Cross-Sell on the Web Extended

STUDENT GUIDE

PEGA Academy

This document is the property of:
Pegasystems Inc.
1 Rogers Street
Cambridge, MA 02142
Phone: (617) 374-9600
Fax: (617) 374-9620
www.pega.com

Mission: Cross-Sell on the Web Extended

Product: Pega Customer Decision Hub™ 25

URL: Cross-Sell on the Web Extended | Pega Academy

Date: 06 August 2026

Contents

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

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

Transcript

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.

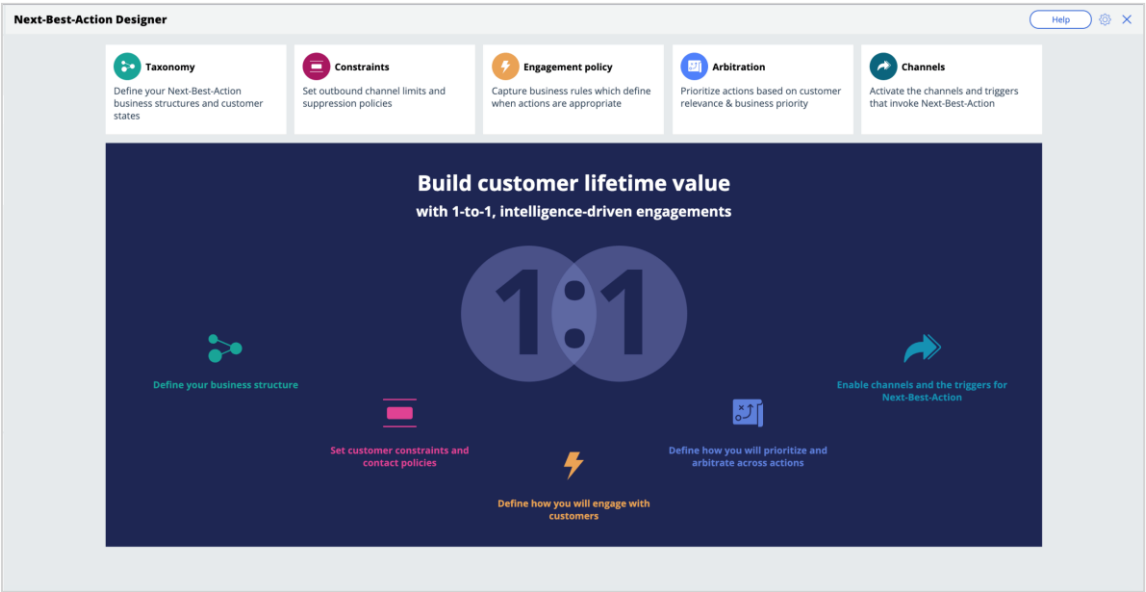
Next-Best-Action Designer guides you through the creation of a Next-Best-Action strategy for your business.

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.

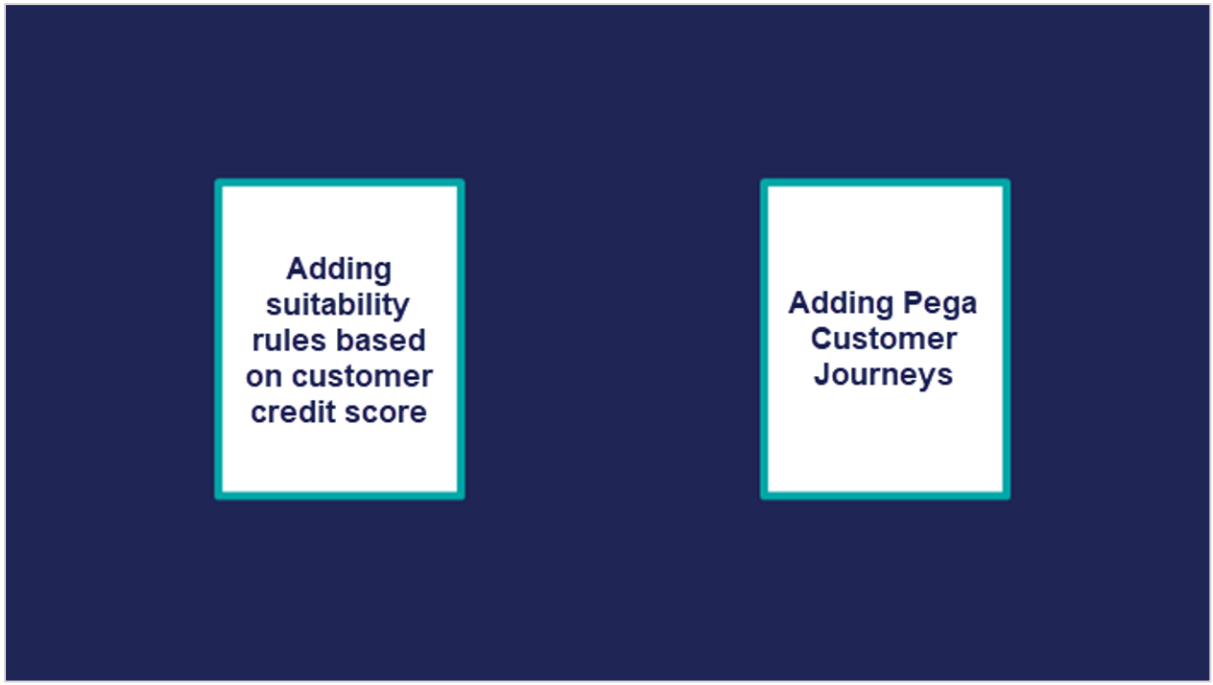
This framework leverages best practices to automatically generate Next-Best-Action decision strategies at the enterprise level.

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.



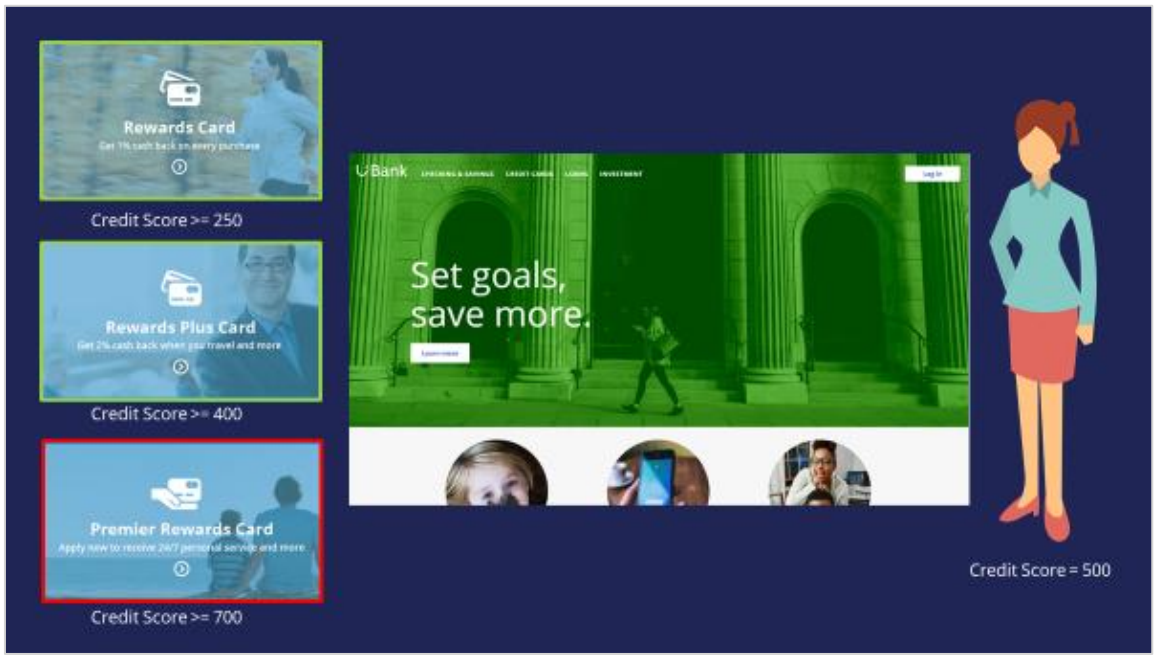
U+ Bank wants to implement additional use cases to meet new business requirements. These use cases require U+ bank to extend Next-Best-Action Designer capabilities.



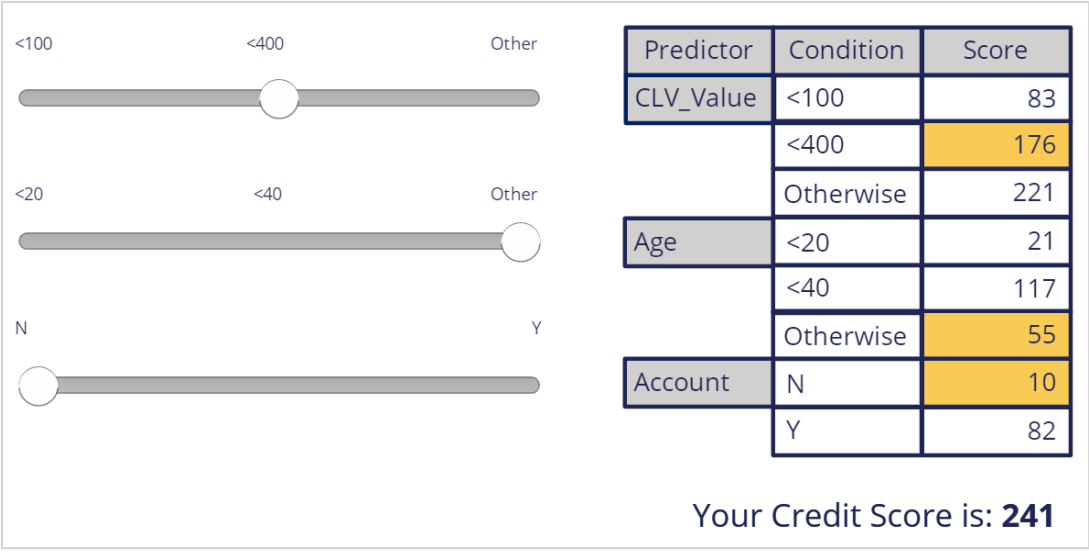
The first use case is to create suitability rules based on a customer's credit score. The bank wants to determine whether or not a customer is suitable for an offer based on their credit score. In this scenario, the credit score is not available, it needs to be computed in real-time based on customer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

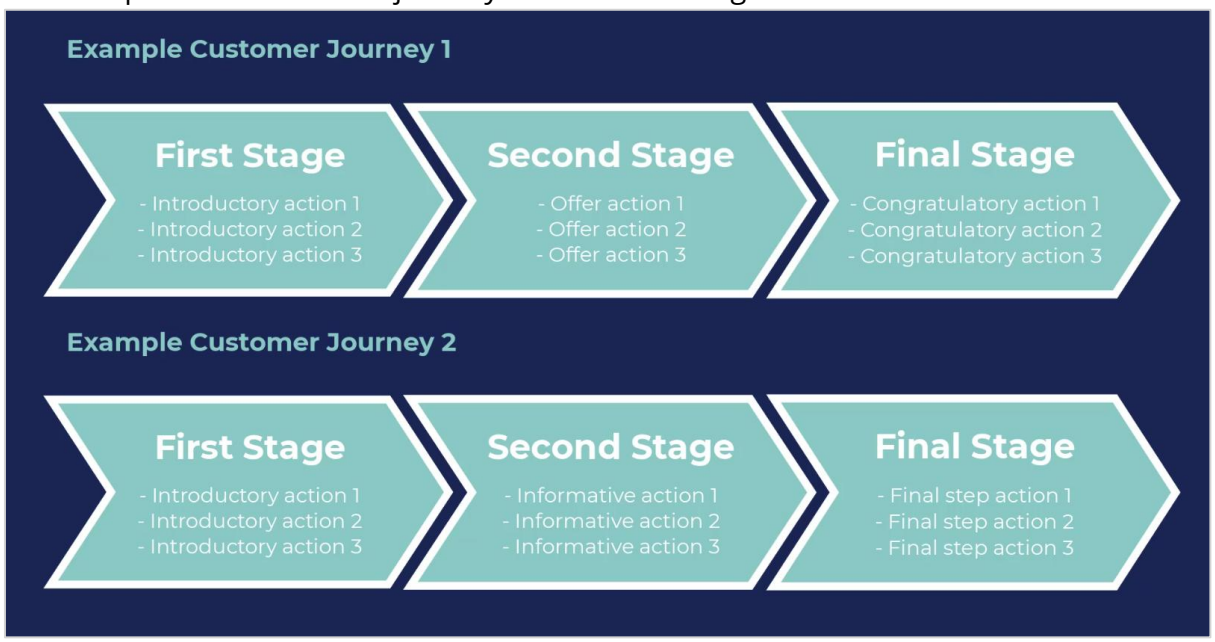


To determine suitability, the bank wants to calculate a customer's credit score using a scorecard. A scorecard is used to assign importance to pieces of data for use in a calculation. A scorecard uses a subset of customer property values divided into ranges and assigns scores to each range to compute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

