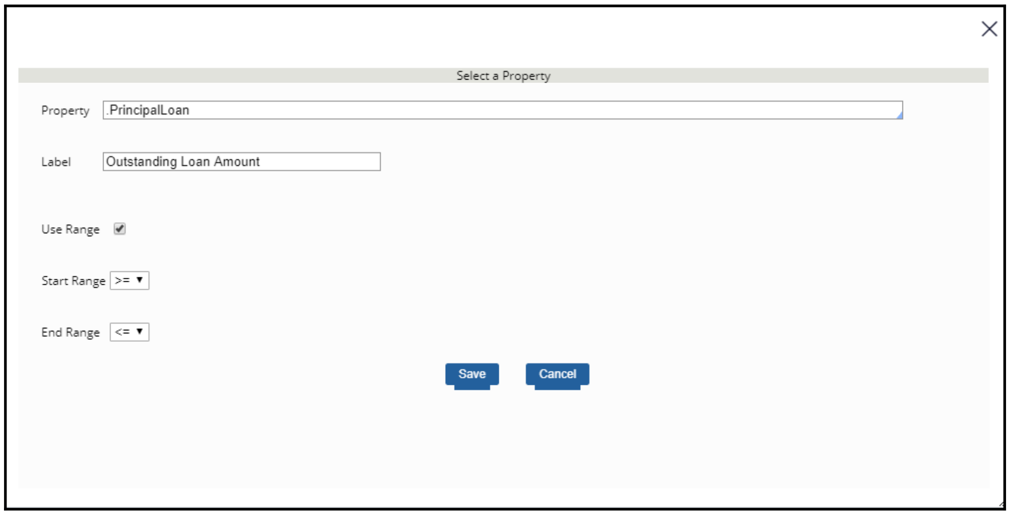
To use Credit Score as a parameter, you first need to define it on the **Parameters** tab. For this scenario, define the Customer class and reference it from the DT component.



Name	Description	Data type
CreditScore	Customer's credit score	Integer

Creating the table requires two condition columns. First is the **Principal Loan**, which is the property in the data model representing the outstanding loan amount.

The Operator represents the condition applied to the column. In this case, **greater than**, which allows you to express the **Outstanding loan amount** condition from the requirement.

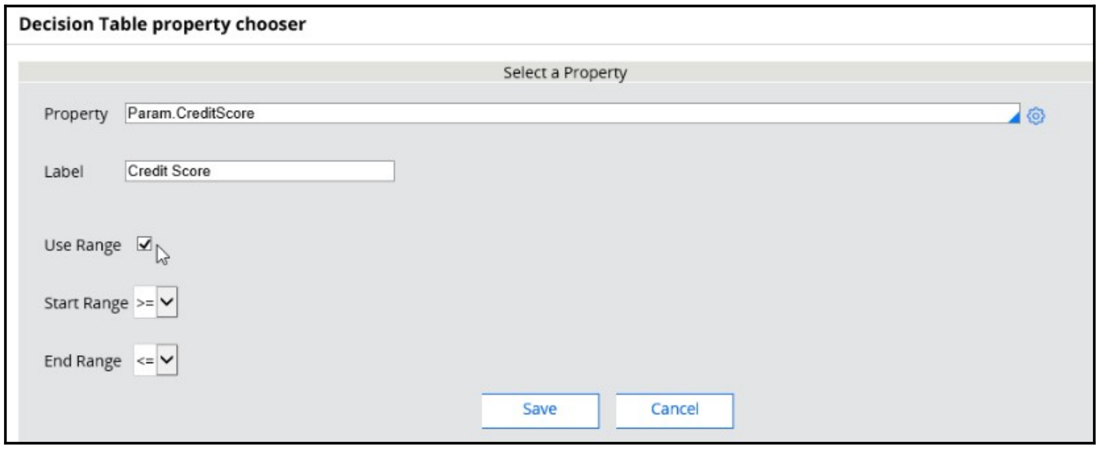


Select a Property

Property	.PrincipalLoan
Label	Outstanding Loan Amount
Use Range	✓
Start Range	>=
End Range	<=

The second condition column is the Credit Score property. The Credit Score is a parameter, so you need to use the **Param.PropertyName** construct.

The **Use Range** checkbox groups the credit scores together.

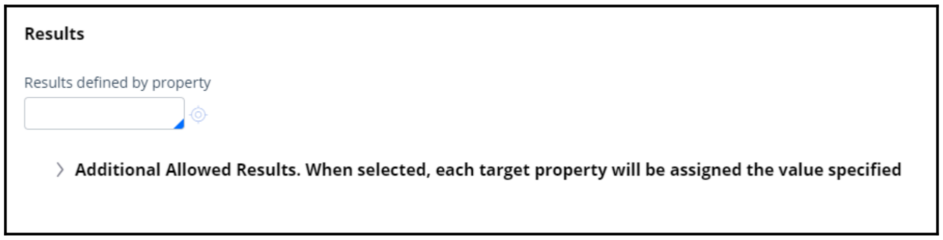


Decision Table property chooser

Select a Property

Property	Param.CreditScore
Label	Credit Score
Use Range	✓
Start Range	>=
End Range	<=

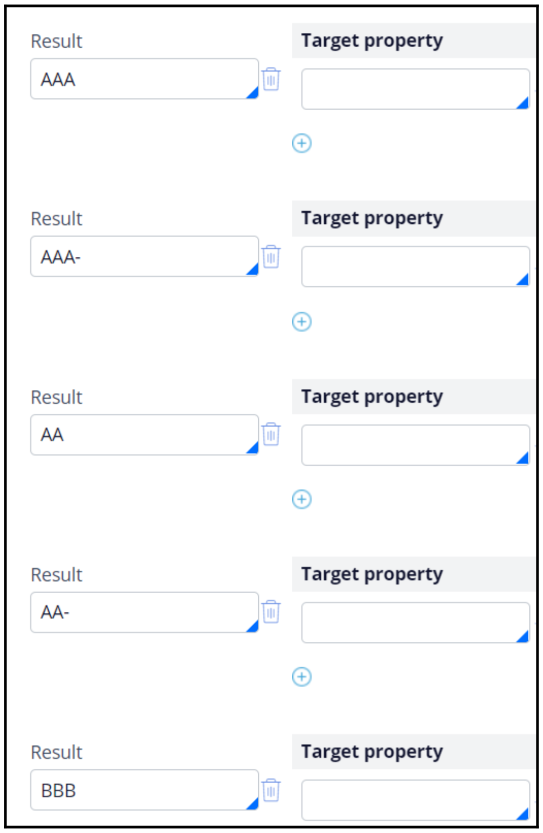
Next, you need to specify the possible outputs of the Decision Table in the **Results** tab.



Results

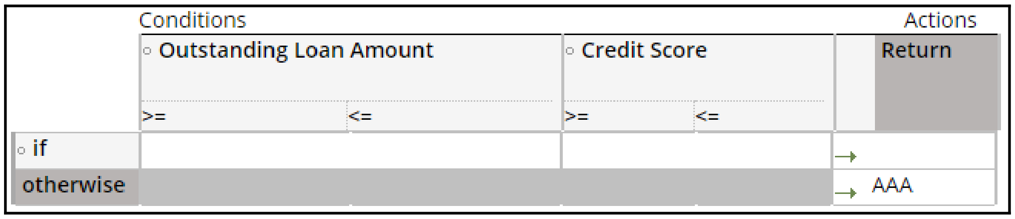
Results defined by property

Additional Allowed Results. When selected, each target property will be assigned the value specified



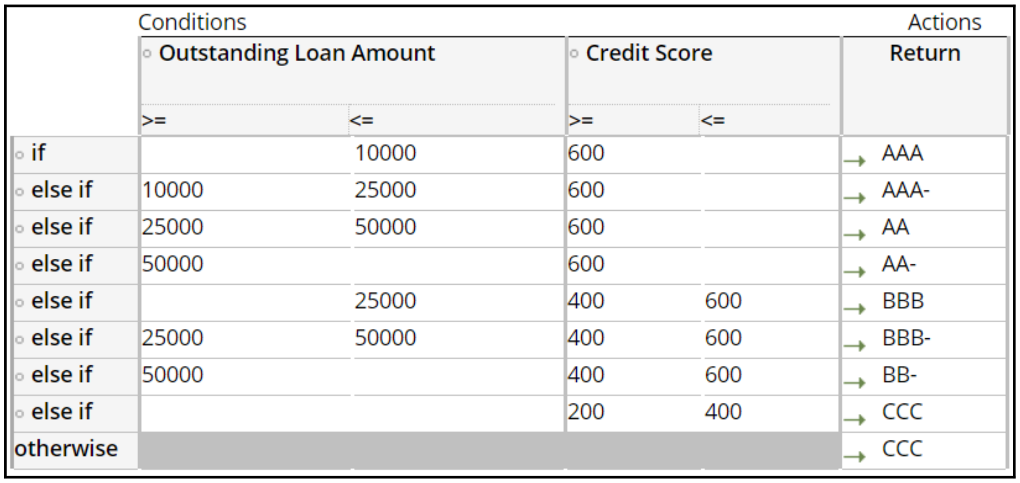
Result	Target property
AAA	
AAA-	
AA	
AA-	
BBB	

After adding the **Target properties** and **Results**, the Table will look like the following.



	Conditions				Actions
	Outstanding Loan Amount		Credit Score		Return
	>=	<=	>=	<=	
if					
otherwise					AAA

Fill in the bank's requirements and specify a **Return** value for each row in the table.



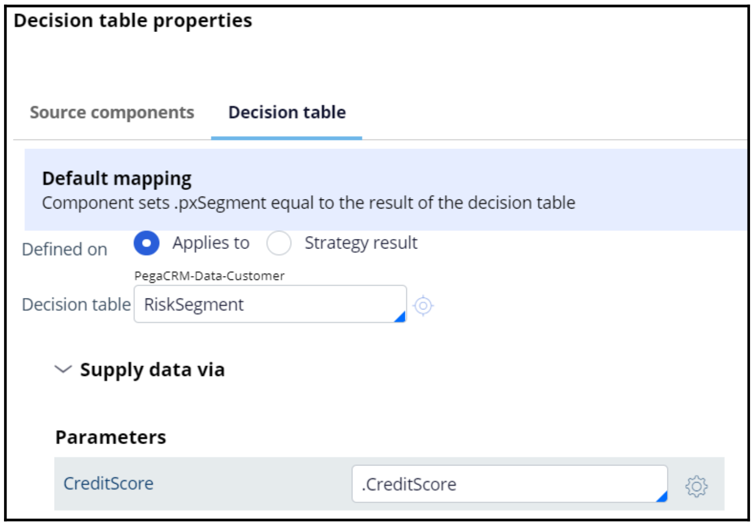
	Conditions				Actions
	Outstanding Loan Amount		Credit Score		Return
	>=	<=	>=	<=	
if		10000	600		AAA
else if	10000	25000	600		AAA-
else if	25000	50000	600		AA
else if	50000		600		AA-
else if		25000	400	600	BBB
else if	25000	50000	400	600	BBB-
else if	50000		400	600	BB-
else if			200	400	CCC
otherwise					CCC

Note, if you leave a value blank it is ignored by the Decision Table. For example, leaving the Credit Score value blank, means that credit score comparison always returns a value of True.

If none of the conditions are met, for example, if the loan amount is zero and the credit score is 50, the Otherwise path is taken; in this example the result will be CCC.

It is important to note that once a Decision Table condition is satisfied, the processing stops. For example, if the loan amount is 30,000, and the credit score is 650, the processing stops at row three returning the result "AA". The other rows are not processed.

After the Table is configured, go back to the property panel of the Decision Table component and map the Decision Table parameter to a Strategy property.



Decision table properties

Source components | **Decision table**

Default mapping
Component sets .pxSegment equal to the result of the decision table

Defined on: [✓ Applies to] Strategy result

PegaCRM-Data-Customer

Decision table: RiskSegment

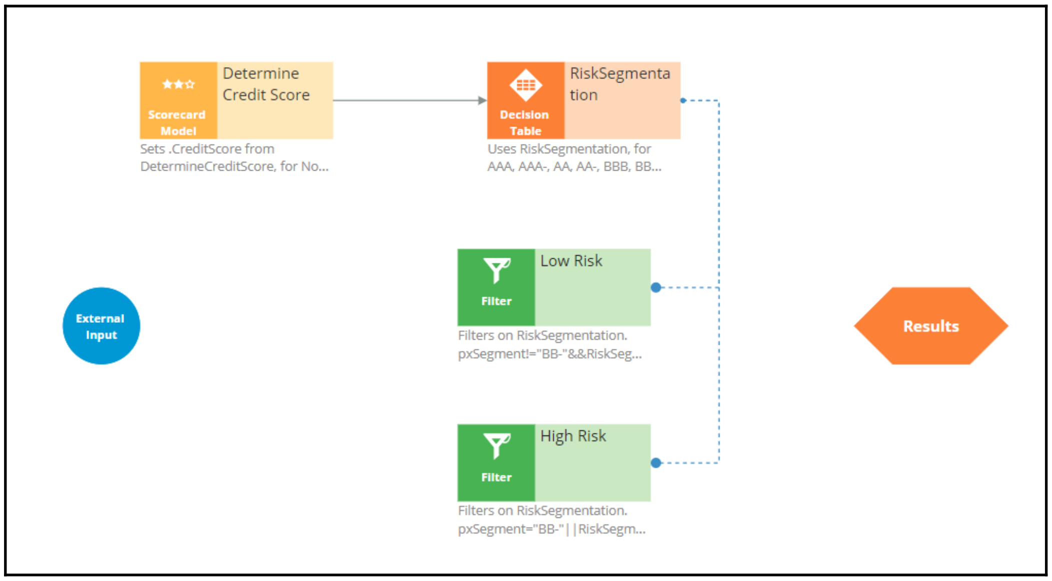
Supply data via

Parameters

CreditScore	.CreditScore

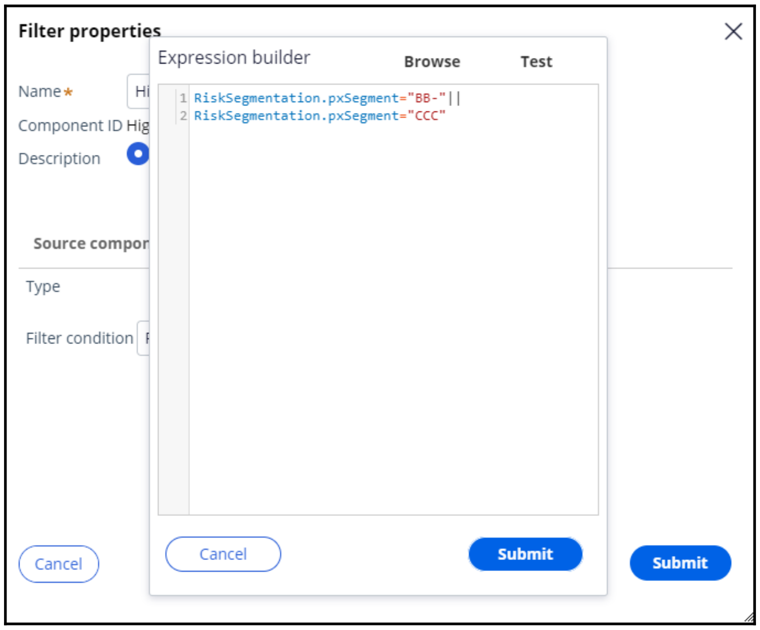
Filter credit cards based on the calculated risk segment

Configure two Filter components to ensure that you identify High Risk and Low Risk customers.


Scorecard Model
Determine Credit Score
Sets .CreditScore from DetermineCreditScore, for No...
Decision Table
RiskSegmentation
Uses RiskSegmentation, for AAA, AAA-, AA, AA-, BBB, BB...
Filter
Low Risk
Filters on RiskSegmentation. pxSegment!="BB-"&&RiskSeg...
External Input
Results
Filter
High Risk
Filters on RiskSegmentation. pxSegment="BB-"||RiskSegm...

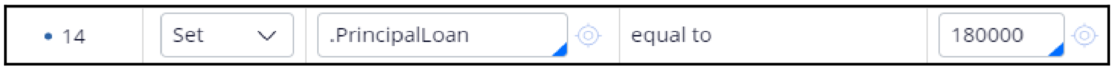
Customers in the risk category BB- and CCC are not eligible for credit cards, but all other customers are eligible. The **pxSegment** Strategy property contains the output of the Decision Table. So, create relevant expressions.

Low Risk Filter: *RiskSegmentation.pxSegment!="BB-"&&*
RiskSegmentation.pxSegment!="CCC"
High Risk Filter: *RiskSegmentation.pxSegment="BB-"||*
RiskSegmentation.pxSegment="CCC"



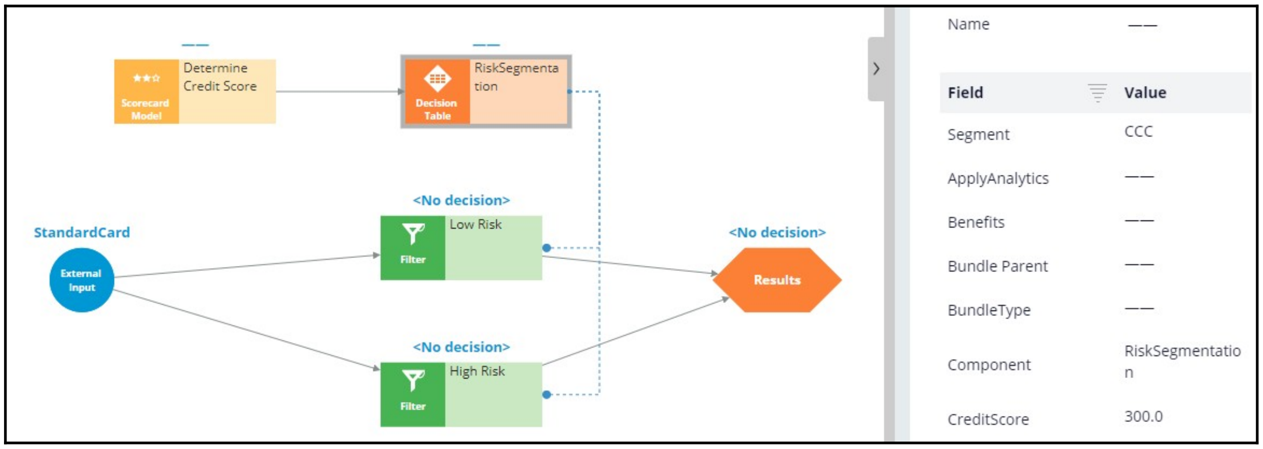
```
RiskSegmentation.pxSegment="BB-"||
RiskSegmentation.pxSegment="CCC"
```

When you test the strategy using the Robert data transform, you will see that Robert falls under the category of high-risk customers, as he has a huge outstanding loan amount, over 50,000.



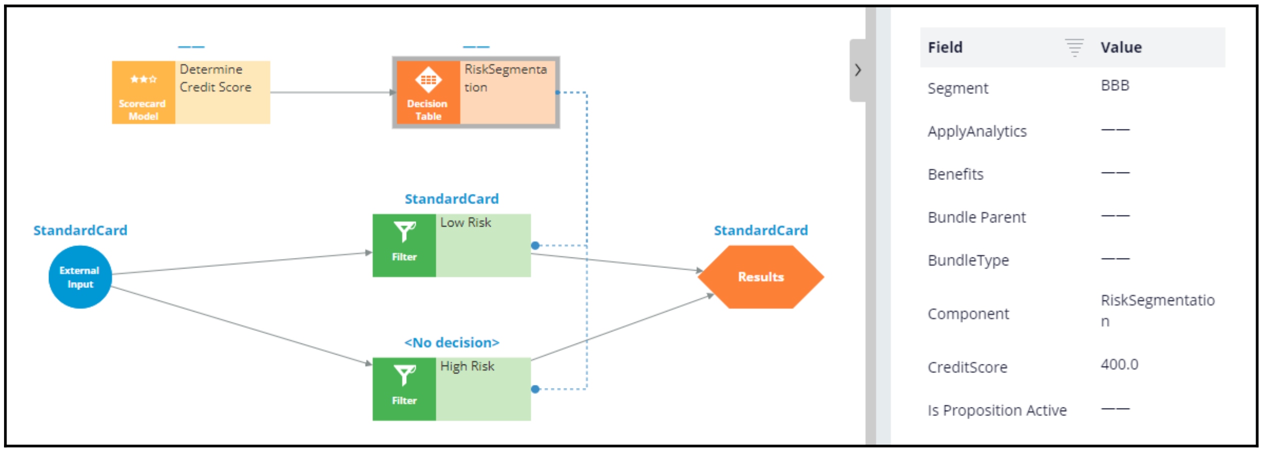
• 14	Set	.PrincipalLoan	equal to	180000

Therefore, you can expect his risk segment to be CCC.



Field	Value
Name	——
Segment	CCC
ApplyAnalytics	——
Benefits	——
Bundle Parent	——
BundleType	——
Component	RiskSegmentation
CreditScore	300.0

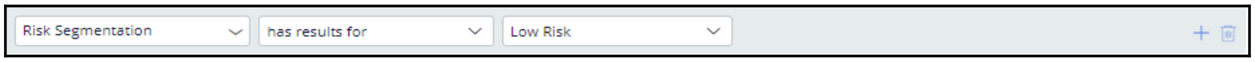
When you test the strategy for Arnold, who has an outstanding loan of 8000 and a credit score of 400, the Decision Table correctly classifies Arnold in the BBB risk segment and allows all credit cards for him.



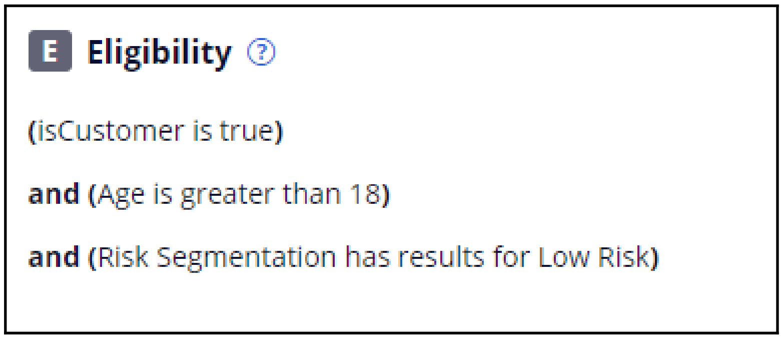
Field	Value
Segment	BBB
ApplyAnalytics	——
Benefits	——
Bundle Parent	——
BundleType	——
Component	RiskSegmentation
CreditScore	400.0
Is Proposition Active	——

Define the Eligibility criteria

Once the decision strategy is ready, you can complete the Eligibility criteria definition in Next-Best-Action Designer.