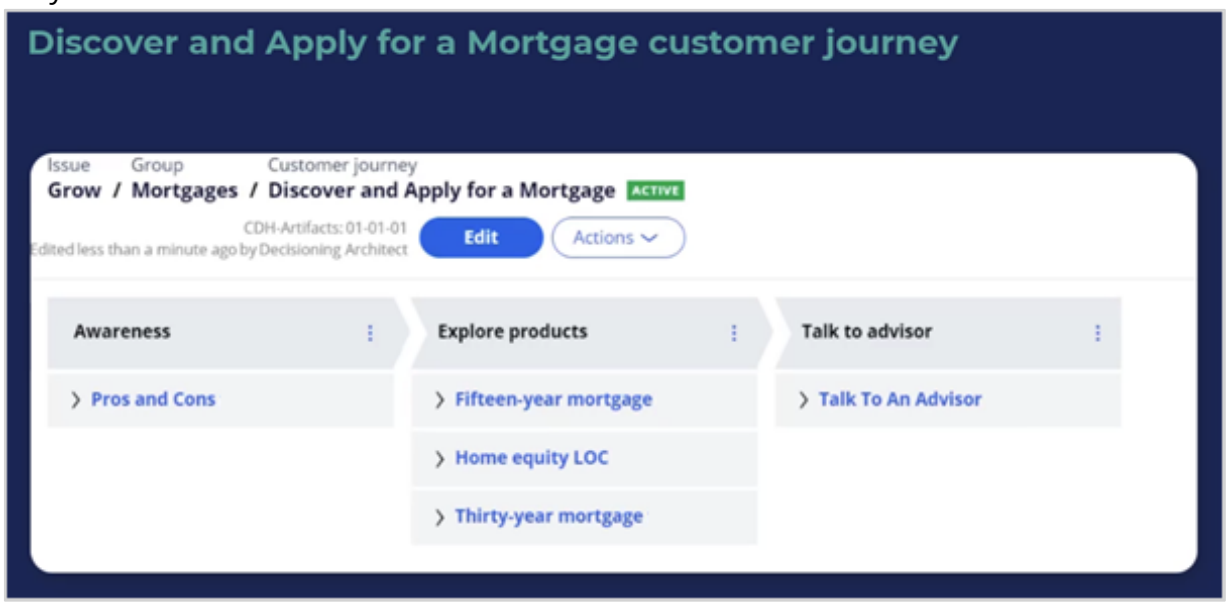


To implement this, you use a special Suitability condition in Next-Best-Action Designer. The Suitability condition uses a decision strategy that references a Scorecard rule. The Scorecard rule is used to determine the customer's credit score.

In the next use case, the bank wants to regulate and document the full customer experience. To implement this business requirement, you use Pega Customer Journeys. You can set up various customer journeys divided into stages that contain a number of actions.



The customers can advance the journey by moving to the next stage when they meet the required entry criteria. Customer journeys provide a good mental model for managing content and influencing next-best-action decisions to present suitable actions seamlessly for your customers.



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.

Decision Strategies Overview

Description

Decision strategies optimize business processes by using data-driven, real-time arbitration to select the best actions for specific contexts. This module explains how to create and run a decision strategy, detailing the use of various decision strategy components. It covers the importance of understanding component interactions, building expressions, and testing strategies to ensure customer-centric interactions that enhance customer experience and business outcomes.

Learning objectives

- Create and configure decision strategies from scratch using various decision components.
- Utilize various components to build effective decision strategies.
- Develop and validate expressions within decision strategies to calculate and prioritize actions.
- Test and refine decision strategies to ensure they meet business requirements and improve customer interactions.

Creating decision strategies

Introduction

You can optimize decision points in a business process or in a case life cycle with the help of decision strategies. Decision strategies allow you to perform data-driven, real-time arbitration to select the best action for a given context. 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.

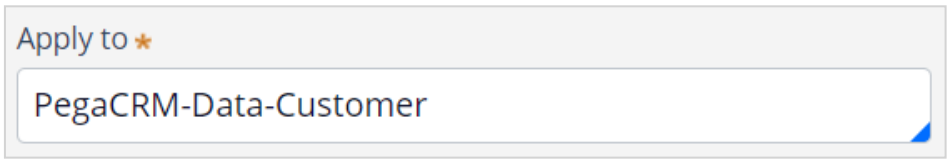
The Strategy-Results class limits the output of the strategy to the actions contained in the Business issue and Group.

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.

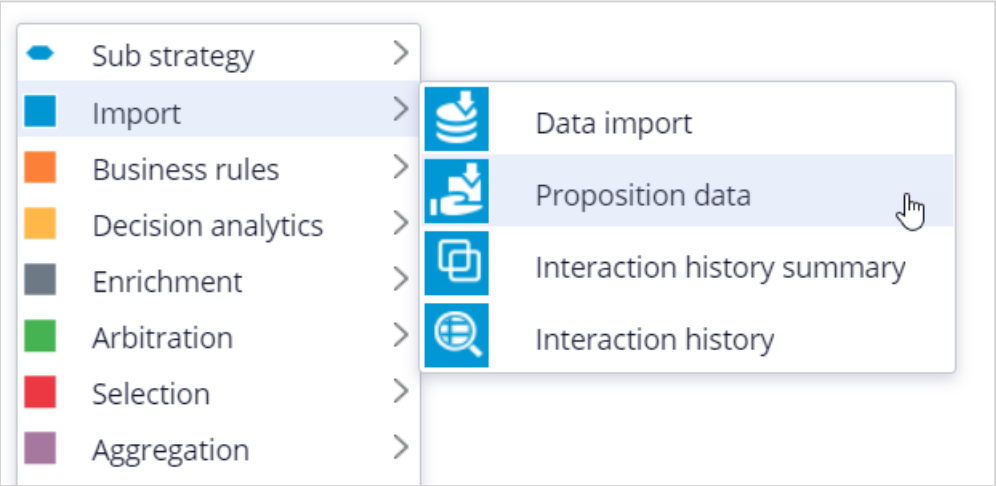


You can now start building the strategy. Right-click on the canvas to get the Context menu, which shows all component categories.

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.

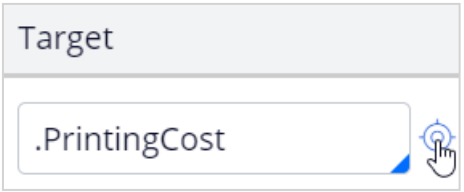
Click 'Add Item' to create the equation that will calculate the printing cost for each of the components.

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.

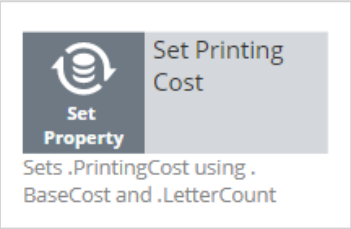
For now, you calculate the printing cost for each action that does not use customer properties.

Finalize the Expression.

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.



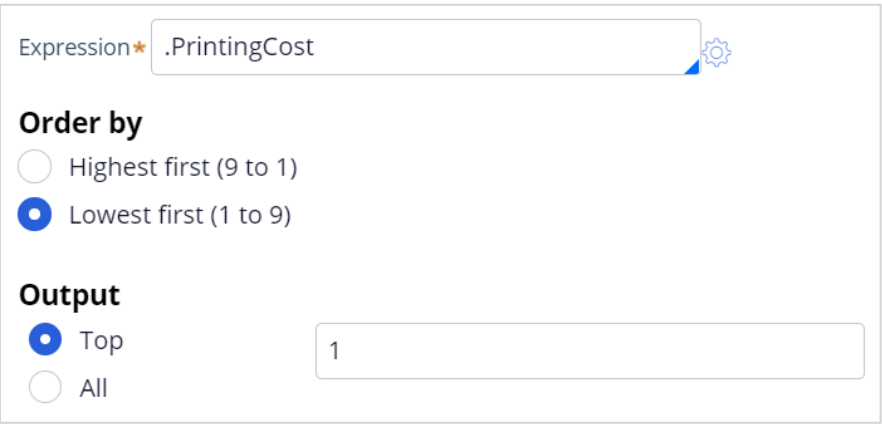
Now you want to ensure that the actions will be prioritized based on the lowest printing cost. So, you need to add the Prioritize component from the Arbitration category.

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.

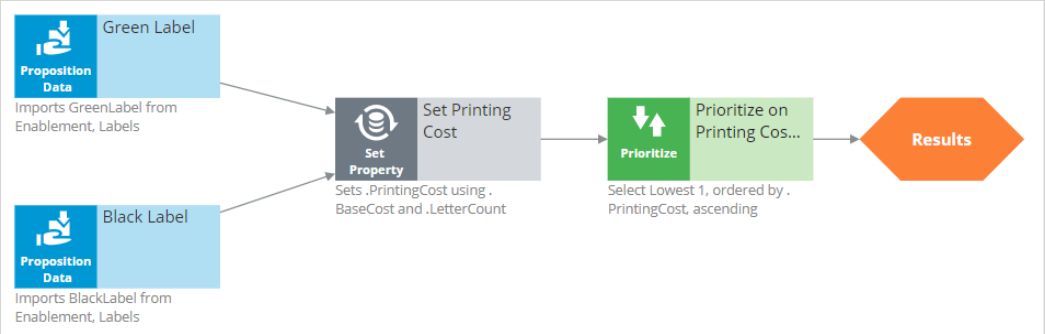
Here you can select the order in which the top actions are presented. Since you are interested in the lowest printing costs, configure it accordingly.

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.



Now you can connect the components and save the strategy.



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

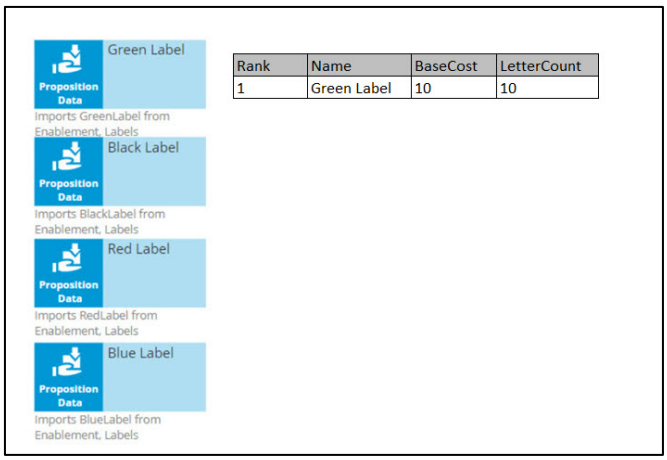
This demo explains what's going on inside each component when a Decision Strategy is executed.

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.

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.



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.

Rank	Name	BaseCost	LetterCount
1	Green Label	10	10

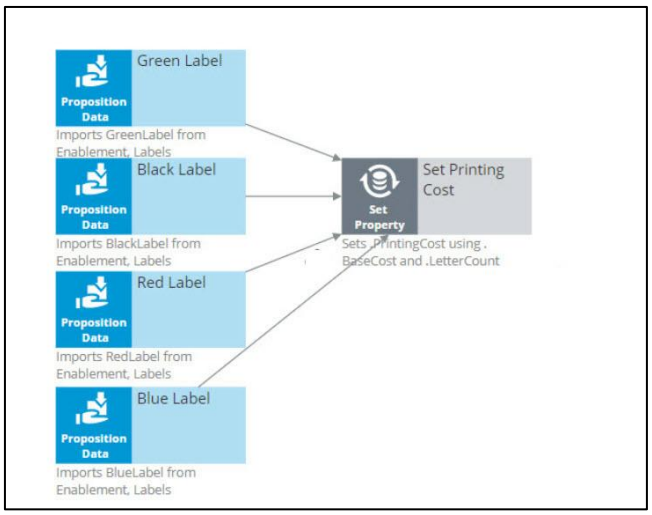
Rank	Name	BaseCost	LetterCount
1	Black Label	20	10

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.

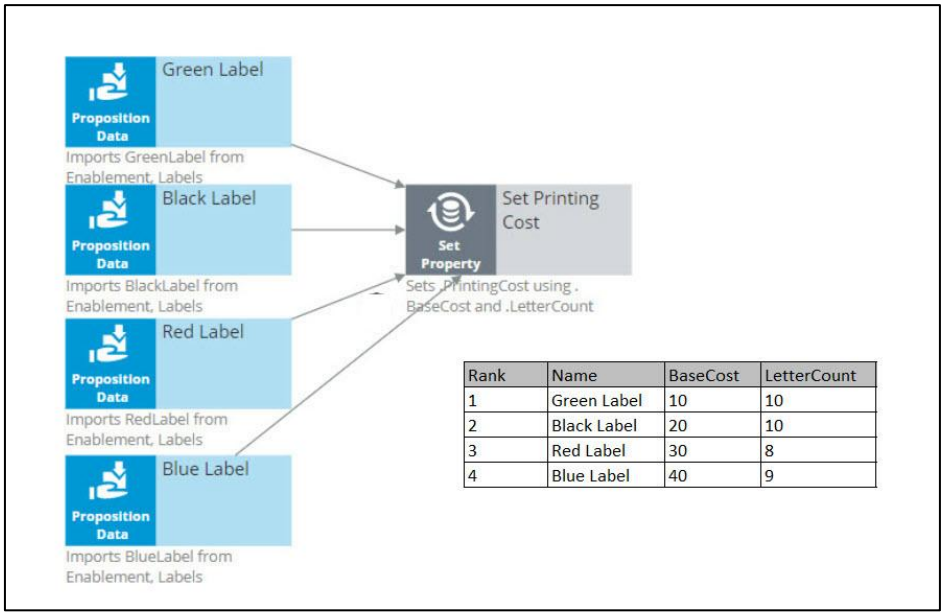
On the strategy canvas, components are connected by drawing arrows from component to component. So, what do these arrows mean exactly?



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.



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

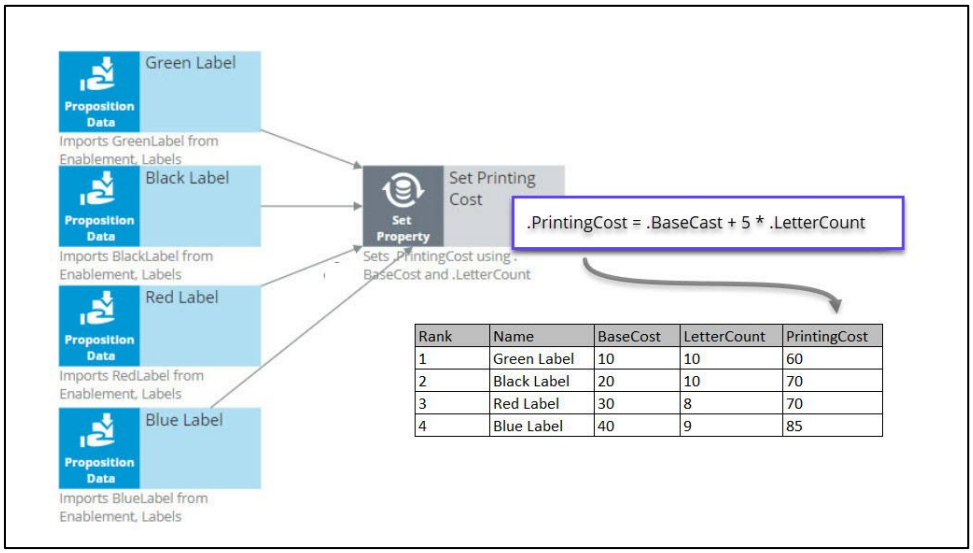
In the Set Property component, the Rank is determined by the order in which the actions are received by the component.

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.



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

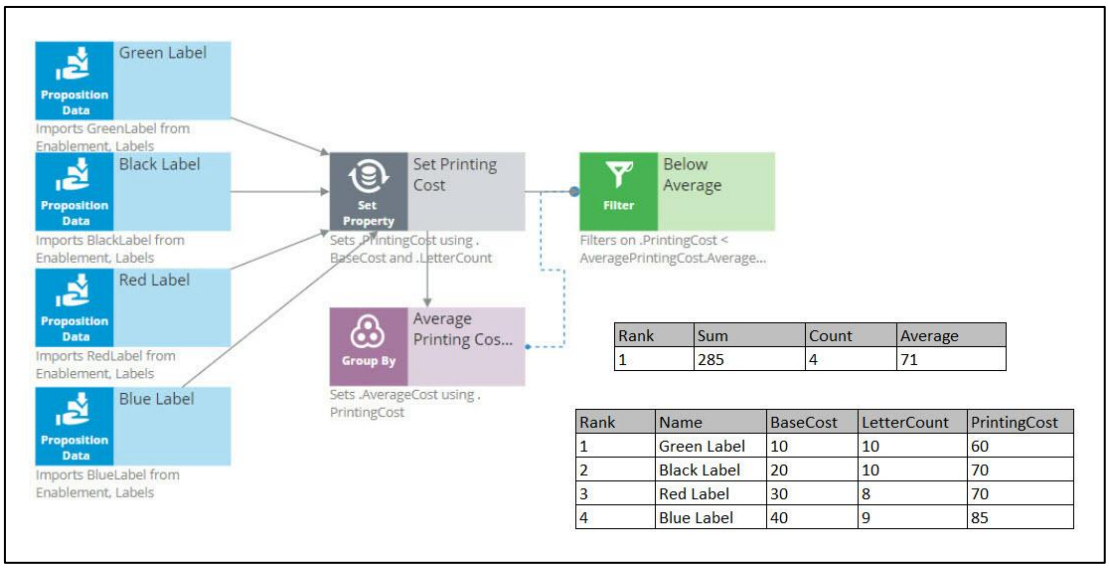
Because we are combining the data in our four Proposition Data components into one Set Property component, we only need to add one PrintingCost property to the new component, and it automatically computes the printing cost for all four actions.

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.



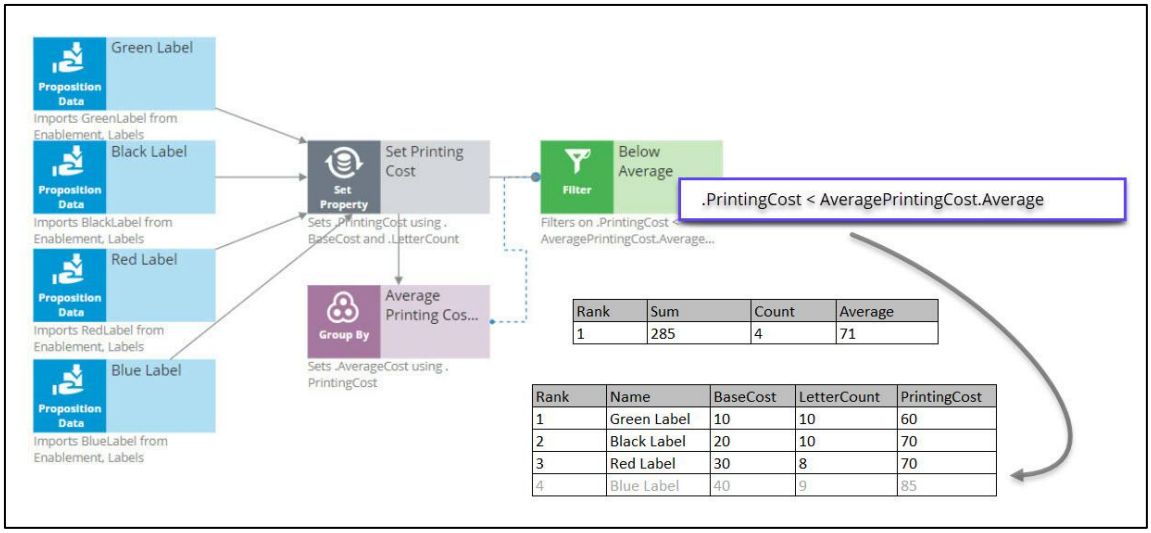
Rank	Sum	Count	Average
1	285	4	71

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.



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.

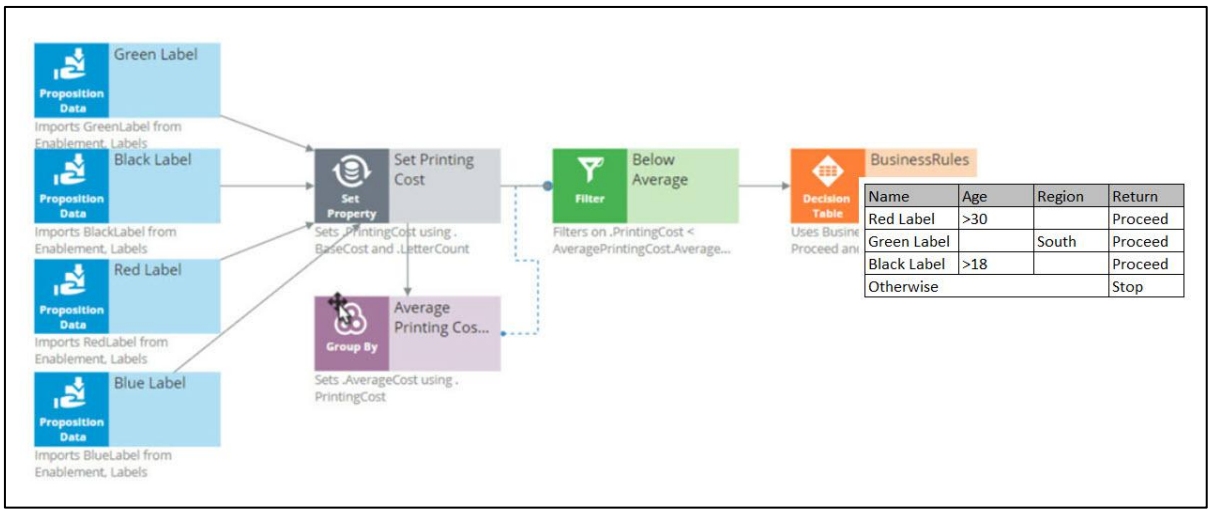
The Rank 1 average equals 71 in the ‘Group by’ component. This means that the filter will allow Label actions through that have a printing cost lower than 71.

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.



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.

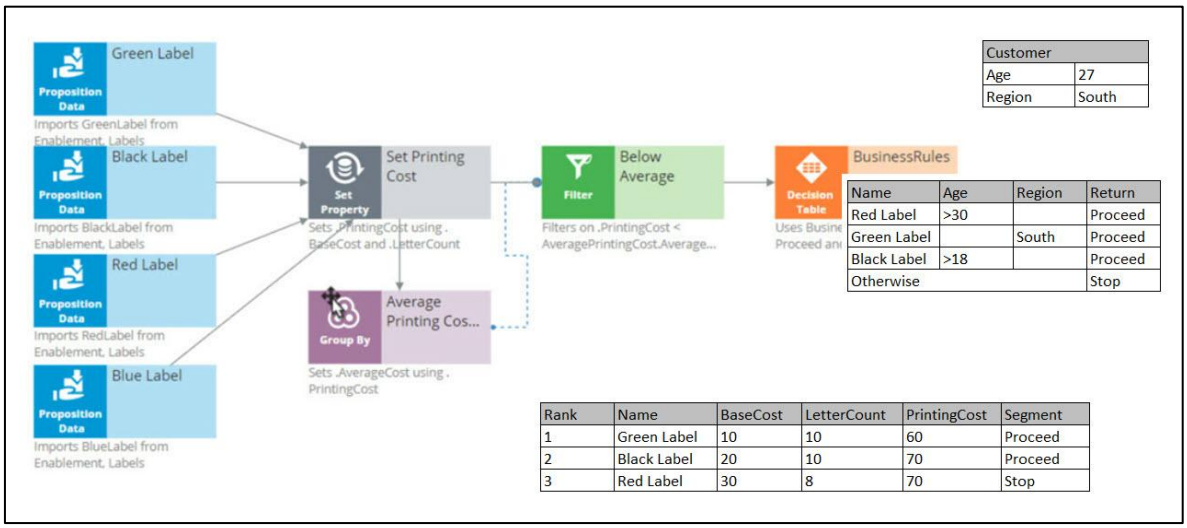
The business criteria say that the Red Label action can be offered if the customer's age is over 30 and they are from any region. If these criteria are met, the Return value is 'Proceed'.

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.



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.

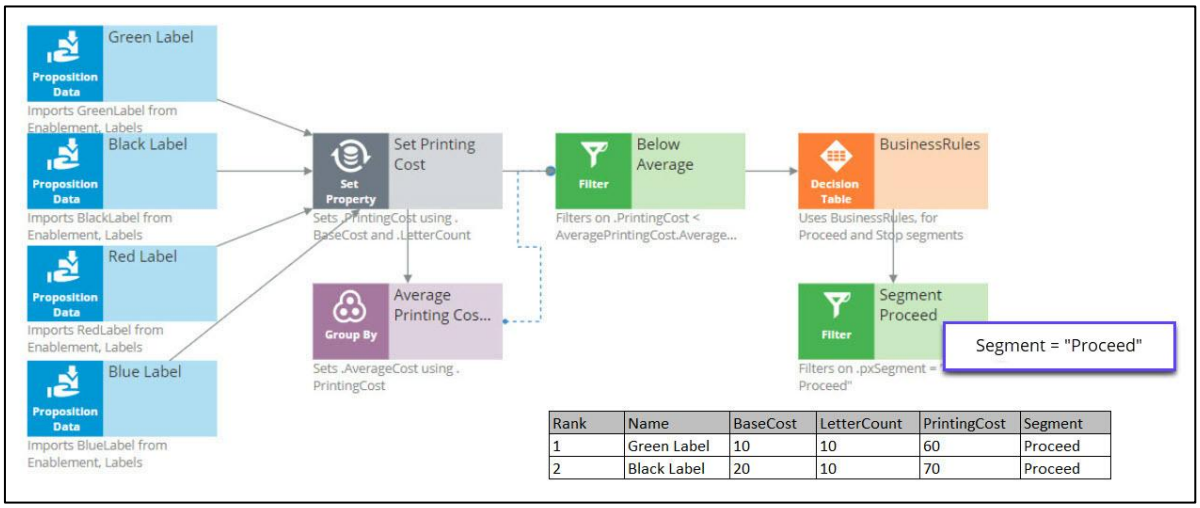
Next, it looks at the Black action and sees that the criteria for Black is that the customer's age is greater than 18. This customer is 27.