Risk Segmentation | has results for | Low Risk



E Eligibility

(isCustomer is true)

and (Age is greater than 18)

and (Risk Segmentation has results for Low Risk)

Leveraging predictive models

Description

Predictive analytics requires historical data that contains the customer behavior you want to predict. A predictive model is used to predict an outcome based on customer behavior such as offer acceptance and churn based on customer characteristics such as credit risk, income, product subscriptions, etc. A predictive model component references a predictive model, which in turn predicts customer behavior.

Learning objectives

Describe predictive analytics

Describe how to use a predictive model in a decision strategy

Arbitrate between business issues using applicability rules in Next-Best-Action Designer

Predictive models drive predictions

Introduction

Learn how to better address your customers' needs by predicting customer behavior and business events with predictions. For example, you can determine the likelihood of customer churn or chances of successful case completion.

Predictions combine predictive analytics and best practices in data science.

Transcript

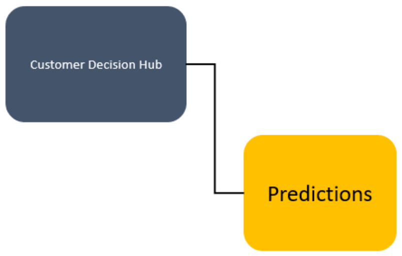
This video shows you how predictions can predict events in a business activity.

You can create and use predictions for several use cases.

Pega Customer Decision Hub™ predictions are used in decision strategies to optimize engagement with customers.

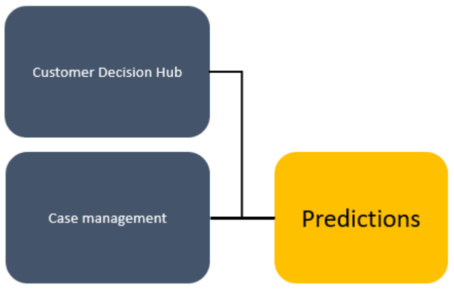
The most common predictions are provided out of the box. You can use the default predictions or create your own.

For example, you can use predictions to predict whether customers are likely to accept your offer or cancel a subscription.

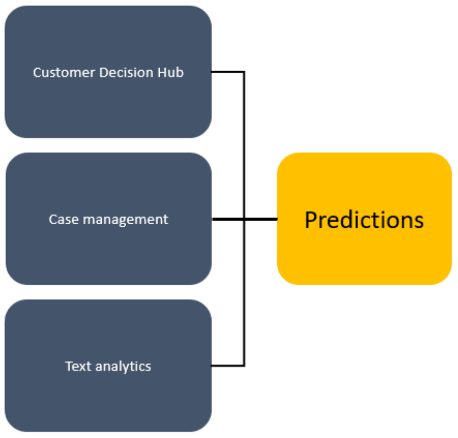


Case management predictions are used in case automation to route cases or prioritize work.

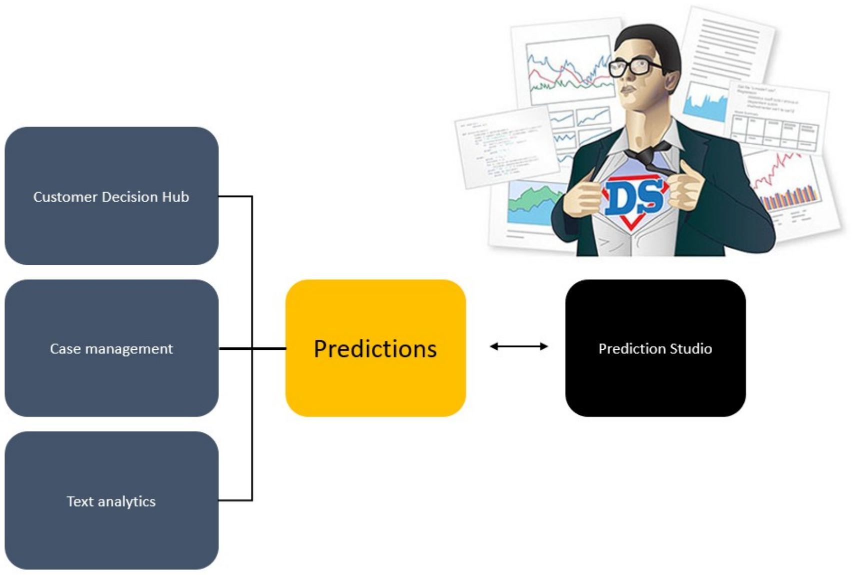
For example, you can create a prediction that predicts whether a case is likely to be completed successfully, or the probability that a claim is fraudulent, which might be useful in straight-through processing of claims with a low score.



Text analytics predictions analyze the text that comes through your channels, such as email or chat, to predict the topic or sentiment of the text.



Predictions are created and managed in Prediction Studio, the dedicated workspace for data scientists.



A next-best-action designer can include the predictions in decision strategies in Customer Decision Hub, to better adjust to customer needs and achieve business goals at the same time.

This handover of predictions from the data scientist to the next-best-action designer contributes to a separation of concerns.

Predictions combine predictive analytics and best practices in data science.

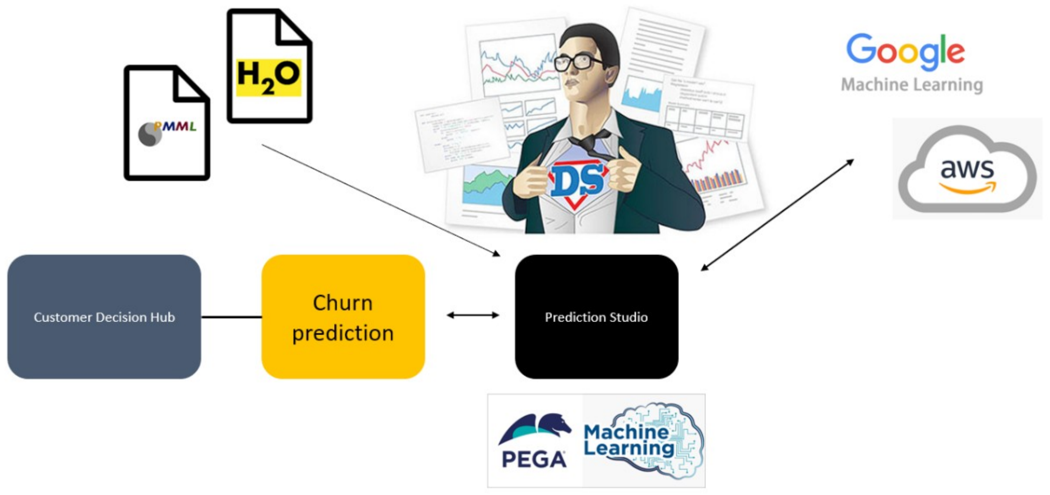
A predictive model drives a prediction. A data scientist can replace the predictive model at any time. However, the prediction always predicts the same outcome.

For example, you can create a new Customer Decision Hub prediction that calculates the likelihood that a customer might end a subscription soon.

To drive the prediction, a data scientist creates a predictive churn model on a platform of choice.

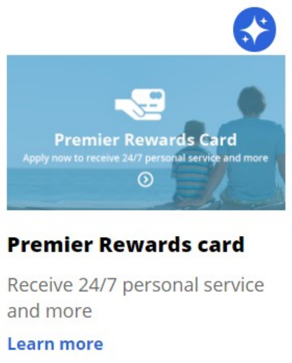
Prediction Studio supports the PMML and H2O model file formats. It can connect to machine learning services like Google ML and Amazon SageMaker.

You also have the option to use Pega machine learning.



One of the predictions that is shipped with Customer Decision Hub, **Predict Web Propensity**, calculates the likelihood that a customer clicks on a web banner.

Customer Decision Hub decides which banner to show based on the calculated propensity and weighting in business requirements and context.


Premier Rewards Card
Apply now to receive 24/7 personal service and more
Premier Rewards card
Receive 24/7 personal service and more
Learn more

In this case, the Premier Rewards card has a propensity of 0.8, and it is ranked as the next best action.

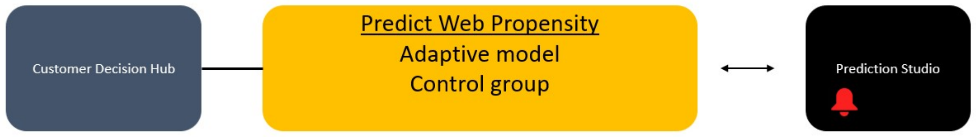
The **Predict Web Propensity** prediction is driven by an adaptive model that learns from each customer interaction.

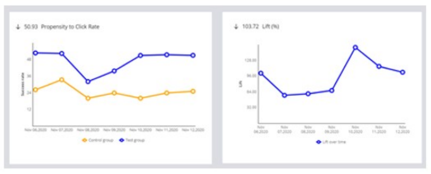
Best practices in data science include the use of a control group. Customers in the control group are shown a random banner instead of one based on the calculated propensity.

Comparing the success rate in the control group and the test group measures lift, which is the boost in success rate that the prediction generates. Lift is an important business metric.

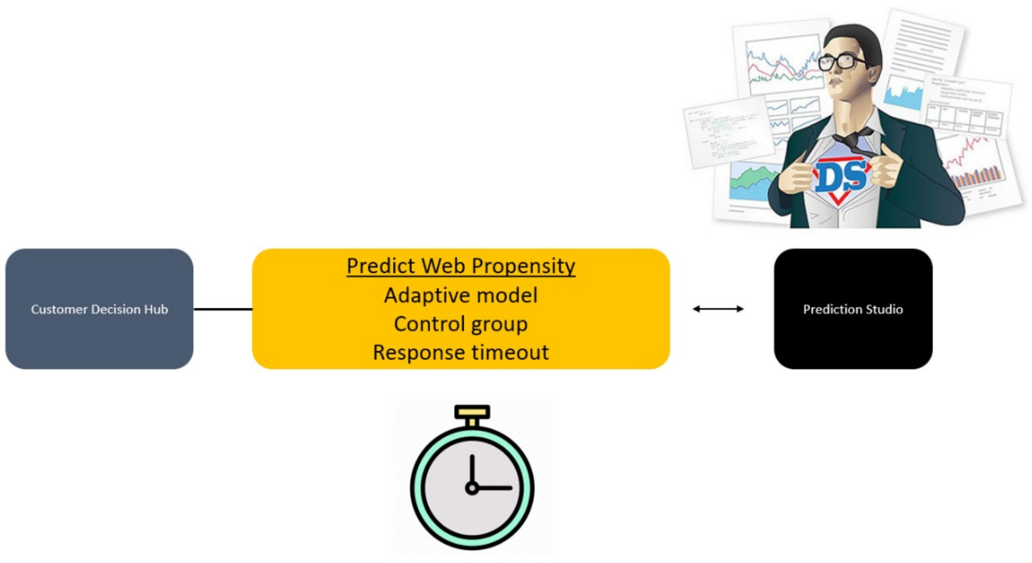
Also, the use of a control groups allows continuous exploration by the predictive models.