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.



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.

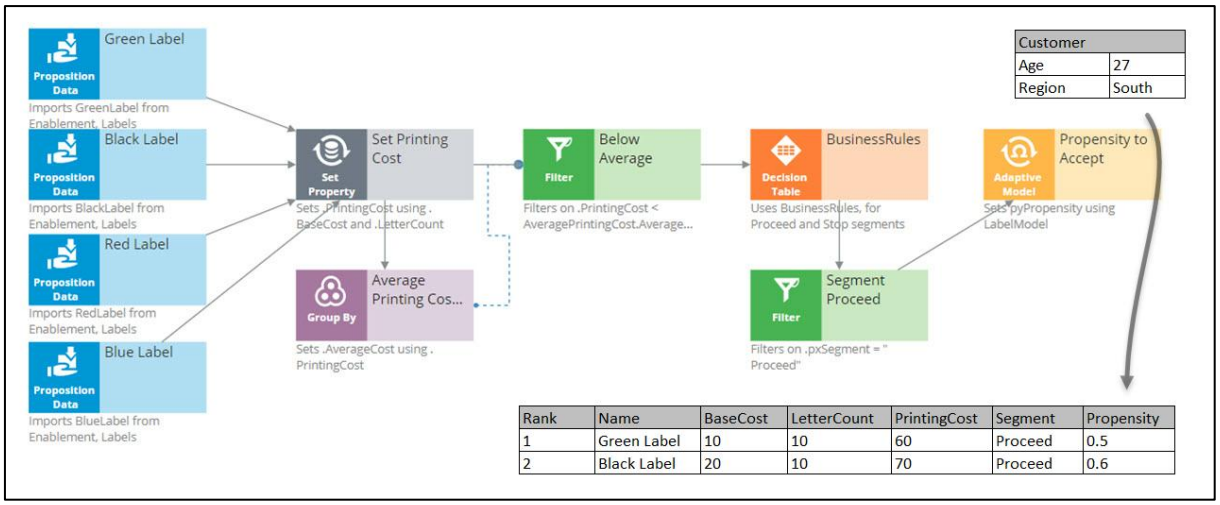
First is the Green Label. Green is allowed through, which means its properties will be available in the new component.

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.



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.

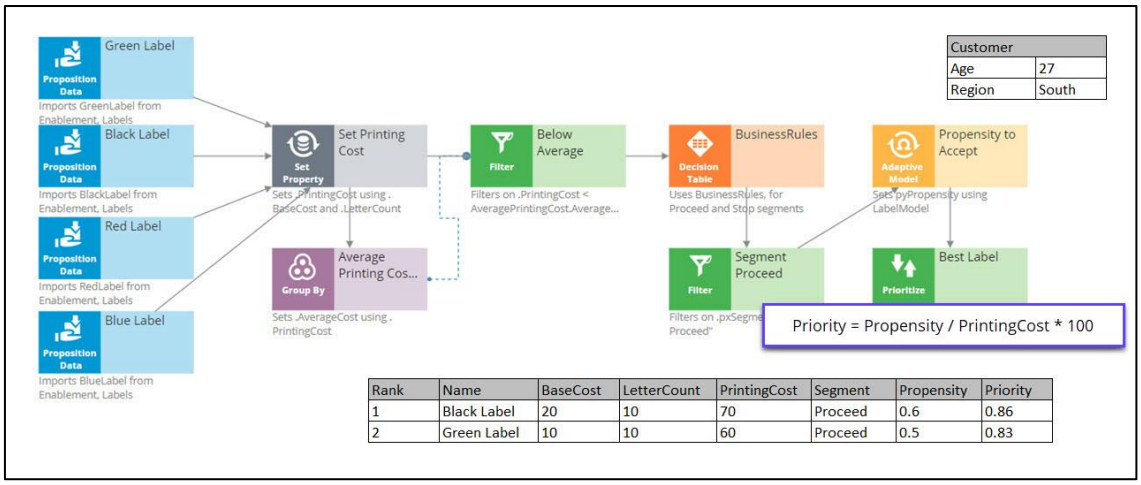
For the Black Label action, the likelihood turns out to be 0.6.

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.



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.

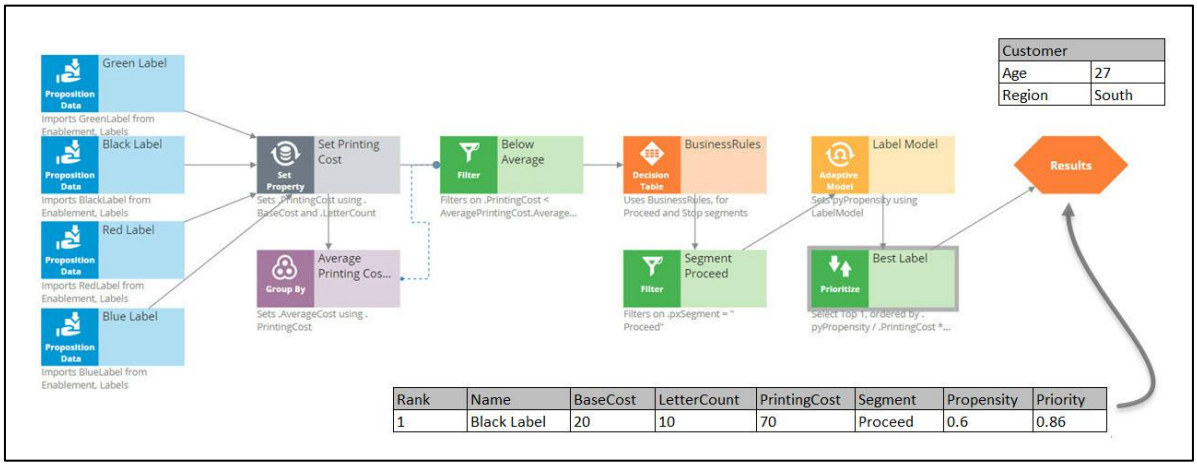
Because 0.86 is higher than 0.83, the Black Label action is now ranked number one.

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.



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.

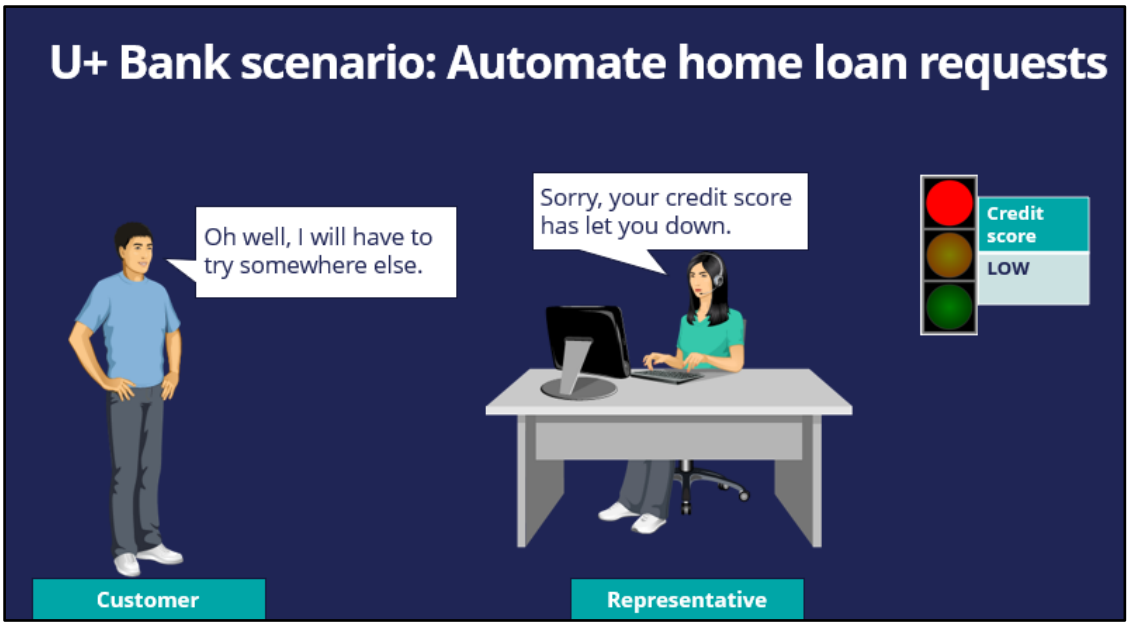
Tom is a U+ Bank customer who visits the bank to apply for a home loan. Iris is a U+ bank loan representative who is responsible for processing loan requests.

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

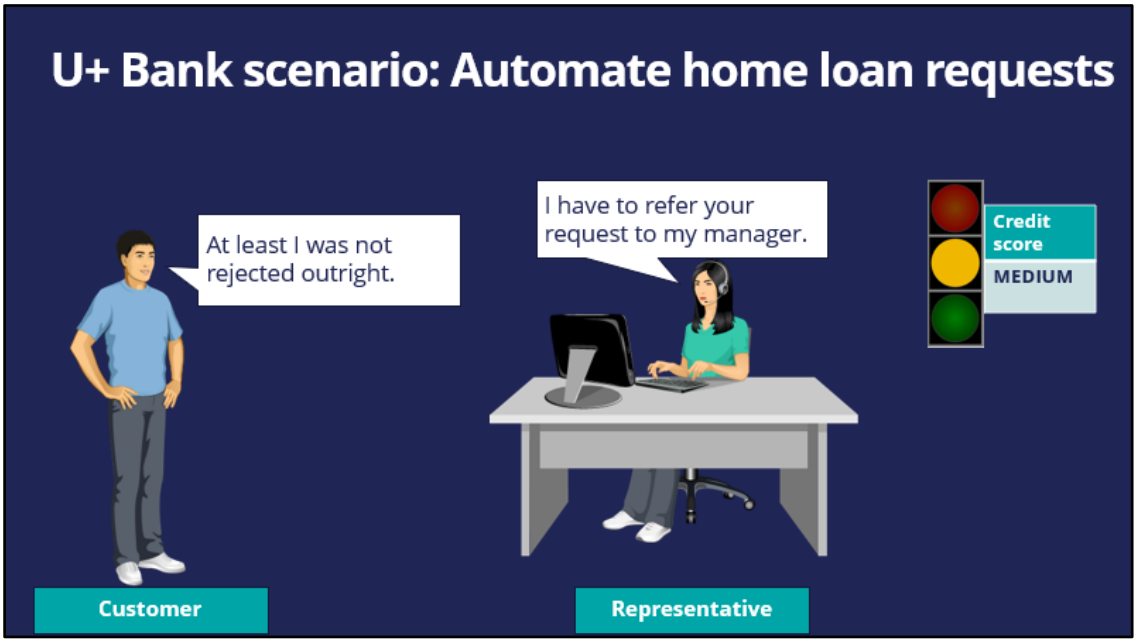
Depending on Tom's credit score, a decision is made on his loan application.

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.

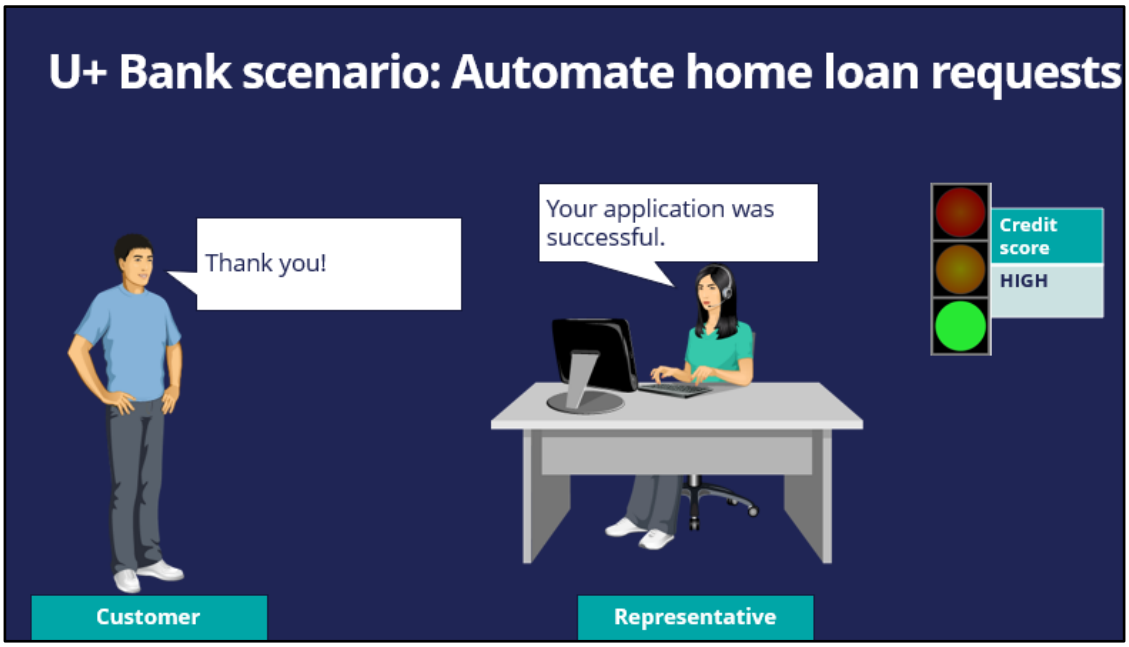


If Tom's credit score is **MEDIUM**, then his application needs to be reviewed further by manager.

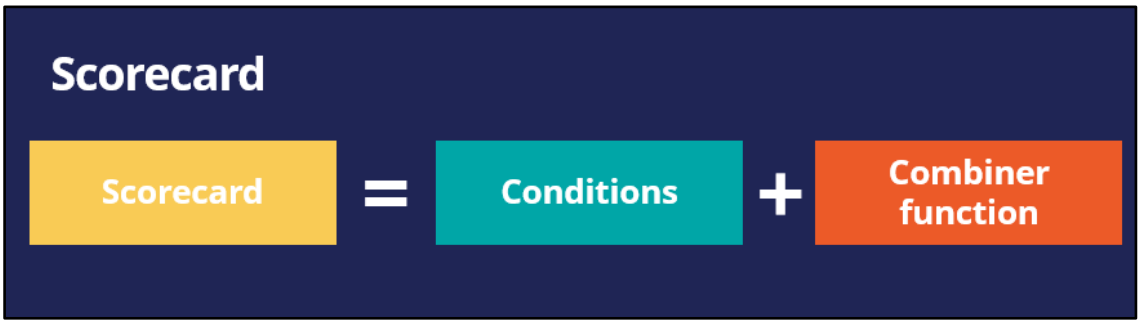


If Tom's credit score is **HIGH,** then he is automatically eligible to get the loan right away.

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



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.

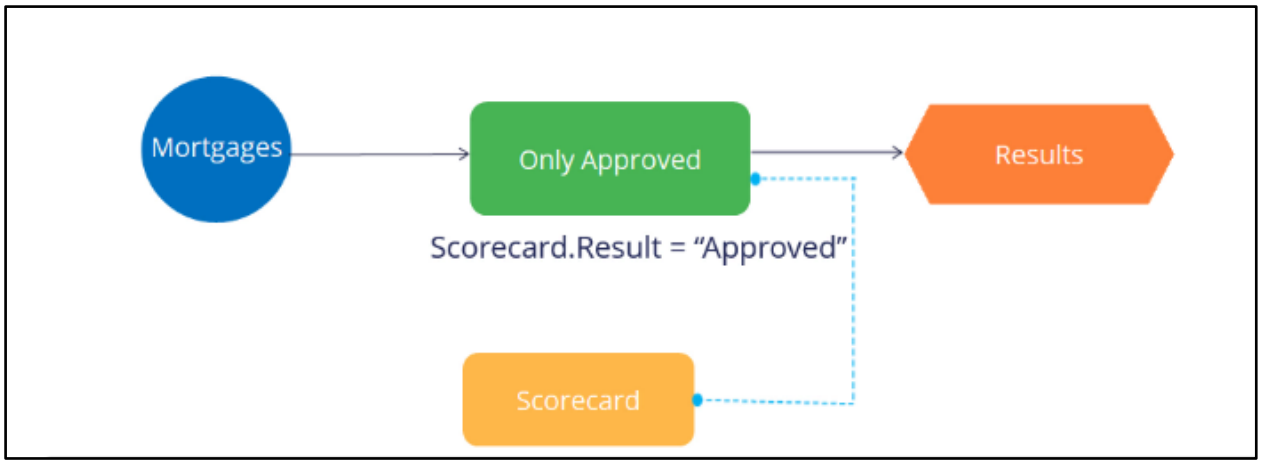
Customer A has a customer lifetime value of 150, their age is 18, and they don't have a U+ Bank account. Therefore, their credit score is 252, and the segment result is **Not Approved**, because their credit score value is less than 400. This means they do not qualify for a loan.

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.

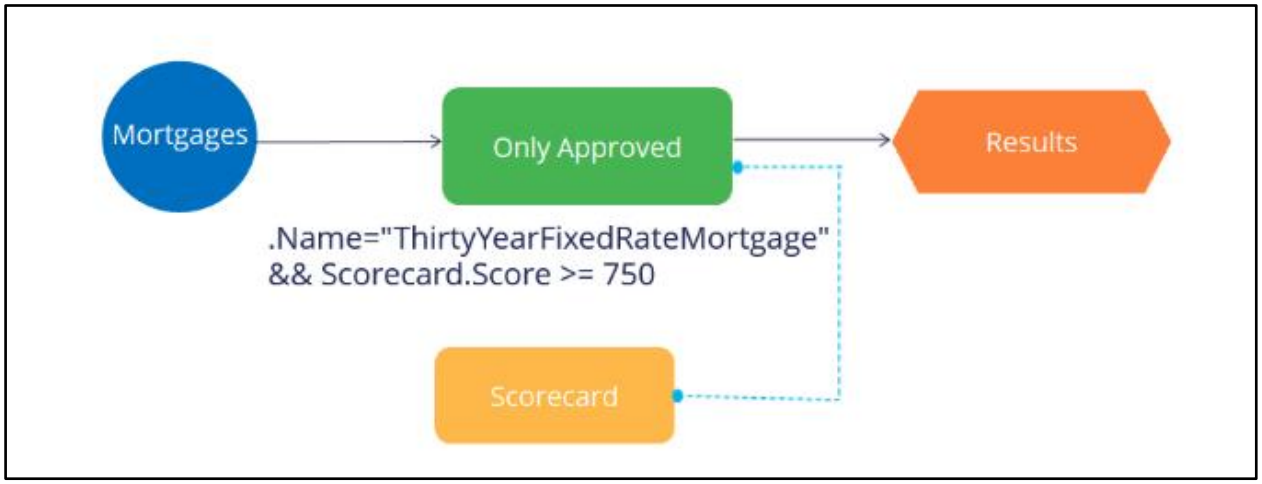
Customer C has a credit score of 649, which is higher than 600, so he is eligible to get a loan right away.

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.



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

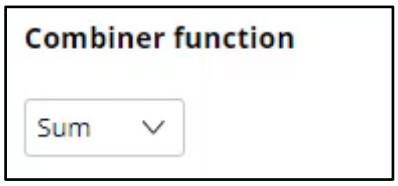
U+ bank wants to build a scorecard that calculates the credit score of a customer to determine if the customer is suitable for a mortgage or not.

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.



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.

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.

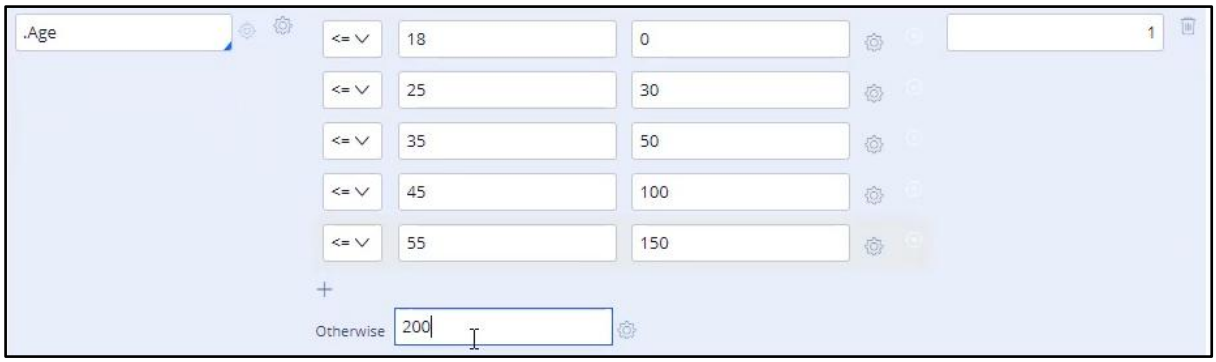
The data model contains an Income property, which contains the monthly income of the customer. The scorecard allows you to use this property to build the expression to compute the customer's yearly income.

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.

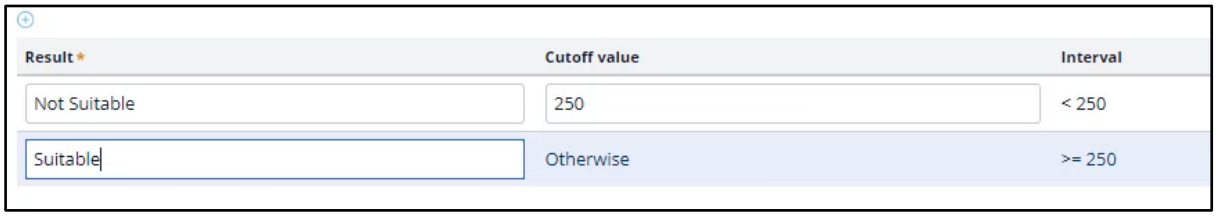
In a similar way, split the values for Age into five ranges and assign a score to each range. The higher the range, the higher the score you assign.



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.

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'.



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 an engagement strategy

This demo will show you how to build a decision strategy that can be used in Next-Best-Action Designer to implement more complex Engagement Policy rules.

Engagement Policy rules are business rules that describe which Actions are appropriate for a customer, expressed in the form of either Eligibility, Applicability, or Suitability rules.

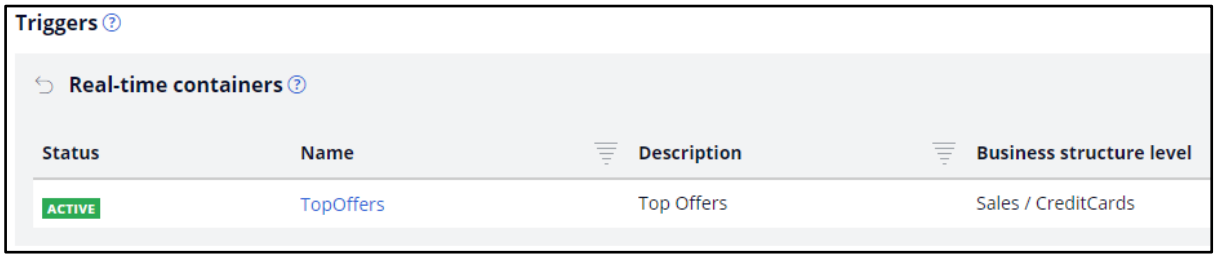
In this video, we will demonstrate the Engagement Policy capability without altering the use case.

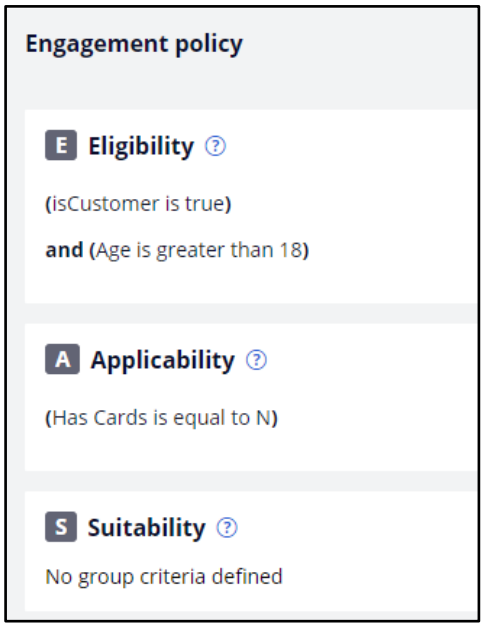
Currently, U+ Bank is doing cross-sell on the web by showing various credit cards to its customers. Every time Troy logs in to his accounts page, a credit card offer is shown.

U+ has a new set of Eligibility rules. As a result, Troy only qualifies for the Standard Card offer.

To see the existing configuration, navigate to Next-Best-Action Designer -> Channels.

As you can see, U+ Bank's website invokes the TopOffers real-time container to present offers from the Sales->CreditCards group.