When the lift drops significantly over time, Prediction Studio notifies the data scientist.





The response timeout is built into predictions. When the timeout expires, NoResponse is automatically recorded as the outcome of the interaction.

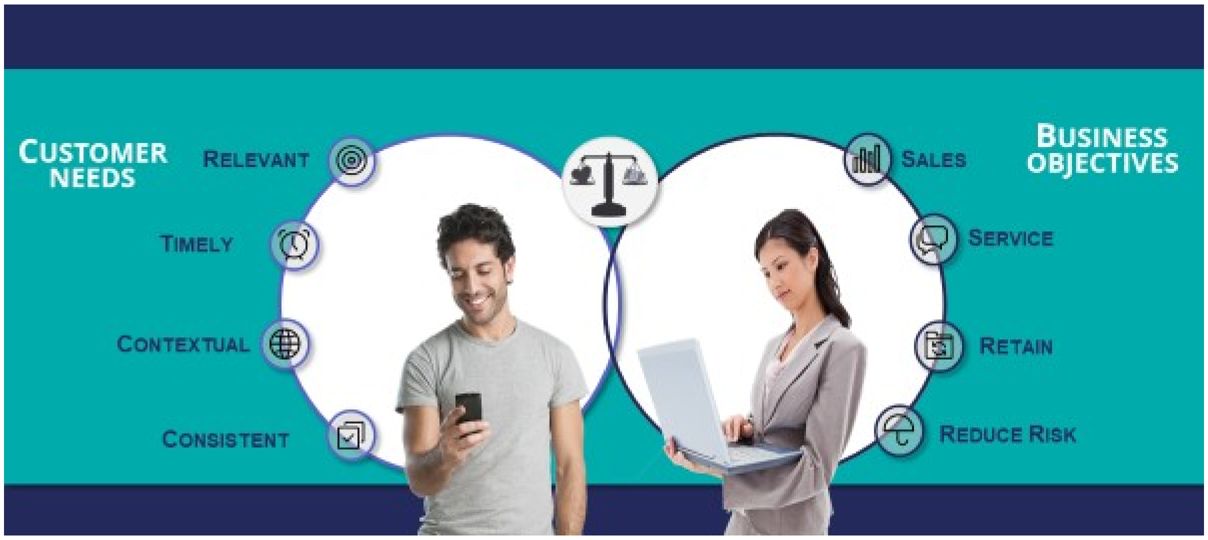


You have reached the end of this video. What did it show you?

- How predictions can predict events in the business activity
- How a prediction uses a control group to measure the boost in success rate that is generated by the prediction
- How NoResponse is automatically recorded as the outcome of the interaction when the response timeout expires

Arbitrating across business issues

Pega Customer Decision Hub™ combines analytics, business rules and customer data to make intelligent decisions. Every next best action weighs customer needs and business objectives to optimize decisions.

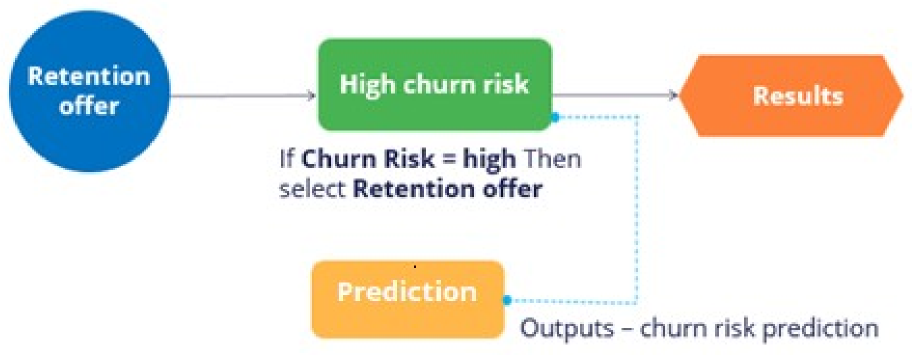


To arbitrate across multiple business issues, a you create a decision strategy and uses this strategy to define applicability conditions in Next Best Action Designer.

For example, consider a bank who is already doing sales. Now it wants to proactively offer retention offers for customers with high churn risk.

To predict the churn risk, a data scientist creates a churn prediction that calculates the likelihood that a customer will soon leave the bank.

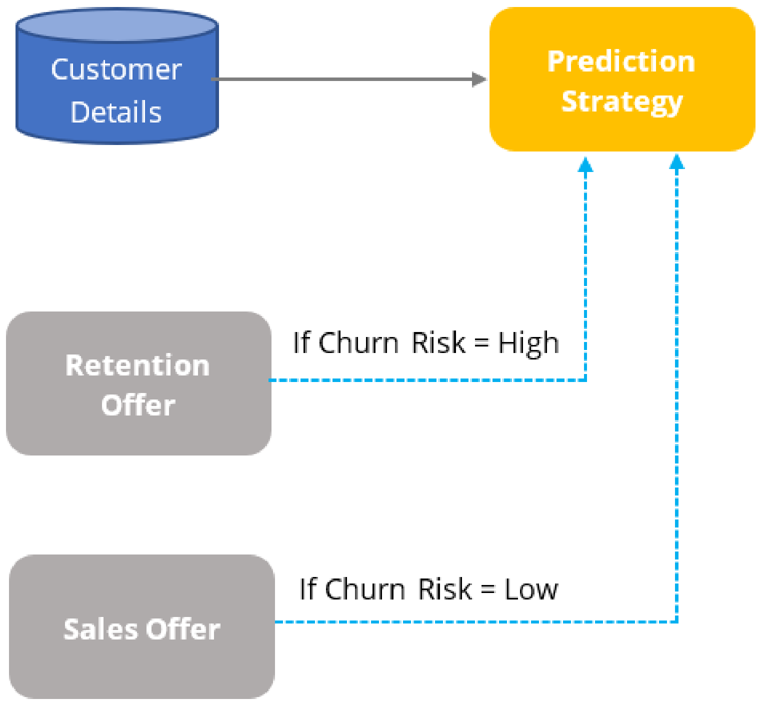
You create a decision strategy that differentiates between high risk and low risk customers that references the churn prediction.


Retention offer
High churn risk
Results
If Churn Risk = high Then select Retention offer
Prediction
Outputs – churn risk prediction

In Next Best Action Designer, you use the decision strategy to define applicability rules for the sales offers and the retention offers.

When the customer has a low churn risk, sales offers are applicable.

Conversely, when the customer has a high churn risk, retention offers are applicable.



Using predictions in engagement strategies

Introduction

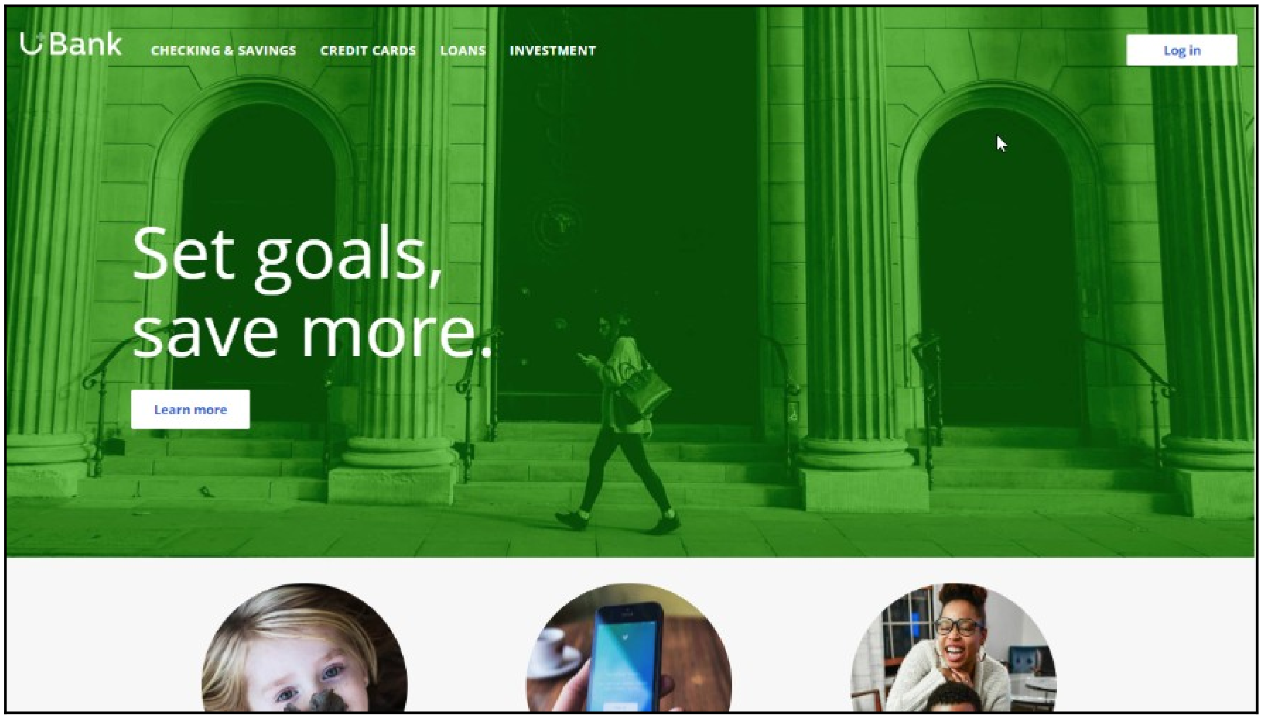
A prediction is used to predict customer behavior such as offer acceptance and churn based on characteristics such as credit risk, income, and product subscriptions. Learn how to arbitrate between different groups of actions to display more relevant offers to customers. Gain experience using a prediction in a decision strategy and learn how applicability rules can be defined to reflect the bank's requirements in a decision strategy.

Transcript

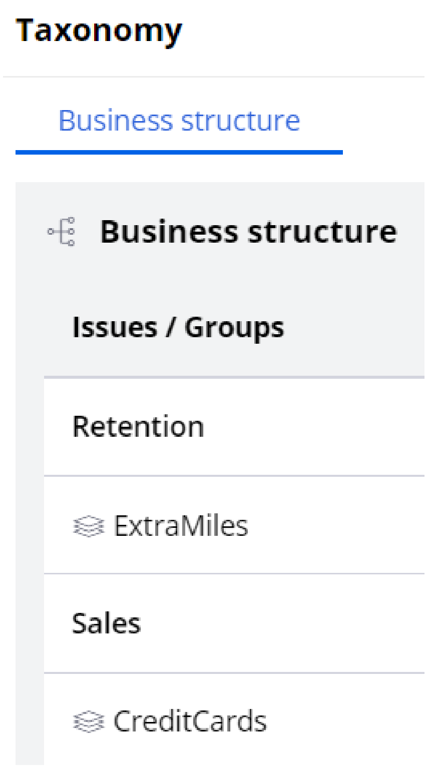
This demo shows you how to use a prediction in an engagement strategy to determine customer applicability for a retention offer.

Currently, U+ Bank is cross-selling on the web by showing various credit cards to eligible customers who log in to its website.