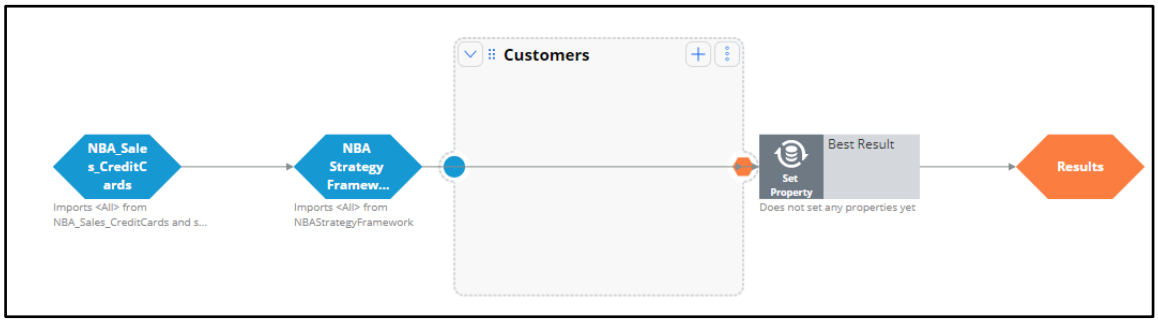




In the Engagement Policy, the Eligibility and Applicability rules have been defined to reflect the bank's current requirements using properties in the criteria.

Sometimes, implementing a specific use case requires a decision strategy.

To understand how the decision strategy is used to enforce Engagement Policy rules, open the Trigger_NBA_Sales_CreditCards strategy. This strategy is associated with the TopOffers real-time container. Every time a decision is requested from the U+ Bank website, this strategy is executed.

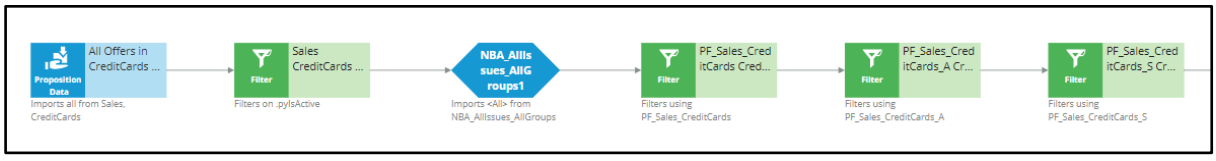


The first component of the strategy is responsible for all Engagement Policy rules.

If you open the strategy, notice that it imports the Actions from the Sales->CreditCards group and then applies the Engagement Policy configuration rules: Eligibility, Applicability, and Suitability.

Each Engagement Policy corresponds to a Filter decision component in the NBA_Sales_CreditCards strategy, and each of these components uses a Proposition Filter rule to implement the rules created in NBA Designer.

A Proposition Filter rule contains the conditions that make certain customers eligible for certain offers.



The decision strategy that implements the new eligibility requirements will be used in this Filter component.

The purpose of this demo is to show the mechanics and required steps for creating an Engagement Strategy, rather than focusing on a concrete use case.

To create a new Engagement Strategy, navigate to the Intelligence -> Strategies and create a strategy with a new canvas.

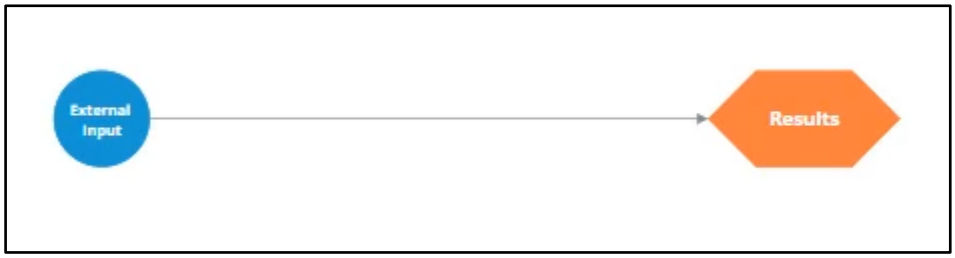
Enter a short description for the new strategy.

Select the business Issue and Group, and ‘Apply to’ class. Note that the ‘Apply to’ class is the Customer class from the Primary Context.

Now, open the strategy.

Enable the External Input component on the canvas to represent all Strategy Results (SR) records that are input into the strategy. The strategy will basically be used as a When rule in a Proposition Filter.

Connect the External Input component to the Results component.



Save and test the strategy with the Troy data transform.

For external input you can use the AllCreditCards strategy, as it imports all Sales->Credit Cards. The test shows that all credit card Actions will reach the Eligibility Strategy, nothing is filtered out by the framework.

Notice that the strategy outputs various credit card offers for Troy.

Complex eligibility rules can be implemented in this decision strategy.

For now, to keep it simple and show the mechanics of the Engagement Strategy, consider that customers qualify only for the Standard Card. Everything else is filtered out.

To implement this, add a Filter component to filter out all credit cards except the Standard Card.

Now, configure the Filter component.

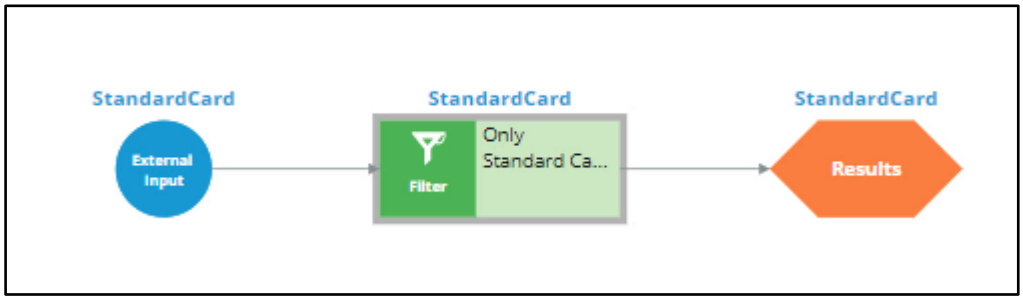
Start by naming the Filter component.

Click the gear icon to open the Expression Editor.

The Filter component should select the Action for which the pyName property is equal to “StandardCard”.

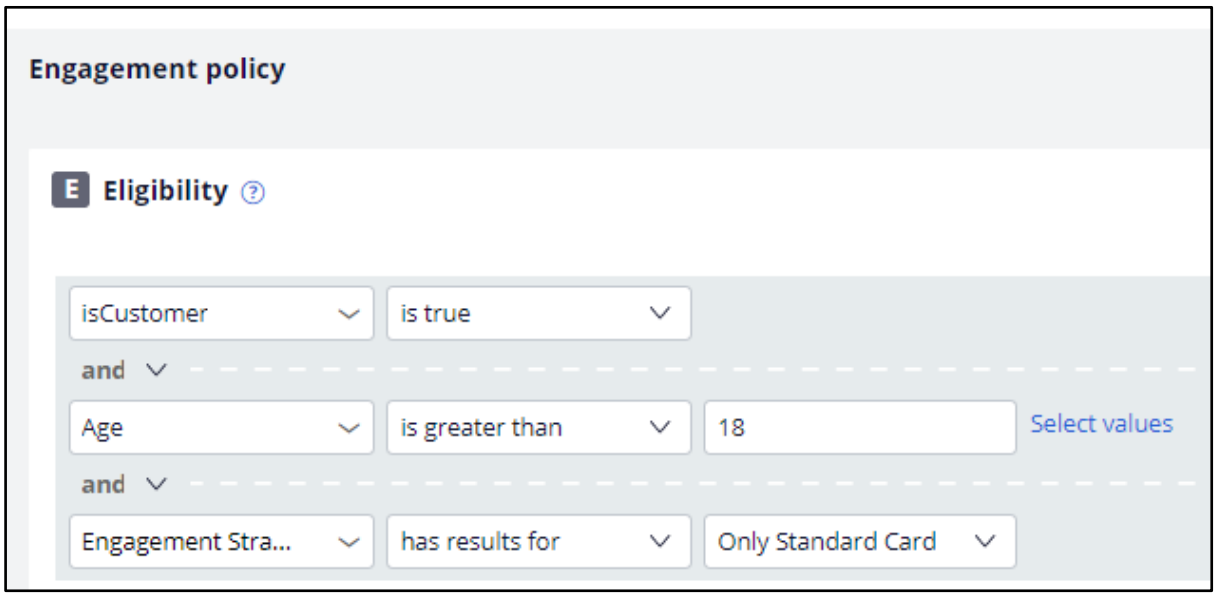
Connect the components on the canvas and re-test the strategy.

Examine the output of the Filter component. Now the Filter component outputs only the Standard Card. All other Actions are filtered out.



Make this strategy accessible in Next-Best-Action Designer as an Engagement Strategy.

Once the Record is added, navigate to NBA Designer and open the Engagement Policy to configure this strategy as an Eligibility Strategy for the Group-level Eligibility rules.



The condition is: Engagement Strategy has results for Only Standard Card.

Saving this completes the required configurations.

Back on the U+ bank website, log in as Troy to see that the Standard Card offer is displayed. Everything else is filtered out by the Engagement Strategy you just created.

This demo has concluded. What did it show you?

- How to build a decision strategy that can be used as an Engagement Policy rule.

- How to define Eligibility rules using a decision strategy in Next-Best-Action Designer.

Using a scorecard in a decision strategy

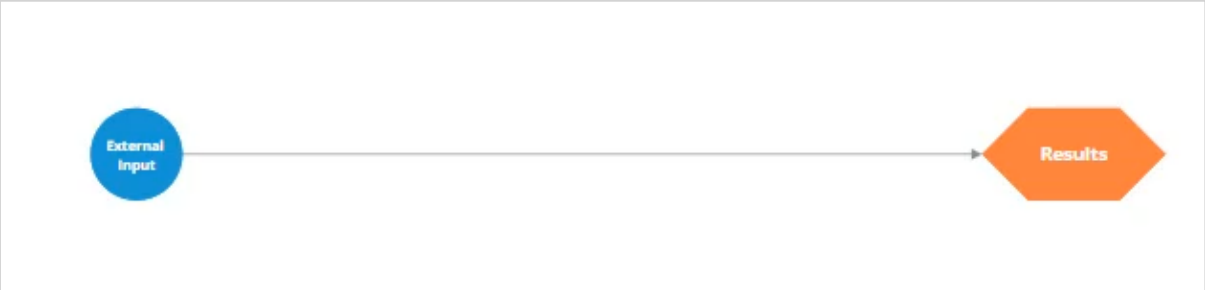
Learn how to use the segmentation result and the score value from a scorecard in a decision strategy. Learn how suitability rules can be defined to reflect the bank's requirements using a decision strategy. Also, learn the basics of strategy optimization.

Transcript

Currently, U+ Bank is cross-selling 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 with a credit score greater than or equal to 600.

To implement this requirement, use an engagement policy that includes a decision strategy.



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

Select the *DetermineCreditScore* scorecard model, which U+ Bank has already created**.**

If you open this scorecard, you can see how the system computes the credit score. Note that the three customer properties are in use, and a score is assigned to the value or range of values that the property can have.

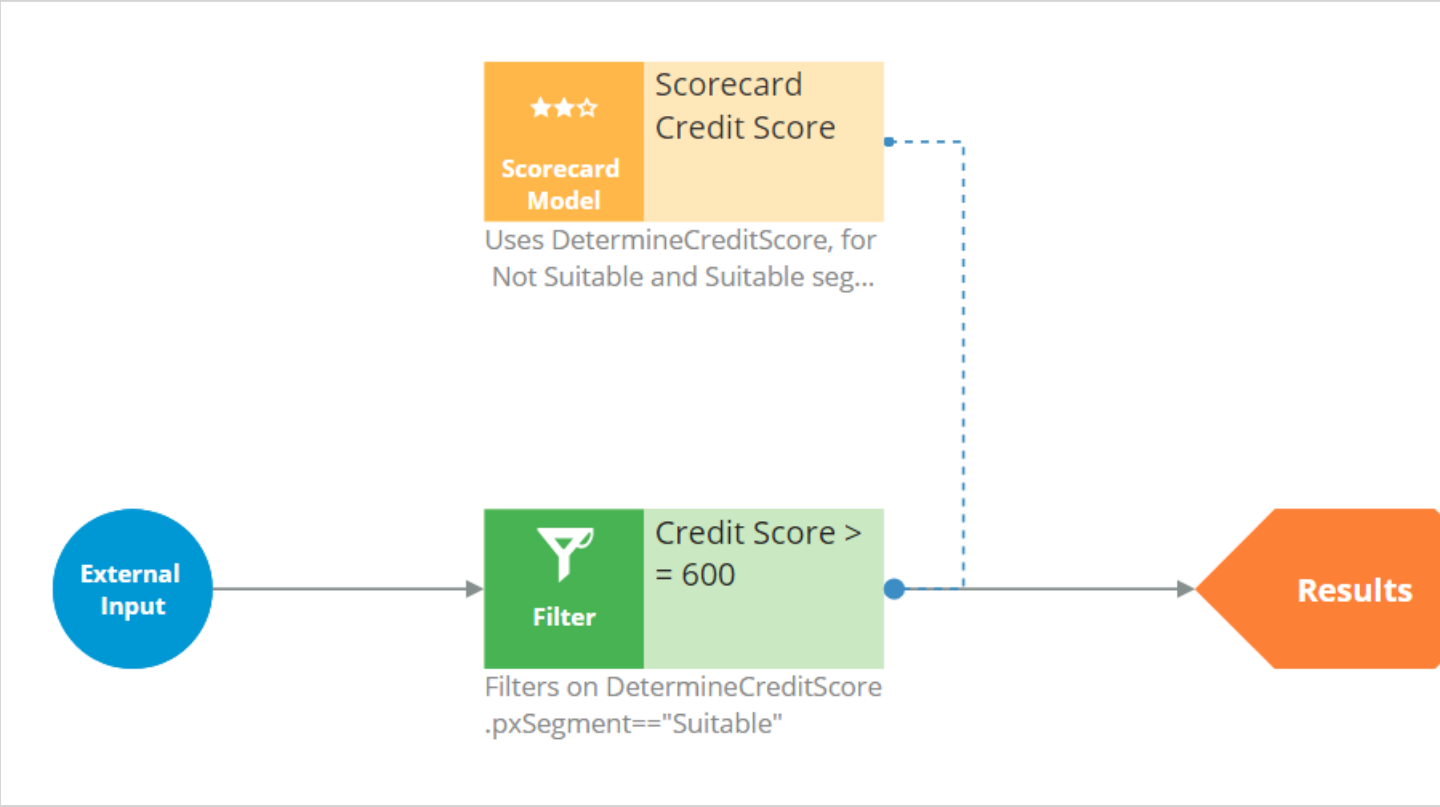
You can also see the segment results. The scorecard returns *Not Suitable* result if the credit score is less than 600. Otherwise, it outputs a *Suitable* result.