The bank now wants to show a retention offer, instead of a credit card offer, to customers who are likely to churn in the near future. The credit card offers are shown only to loyal customers.



To meet this business requirement, a decisioning administrator has already set up the taxonomy by defining a new business issue called Retention, and an offer group.


Taxonomy

Business structure

Business structure

Issues / Groups

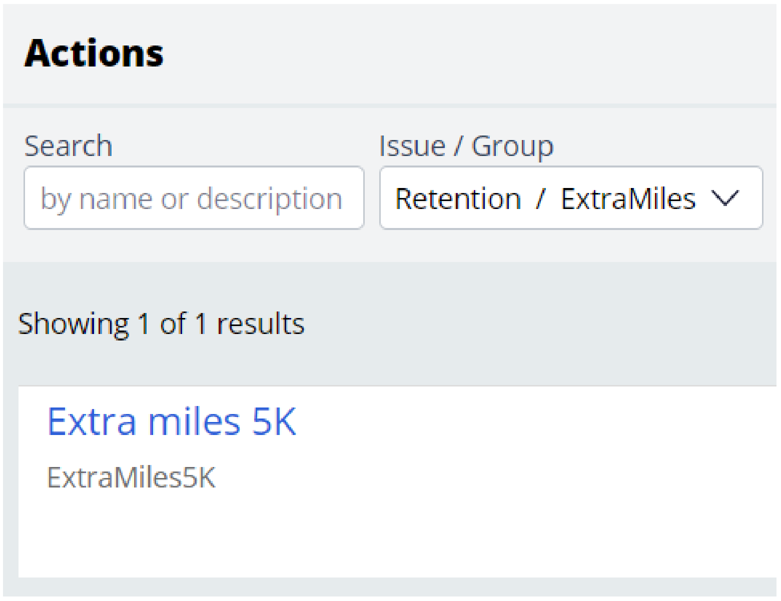
Retention

ExtraMiles

Sales

CreditCards

This ExtraMiles group contains a retention offer, Extra Miles 5K.


Actions

Search: by name or description

Issue / Group: Retention / ExtraMiles

Showing 1 of 1 results

Extra miles 5K

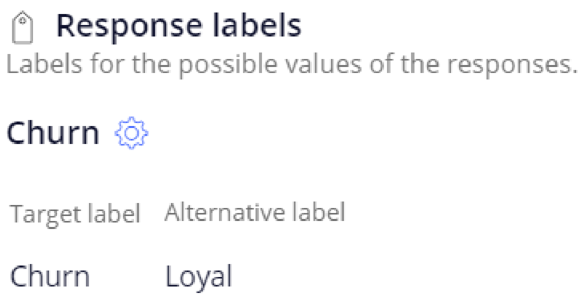
ExtraMiles5K

The next step is to create an applicability condition that makes a customer qualify for a retention offer when there is a high likelihood that the customer might churn.

A data scientist has created a prediction that identifies these high-risk customers.

When you open the prediction in Prediction Studio, notice that the possible response labels are **Churn** and **Loyal** to predict customer behavior.

The result of the prediction is stored in the pxSegment property.



Response labels

Labels for the possible values of the responses.

Churn

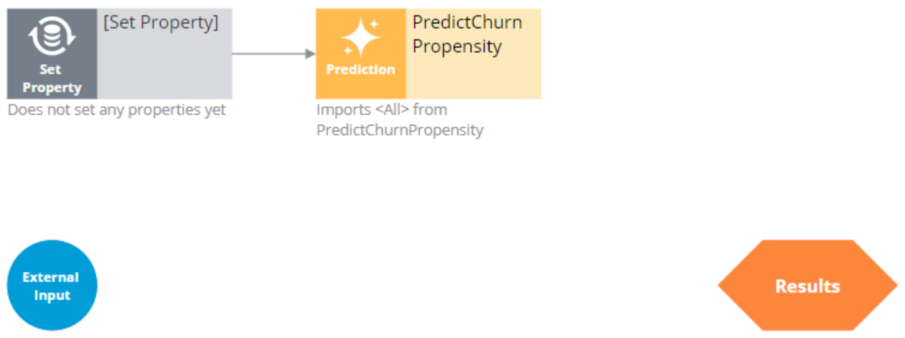
Target label	Alternative label
Churn	Loyal

To define the applicability condition, you create a decision strategy to output a retention offer only if the response label of the prediction is **Churn**.

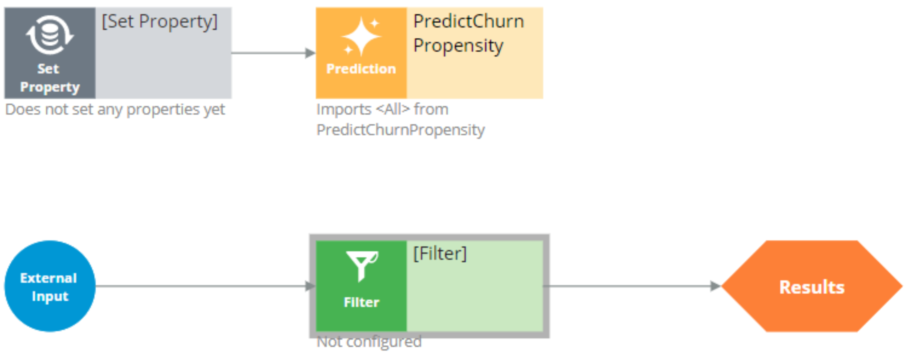
Add a **Prediction** component to the canvas and configure it to reference the churn prediction.

Add a **Set Property** component and connect it to the Prediction component.

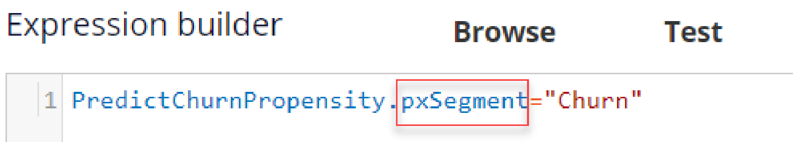
You can configure the Set Property component at a later point to accommodate parameterized fields.



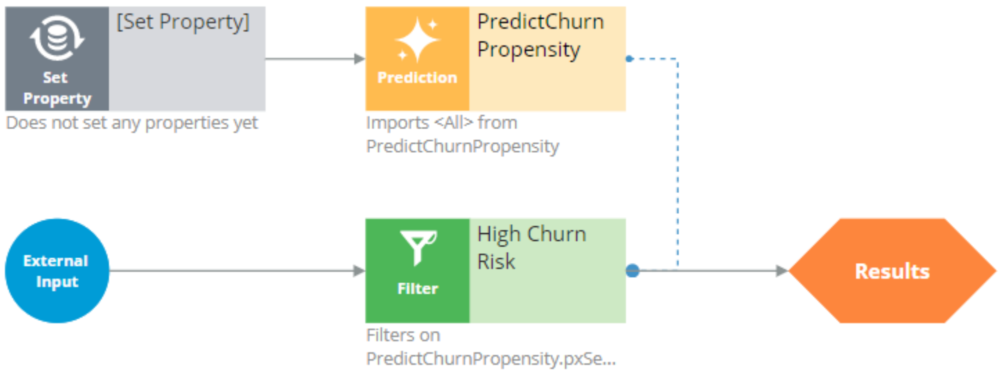
Next, add a filter component to filter out the loyal customers and pass retention offers to high churn risk customers only.



The filter condition is defined to output a retention offer when the pxSegment property of the prediction is equal to **Churn**.



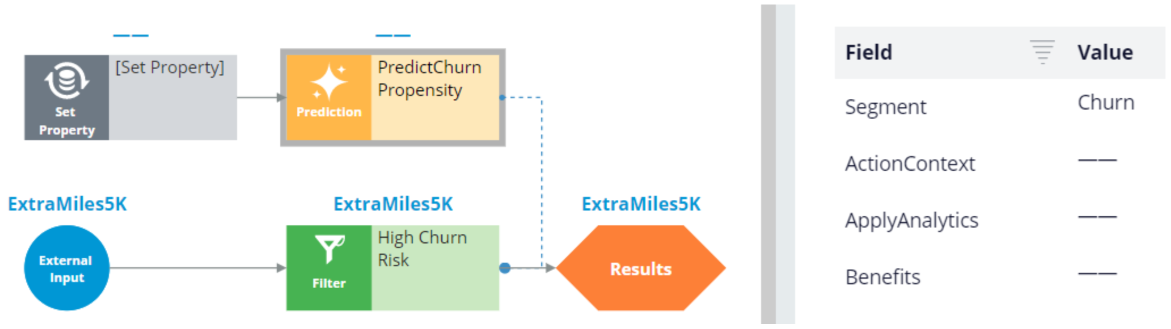
```
PredictChurnPropensity.pxSegment="Churn"
```


[Set Property]
Set Property
Does not set any properties yet
PredictChurn Propensity
Prediction
Imports <All> from PredictChurnPropensity
External Input
High Churn Risk
Filter
Filters on PredictChurnPropensity.pxSe...
Results

Next, test the strategy using two customer profiles, **Troy** and **Barbara**.

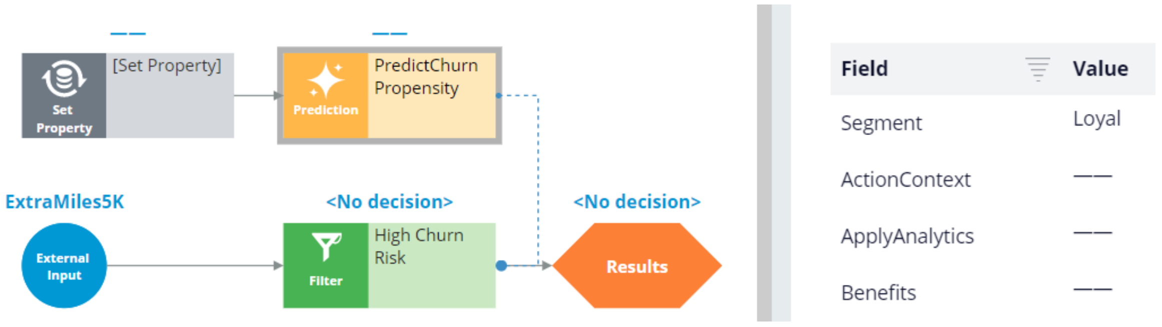
For external inputs, consider all available retention offers.

The strategy outputs a result for **Troy** because the result of the prediction is **Churn**.



Field	Value
Segment	Churn
ActionContext	—
ApplyAnalytics	—
Benefits	—

The strategy does not have a result for **Barbara**, because the Segment value is **Loyal**.



Field	Value
Segment	Loyal
ActionContext	——
ApplyAnalytics	——
Benefits	——

By checking in the strategy, you commit your changes so that they go into effect.

You can now use this strategy in the Next-Best-Action Designer engagement policy as an applicability condition.

The first business rule you need to implement is that the **ExtraMiles** group is applicable only to high churn risk customers.

To implement this rule, in the **Applicability** section, define a condition for the customer field.

Select the **RetentionStrategy**. The condition is: the RetentionStrategy has results for the High Churn Risk component.

The second business rule you need to implement is: U+ Bank wants to show credit card offers to low-risk customers only; meaning the **CreditCards** group is not applicable for high-risk customers.

To implement this rule, modify the **Applicability** section of the **CreditCards** group.

The condition is: the RetentionStrategy doesn't have results for the High Churn Risk component.

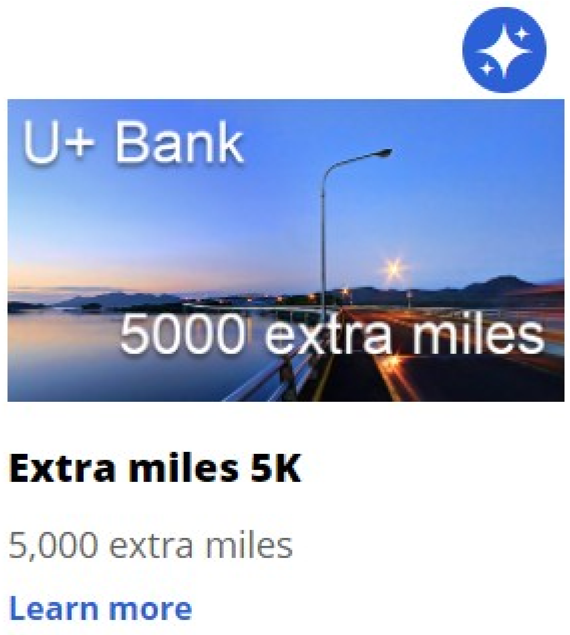
Once the applicability conditions are defined, you need to amend the **Channels** configuration.

Because U+ Bank introduced a new group, **ExtraMiles**, which belongs to a new business issue, **Retention**, you need to select the results from the appropriate business structure level.

In this case, the bank wants to arbitrate between two different business issues: Sales and Retention. Therefore, select All Issues/All Groups from the business structure level.

Saving the configuration implements the business requirement.

On the U+ Bank website, when you log in as **Troy**, notice that the retention offer is displayed because **Troy** is predicted to churn in the near future.

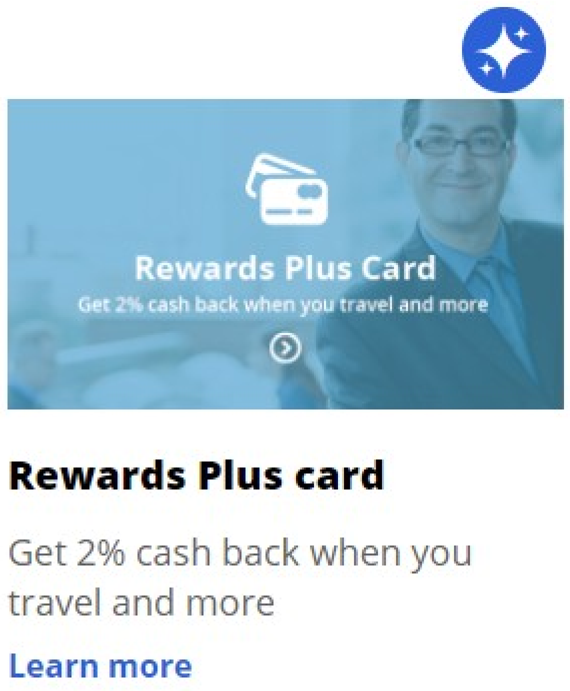


Extra miles 5K

5,000 extra miles

Learn more

Now, when you log in as **Barbara**, notice that the credit card offer is displayed because she is predicted to remain loyal for now.



Rewards Plus card

Get 2% cash back when you travel and more

Learn more

You have reached the end of this demo. What did it show you?

- How to use a prediction in a decision strategy
- How to arbitrate between different groups of actions to display more relevant offers to customers
- How to define applicability rules using a decision strategy in Next-Best-Action Designer

Creating parameterized predictors using a sub strategy

Description

Sub strategy component is used to execute a strategy within a different strategy. It builds the connection between strategies, access a specific component within a strategy, define how to run the strategy if the strategy is in another class and organize and simplify the design flow.

A sub strategy component defines which strategy to import or a specific component within the strategy which is connected to the results component.

Learn how to create a sub strategy and use it to map parametrized predictors to adaptive models in decision components. Input fields that are not directly available in the customer data model can be made accessible to the models by configuring these fields as parameterized predictors.

Learning objectives

- Create a parameterized predictor based on customer behavior data and score of a predictive model
- Create a sub strategy
- Use the sub strategy to map predictors into a adaptive model

Creating parameterized predictors

Introduction

Learn how to improve the predictive power of your adaptive models by creating parameterized predictors. Input fields that are not directly available in the customer data model can be made accessible to the models by configuring these fields as parameterized predictors.

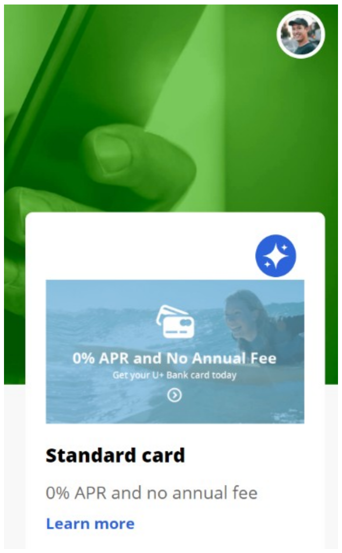
Challenge

To practice what you have learned in this topic, consider taking the Creating parameterized predictors challenge.

Transcript

This demo shows you how to create parameterized predictors for adaptive models.

U+ Bank shows personalized offers on their website when customers log in.

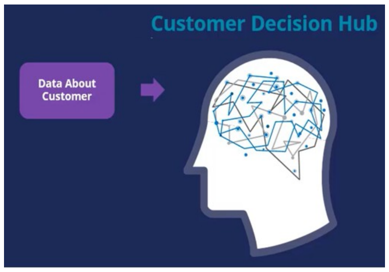


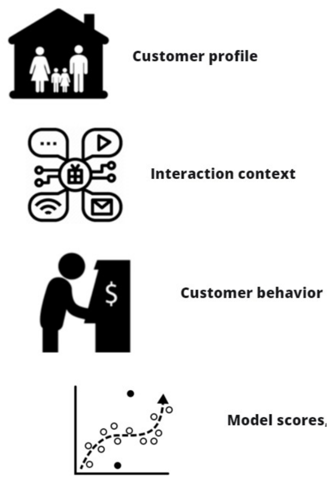
The bank relies on Pega Customer Decision Hub™ to decide which offer to show the customer.

Customer Decision Hub uses a prediction to predict the likelihood that a customer clicks on the offer.

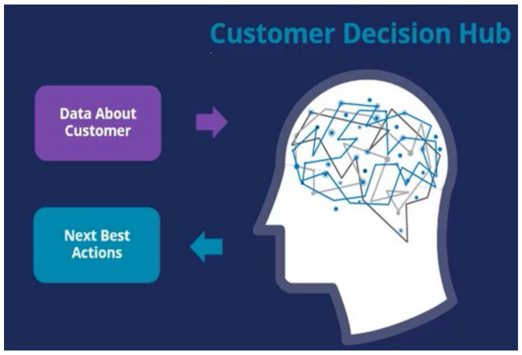
The prediction ingests data about the customer to calculate the propensity.

Ideally, this data includes the customer profile, interaction context, customer behavior and predictive model scores.





Customer Decision Hub weighs the propensity to decide which offer to show on the website, balancing customer relevance and business priority.



Input fields that are not directly available in the customer data model can be made accessible to the models by configuring these fields as parameterized predictors.

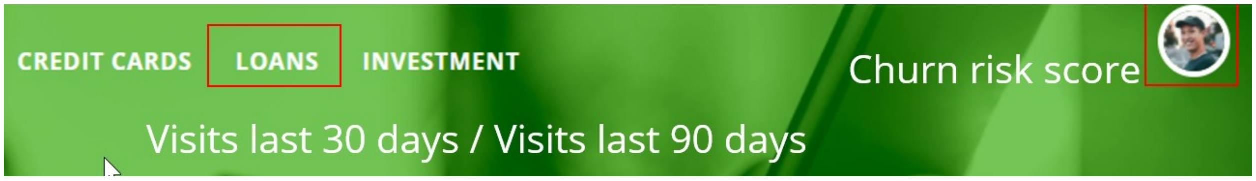
This video demonstrates the implementation of two new parameterized predictors that can add additional predictive power to the models.

The first predictor is the ratio of customer visits to the Loans web page in the last 30 days and in the last 90 days.

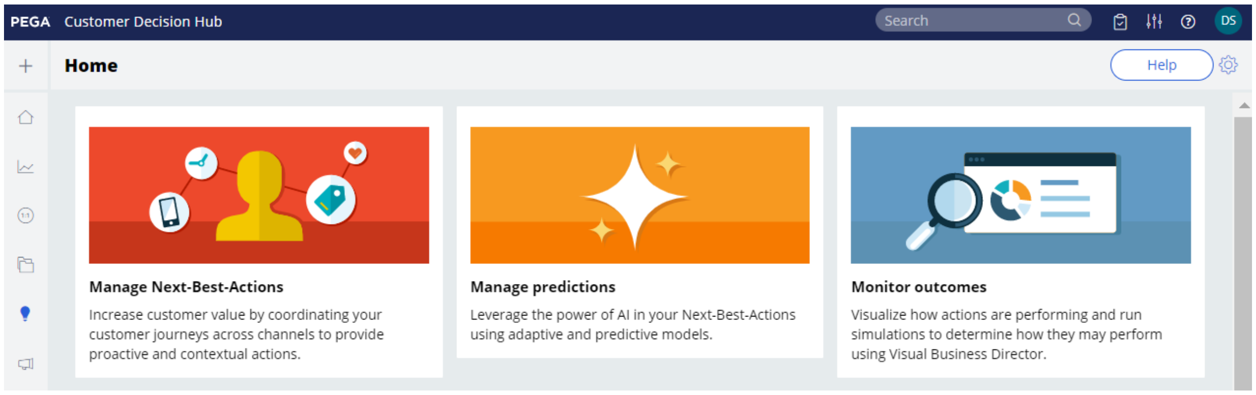
A high value may indicate the increasing interest of the customer in the content of this page.

The second predictor is the score of a churn model running in Customer Decision Hub.

A customer that is likely to churn in the near future may be interested in different offers than a customer that is predicted to remain loyal.



Managing predictions is a regular data scientist task.



Manage Next-Best-Actions

Increase customer value by coordinating your customer journeys across channels to provide proactive and contextual actions.

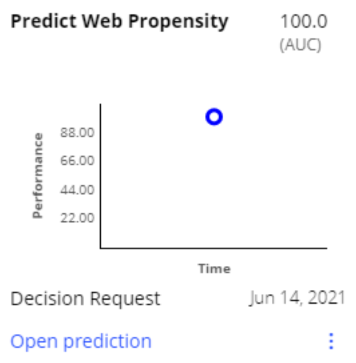
Manage predictions

Leverage the power of AI in your Next-Best-Actions using adaptive and predictive models.