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 system stores the result of the scorecard 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.



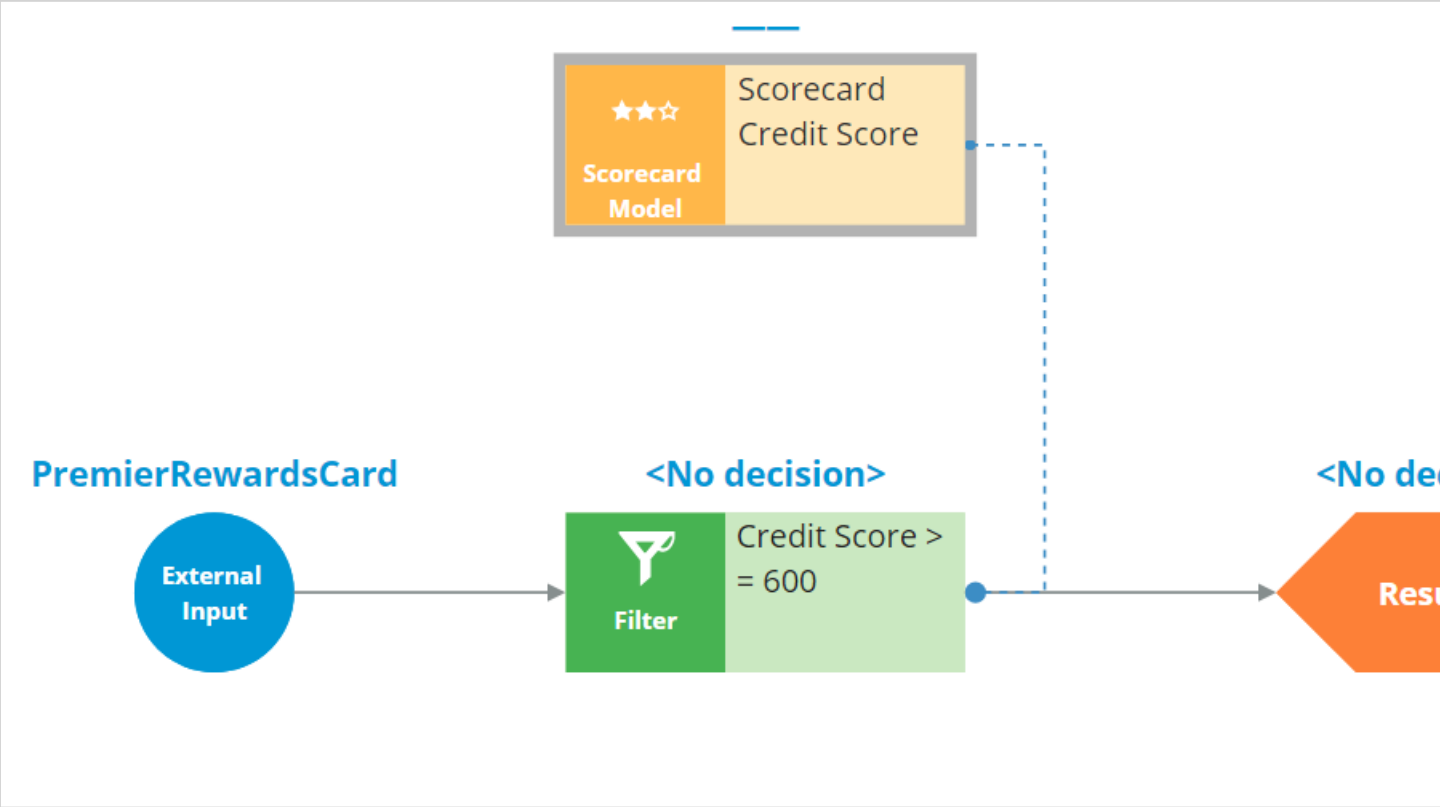
By using the External Inputs component, you can also create more complex conditions, where Action attributes are also in use.

Save the configurations.

Now, test the strategy by using Data Transform for Troy and Robert.

For external inputs, consider all credit cards available.

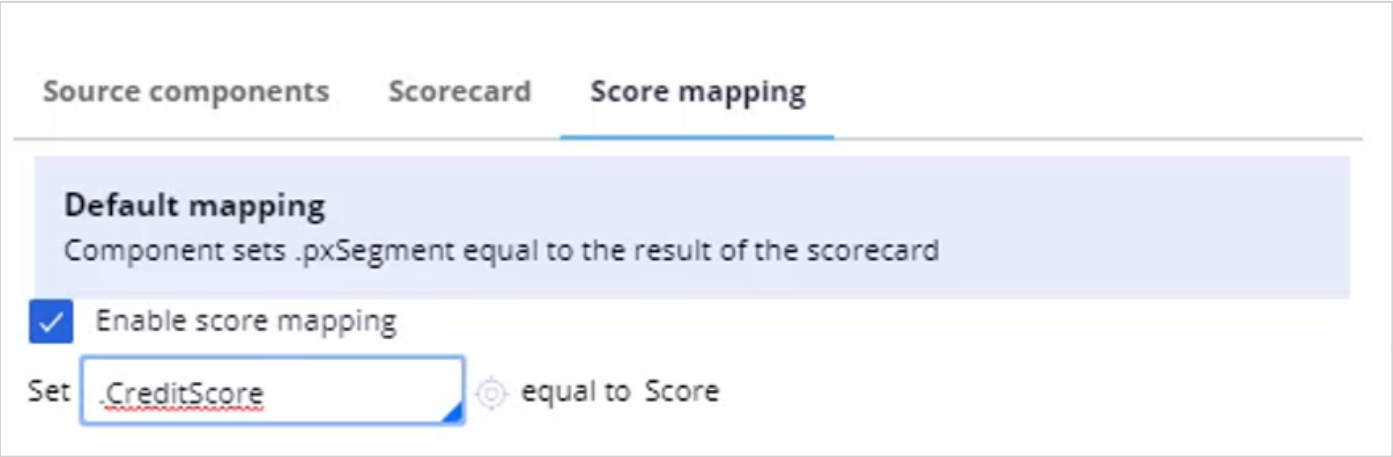
The result of the scorecard is *Not Suitable*. Therefore, the strategy did not output any results for Troy.



Now, repeat the test to verify the results for the Robert Data Transform. The strategy is also not producing any results for Robert, as the segment value is *Not Suitable*.

Now, assume the credit score value for customers is already computed and available in the *Customer* Data Model. However, this value is not set for certain customers, so you want to use the credit score determined by the scorecard itself. To implement this requirement, you must have both scores available and use a switch component to select the right credit score.

To make these adjustments, open the Scorecard component and then map the score computed by the scorecard to the *CreditScore* property. This component now outputs the scorecard calculation in the *CreditScore* property.

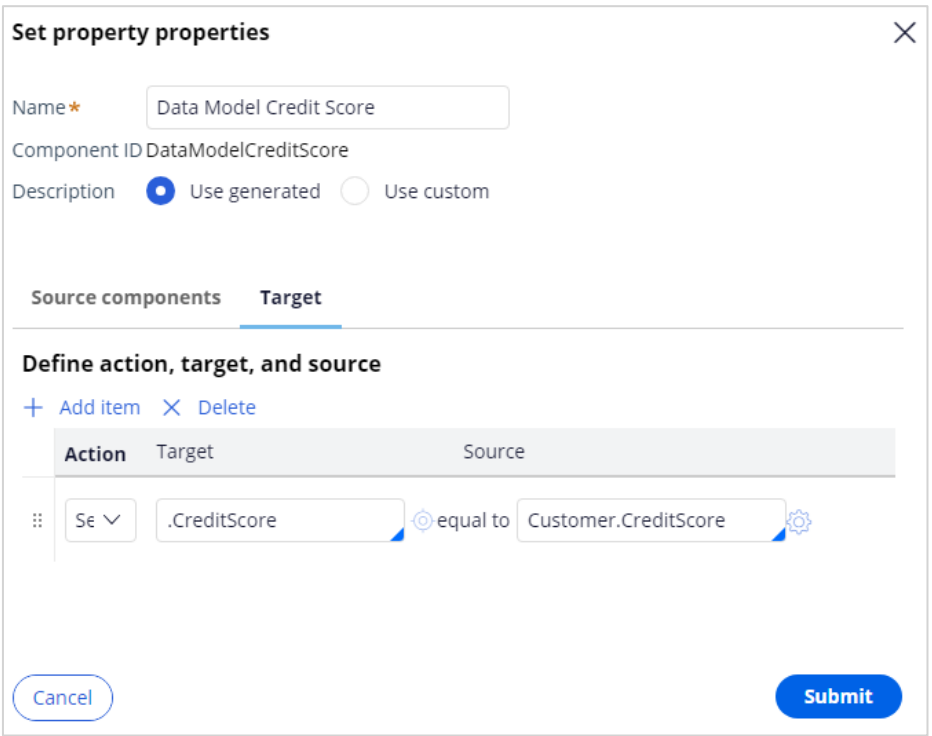


Then, add a **Set Property** component, given that the credit score is already available for certain customers.

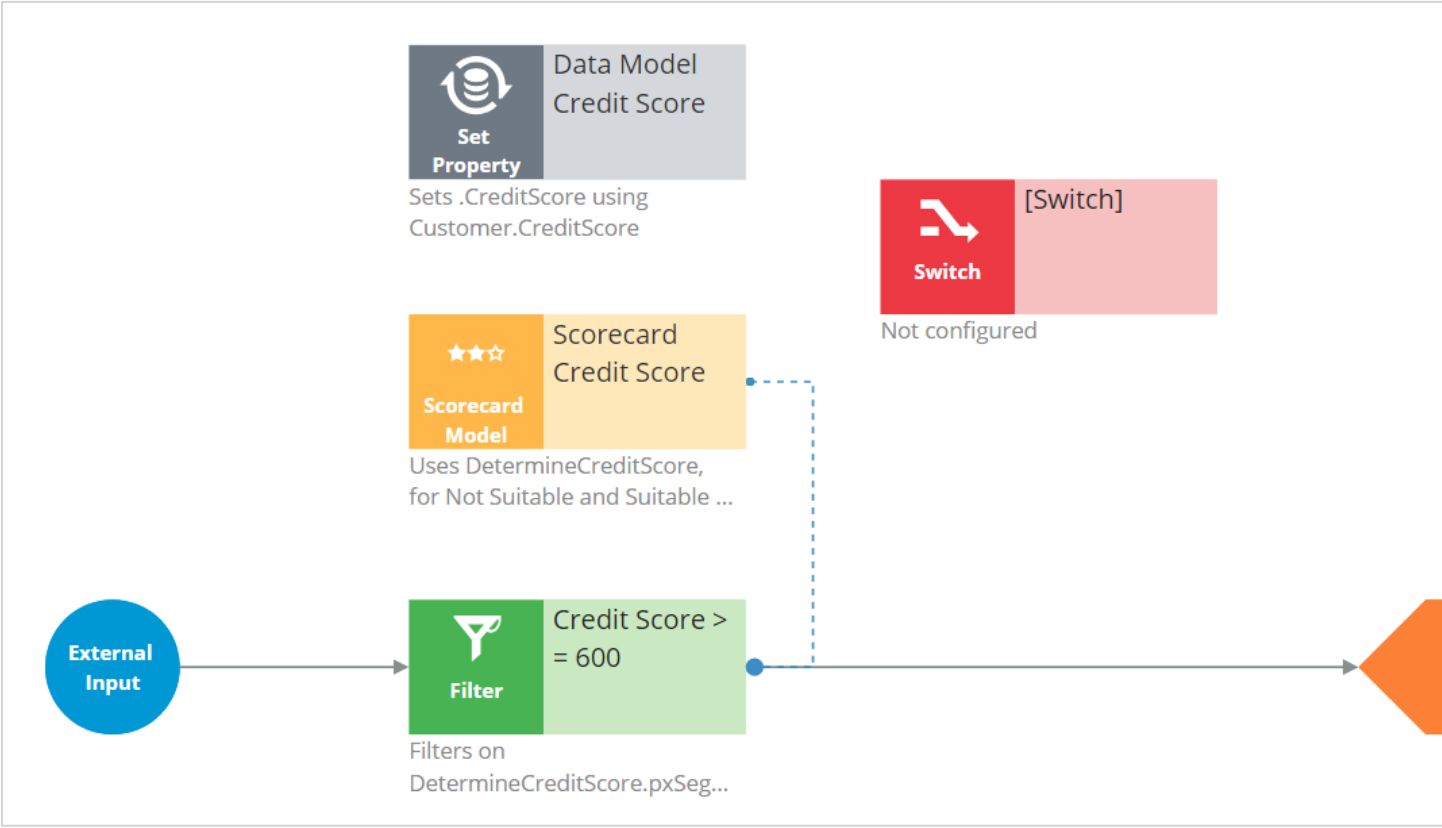
Use **Add item** to set the same *CreditScore* property to the available credit score value from the *Customer* Data Model.