Monitor outcomes

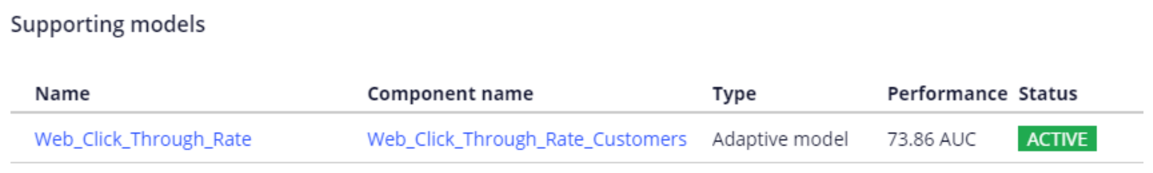
Visualize how actions are performing and run simulations to determine how they may perform using Visual Business Director.

The prediction that calculates the propensity that a customer will click on the offer on the U+ Bank website is the **Predict Web Propensity** prediction.



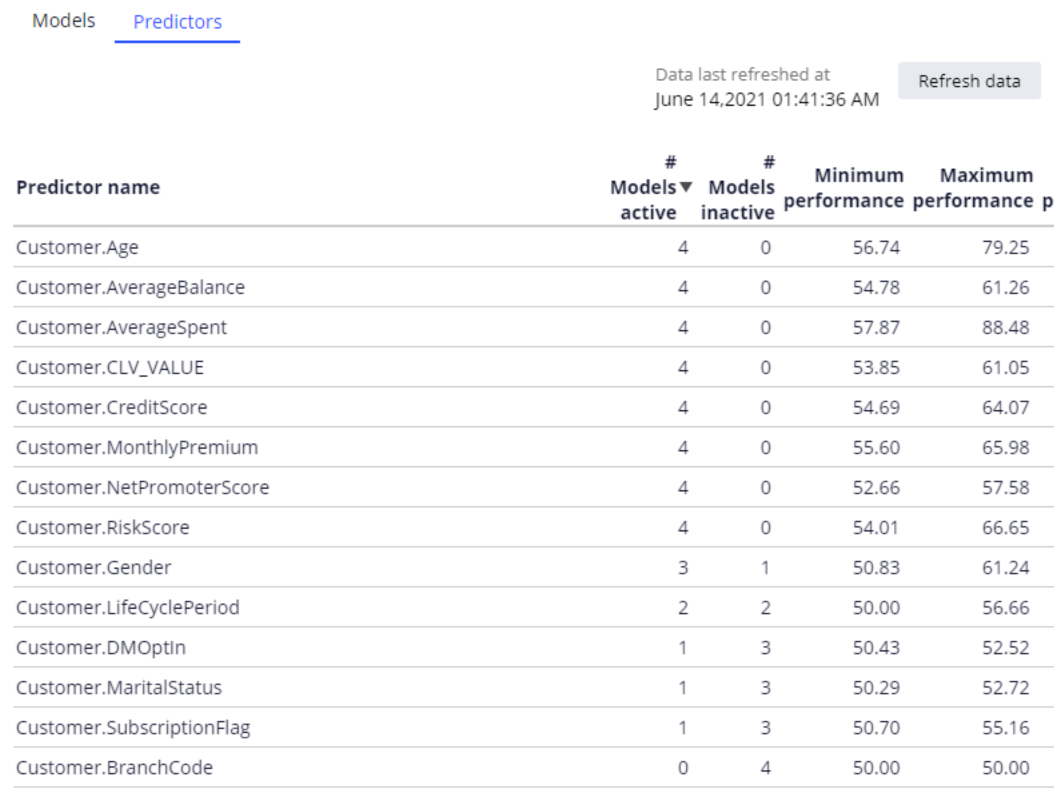
One or more predictive models drive a prediction.

The **Web Click Through Rate** adaptive model drives the **Predict Web Propensity** prediction.


Supporting models

Name	Component name	Type	Performance	Status
Web_Click_Through_Rate	Web_Click_Through_Rate_Customers	Adaptive model	73.86 AUC	ACTIVE

Adaptive models automatically determine which fields are used as predictors, based on the individual predictive performance and the correlation between active predictors.



Data last refreshed at
June 14,2021 01:41:36 AM

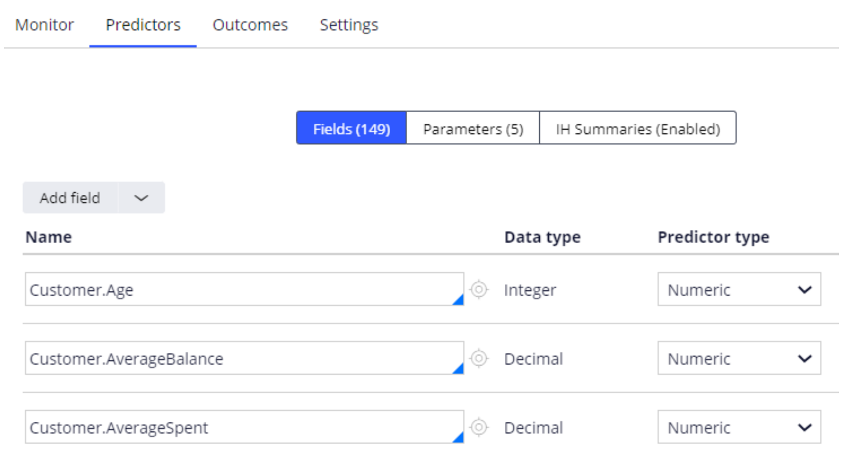
Predictor name	# Models ▼ active	# Models inactive	Minimum performance	Maximum performance	p
Customer.Age	4	0	56.74	79.25	
Customer.AverageBalance	4	0	54.78	61.26	
Customer.AverageSpent	4	0	57.87	88.48	
Customer.CLV_VALUE	4	0	53.85	61.05	
Customer.CreditScore	4	0	54.69	64.07	
Customer.MonthlyPremium	4	0	55.60	65.98	
Customer.NetPromoterScore	4	0	52.66	57.58	
Customer.RiskScore	4	0	54.01	66.65	
Customer.Gender	3	1	50.83	61.24	
Customer.LifeCyclePeriod	2	2	50.00	56.66	
Customer.DMOptIn	1	3	50.43	52.52	
Customer.MaritalStatus	1	3	50.29	52.72	
Customer.SubscriptionFlag	1	3	50.70	55.16	
Customer.BranchCode	0	4	50.00	50.00	

Only fields with a predictive performance above the threshold become active predictors in one or more models.

And, when predictors are highly correlated, they are grouped and only the best-performing predictor from the group is used.

It is therefore a best practice to make many uncorrelated fields available to the models as potential predictors.

In an adaptive model rule, three distinct types of predictors can be defined.

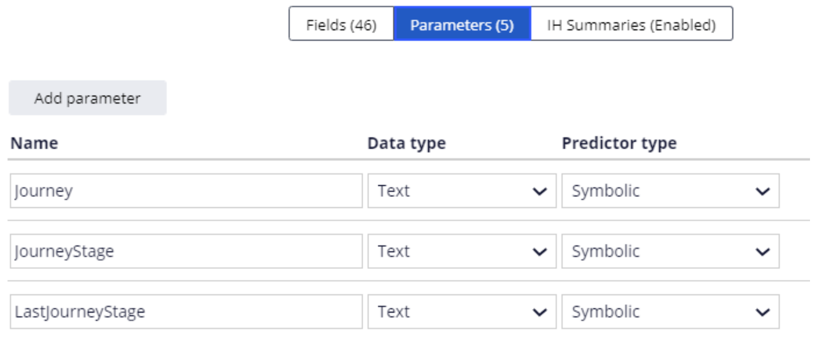


Name	Data type	Predictor type
Customer.Age	Integer	Numeric
Customer.AverageBalance	Decimal	Numeric
Customer.AverageSpent	Decimal	Numeric

The first predictor type concerns fields that contain customer attributes, such as age and average account balance, and customer behavior data summarized in Customer Profile Designer.

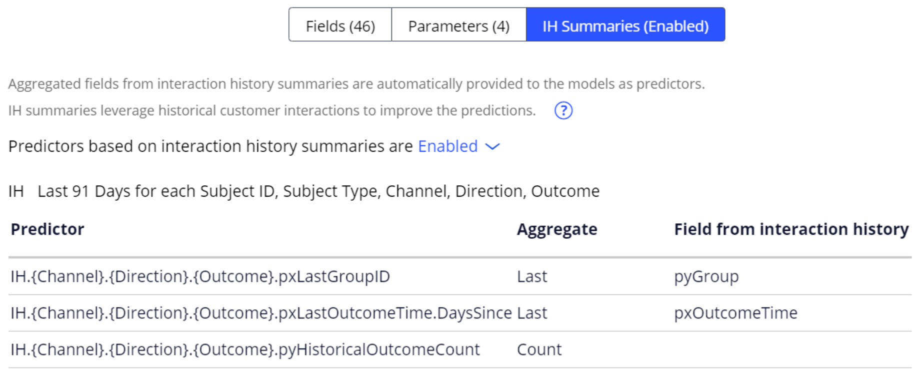
The second predictor type is parameterized to reference attributes that are not part of the customer profile.

Examples that can provide additional predictive power include derived fields, such as the time of day, and model scores.



Name	Data type	Predictor type
Journey	Text	Symbolic
JourneyStage	Text	Symbolic
LastJourneyStage	Text	Symbolic

The third predictor type is an Interaction History summary, which leverages historical customer interactions.



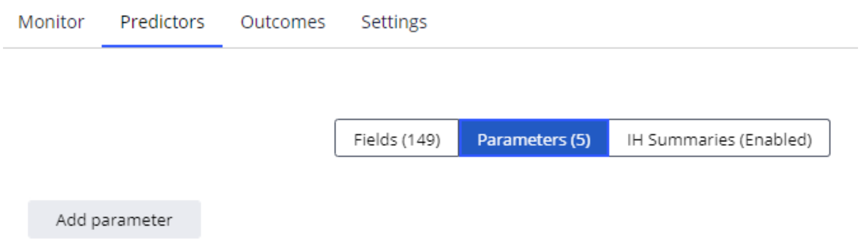
Aggregated fields from interaction history summaries are automatically provided to the models as predictors.
IH summaries leverage historical customer interactions to improve the predictions.

Predictors based on interaction history summaries are Enabled

IH Last 91 Days for each Subject ID, Subject Type, Channel, Direction, Outcome

Predictor	Aggregate	Field from interaction history
IH.{Channel}.{Direction}.{Outcome}.pxLastGroupID	Last	pyGroup
IH.{Channel}.{Direction}.{Outcome}.pxLastOutcomeTime.DaysSince	Last	pxOutcomeTime
IH.{Channel}.{Direction}.{Outcome}.pyHistoricalOutcomeCount	Count	

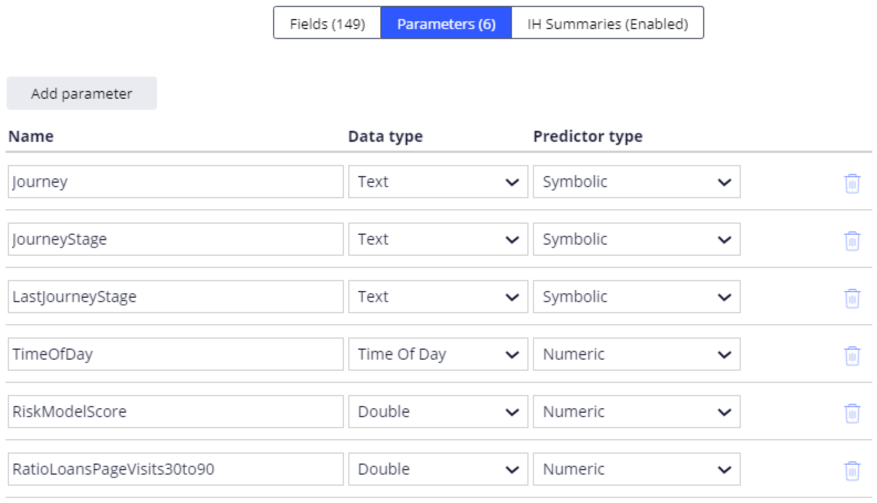
This demo focuses on parameterized predictors.



For adaptive learning, there is no difference between parameterized predictors and regular predictors.

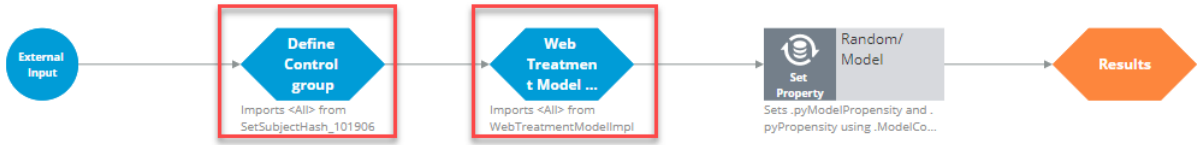
To create a parameterized predictor, you add it in the adaptive model rule.

In the example of *Loans page views 30 days-to-90 days ratio*, the data type is double.

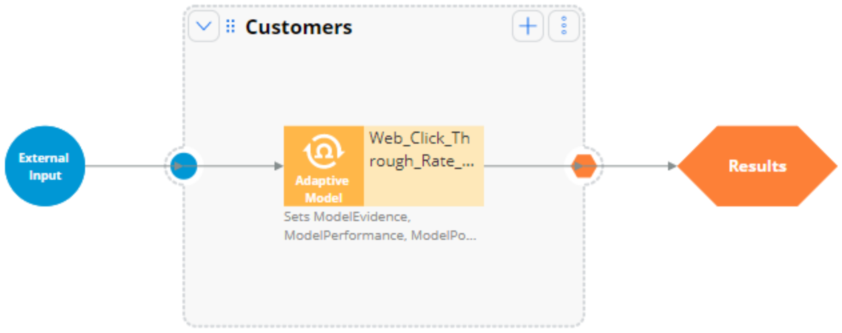


Name	Data type	Predictor type
Journey	Text	Symbolic
JourneyStage	Text	Symbolic
LastJourneyStage	Text	Symbolic
TimeOfDay	Time Of Day	Numeric
RiskModelScore	Double	Numeric
RatioLoansPageVisits30to90	Double	Numeric

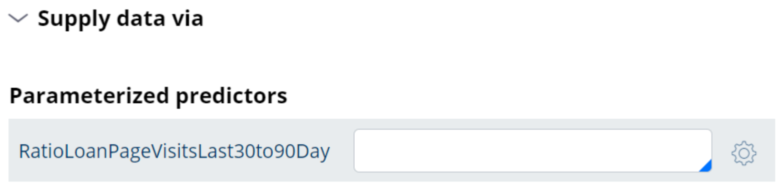
A prediction is implemented as a decision strategy.


External Input
Define Control group
Imports <All> from SetSubjectHash_101906
Web Treatmen t Model ...
Imports <All> from WebTreatmentModelImpl
Set Property
Random/ Model
Sets .pyModelPropensity and . pyPropensity using .ModelCo...
Results

The decision strategy defines the control group and contains a sub strategy that references the adaptive model rule that drives the prediction, in this case the **Web Click Through Rate** model.


Customers
External Input
Adaptive Model
Web_Click_Th rough_Rate_...
Sets ModelEvidence, ModelPerformance, ModelPo...
Results

The values of parameterized predictors are set in the adaptive model component in the decision strategy.



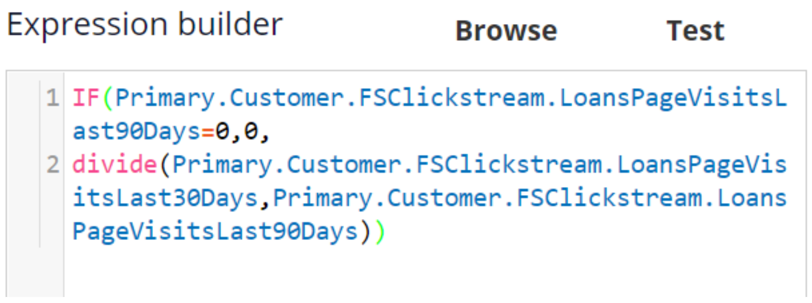
Supply data via

Parameterized predictors

RatioLoanPageVisitsLast30to90Day

The expression used for the new predictor says that if a customer has never visited the page in the last 90 days the value is set to zero ...

... otherwise it is set to the number of visits in the last 30 days divided by the number of visits in the last 90 days.

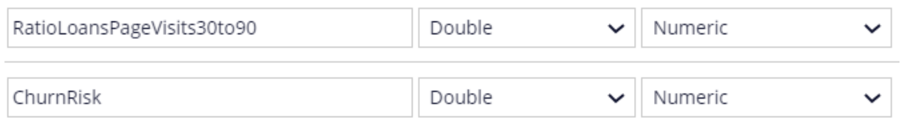


```
IF(Primary.Customer.FSClickstream.LoansPageVisitsLast90Days=0,0,
divide(Primary.Customer.FSClickstream.LoansPageVisitsLast30Days,Primary.Customer.FSClickstream.LoansPageVisitsLast90Days))
```

When you add a parameter in the model rule, it automatically enables the field for input in the adaptive model component.

In the second use case, you want to include the score of a churn model running in Customer Decision Hub as a potential predictor.

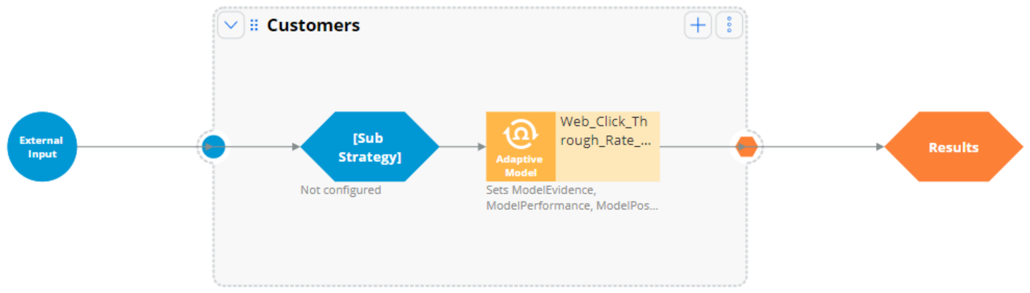
Just as in the first use case, start by adding the new parameter to the adaptive model.



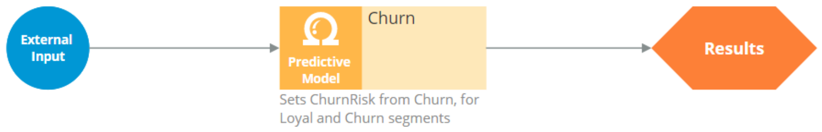
RatioLoansPageVisits30to90	Double	Numeric
ChurnRisk	Double	Numeric

For this use case, you need to alter the prediction decision strategy.

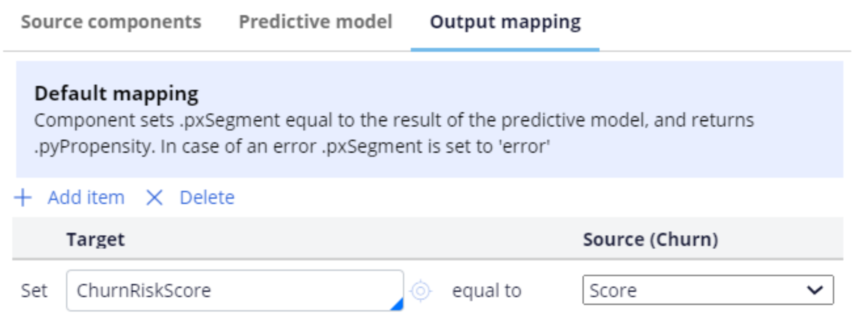
To access the model scores, you add an external sub strategy that returns the score of a predictive model that calculates the churn risk of a customer.


Customers
External Input
[Sub Strategy]
Not configured
Adaptive Model
Web_Click_Through_Rate_...
Sets ModelEvidence, ModelPerformance, ModelPos...
Results

Create a new sub strategy that references the churn model on the customer page ...



... and map the score of the model to a new property in the Strategy Result class.



Source components | Predictive model | **Output mapping**

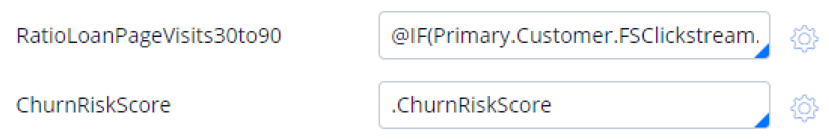
Default mapping
Component sets .pxSegment equal to the result of the predictive model, and returns .pyPropensity. In case of an error .pxSegment is set to 'error'

+ Add item × Delete

	Target		Source (Churn)
Set	ChurnRiskScore	equal to	Score

The **Score** output field of the churn model is a numeric field with values from 0 to 1000. A high value indicates a high churn risk.

You can now use this **ChurnRiskScore** property to populate the predictor in the adaptive model component in the decision strategy.

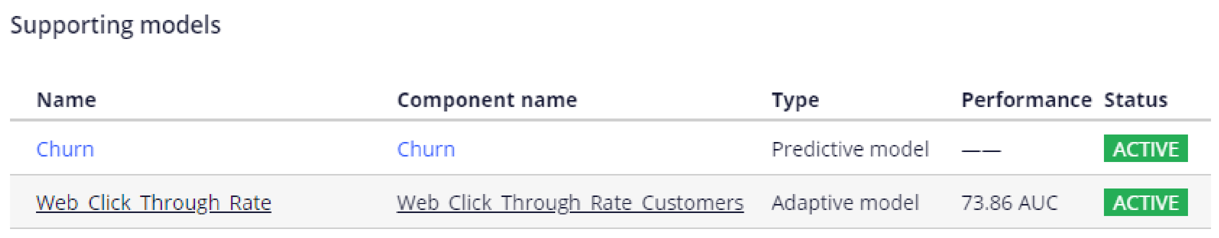


RatioLoanPageVisits30to90	@IF(Primary.Customer.FSClickstream.
ChurnRiskScore	.ChurnRiskScore

After refreshing the data, the two parameterized predictors are available to the models.

They are currently inactive, but they will become active predictors over time, when they prove to have predictive power.

The Churn model is now one of the supporting models in the prediction that calculates the likelihood that a customer clicks a specific offer.



Supporting models

Name	Component name	Type	Performance	Status
Churn	Churn	Predictive model	——	ACTIVE
Web_Click_Through_Rate	Web_Click_Through_Rate_Customers	Adaptive model	73.86 AUC	ACTIVE

You have reached the end of this demo. What did it show you?

How to create parameterized predictors