Define the expression as *Customer.CreditScore.*

If set, this expression results in the credit score from the *Customer* Data Model.



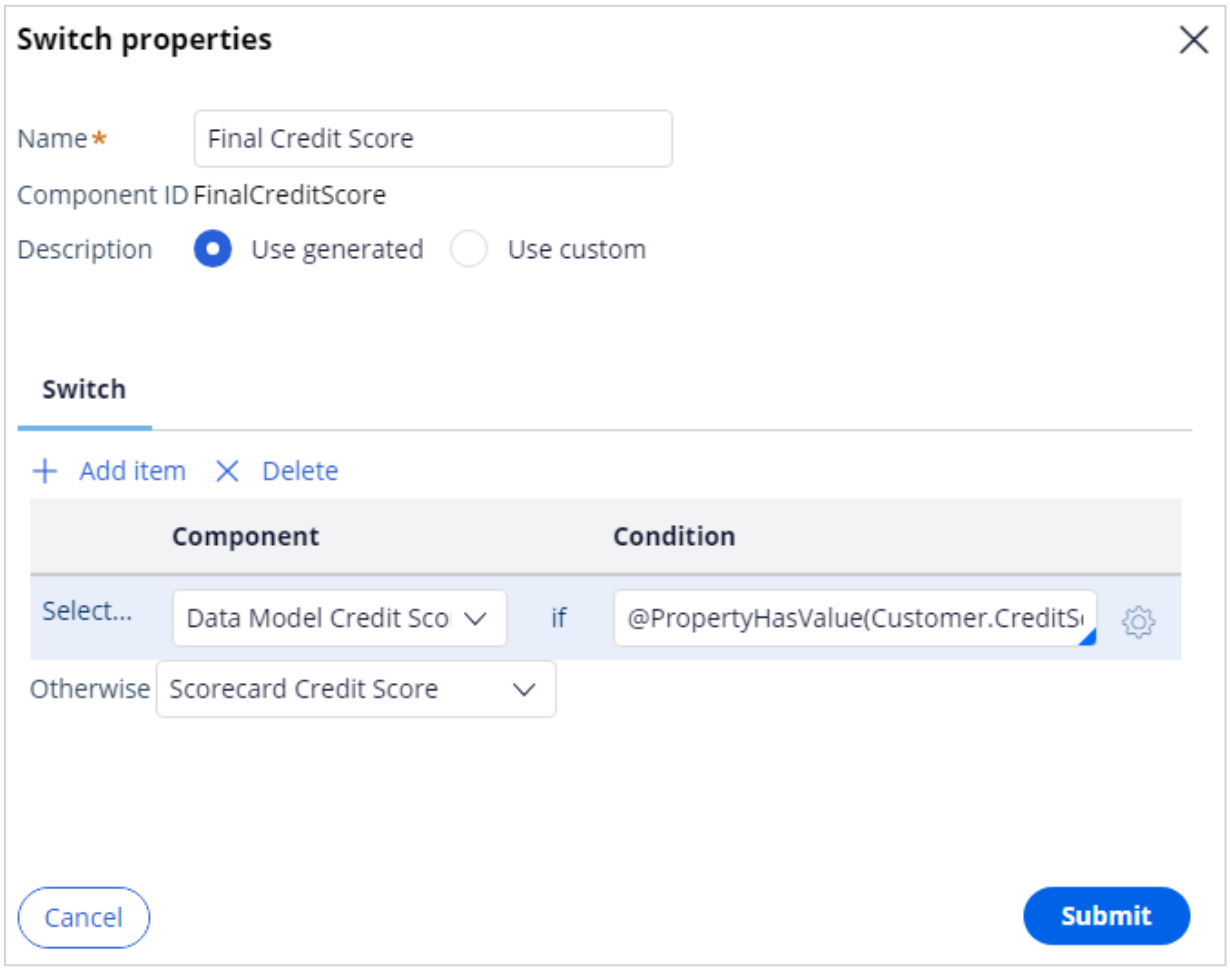
To select between the two credit scores, add a Switch component.



Set the name of the Switch component to *Final Credit Score* and configure the condition. To build the condition, use the Expression Builder. The condition reads:

Select Data Model Credit Score if @PropertyHasValue(Customer.CreditScore). Otherwise select Scorecard Credit Score.

@PropertyHasValue is a function that checks if a property has a value. In this case, it checks whether the *CreditScore* property of a customer has a value in the *Customer* Data Model.



With this configuration, the Switch component selects the *CreditScore* Data Model only if the *CreditScore* property of a customer has a value. If the *CreditScore* property is empty or does not have a value, the Switch component continues to run the scorecard to calculate the CreditScore.

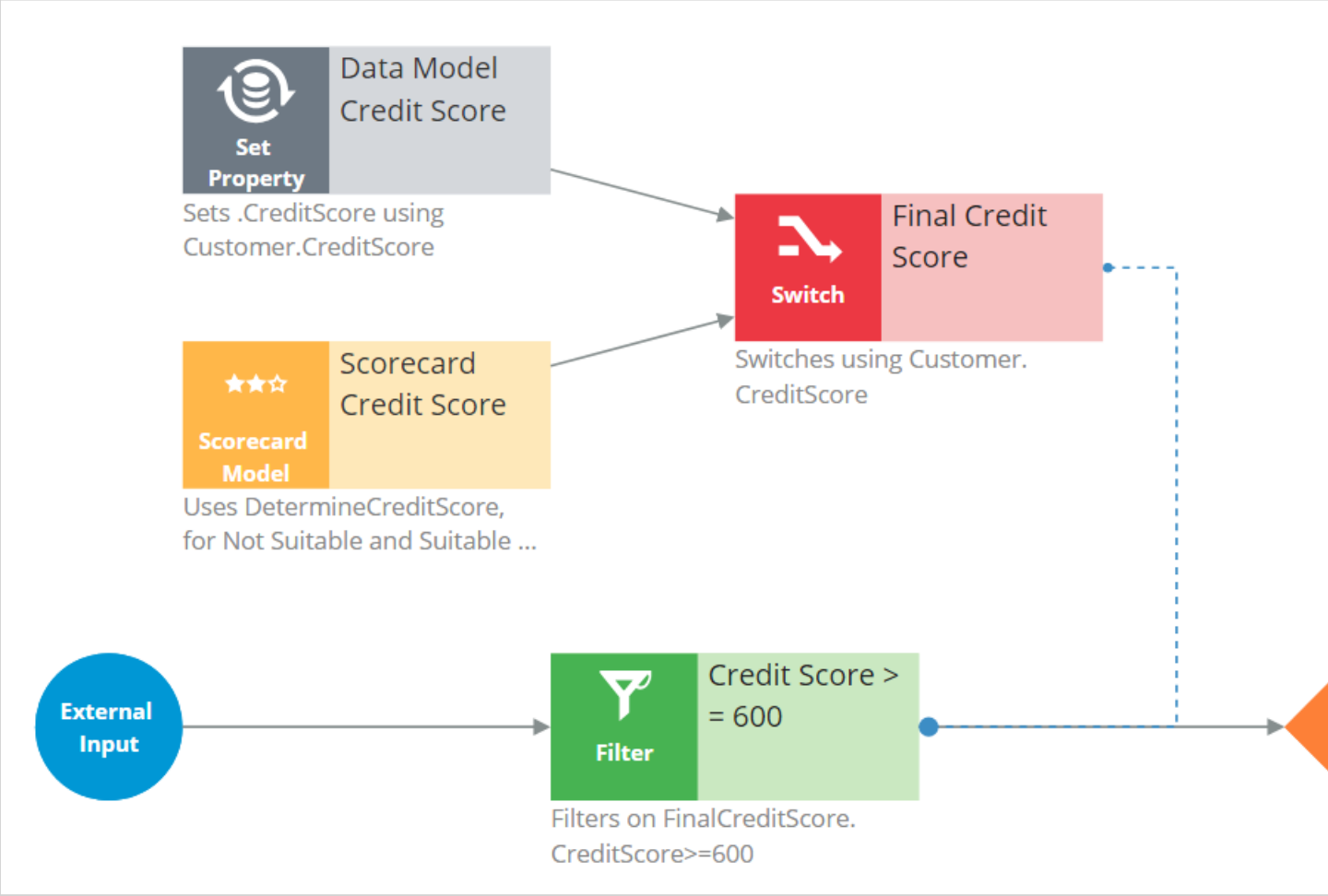
This configuration ensures that the Switch component optimizes the strategy by running the scorecard only for customers without a credit score in the external customer data. It effectively avoids redundant calculations and improves the efficiency of the decision strategy.

The arrows from Set Property and Scorecard Model components point to the Switch component. The system adds them automatically to ensure that the CreditScore values determined by these components are copied to the Switch component.

After you configure the Switch component, open the Filter component properties to modify the Filter condition. In this scenario, the bank decides to present various credit cards to suitable customers who have credit scores greater than or equal to 600.

So, open the Expression Builder to modify the condition as *FinalCreditScore.CreditScore>=600*.

Notice that the Filter component now references data from the Switch component.



Save the configurations.

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

Note that the strategy outputs results for Troy because his credit score, 625, is greater than 600.

If you open the Troy Data Transform, note that the CreditScore in the Data Model is 625.

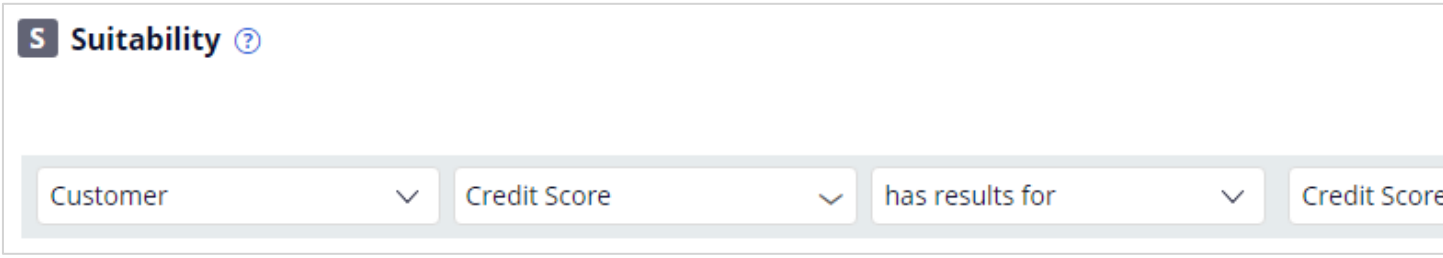
The Switch component picks up the credit score value available in the Data Model (that is, CreditScore = 625) and prevents unnecessary calculation by the scorecard.

The strategy also outputs results for Robert because his credit score is 675. The Switch component also picks up the credit score value available in the Data Model for Robert.

Because you configured the strategy, check it in so that the strategy becomes available on the U+ Bank website.

Now, mark the strategy as a relevant record and associate it with the Suitability category to make it available in the Next-Best-Action Designer engagement policy configuration.

You can now configure this strategy in the Next-Best-Action Designer engagement policy as a suitability condition for the group-level Suitability Rules.



Select the strategy. The condition is *Credit Score has results for the Credit Score >=600.*

Save to complete all the required configurations.

On the U+ Bank website, if you log in as Troy, notice that a credit card offer is displayed. The offer is shown because credit cards are suitable for Troy.

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

You have reached the end of this video. 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 whether a *Customer* property contains a value.
- How to optimize the decision strategy by using a Switch component.

Using 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

With the decision management capability of Pega Platform™, you can enhance applications to help optimize business processes, predict customer behavior, analyze natural language, and make informed decisions to better meet customers' needs and to achieve positive business outcomes.

Transcript

This video introduces you to Pega AI, a feature of the decision management capability of Pega Platform™.

Other decisioning features of the Pega Platform include:

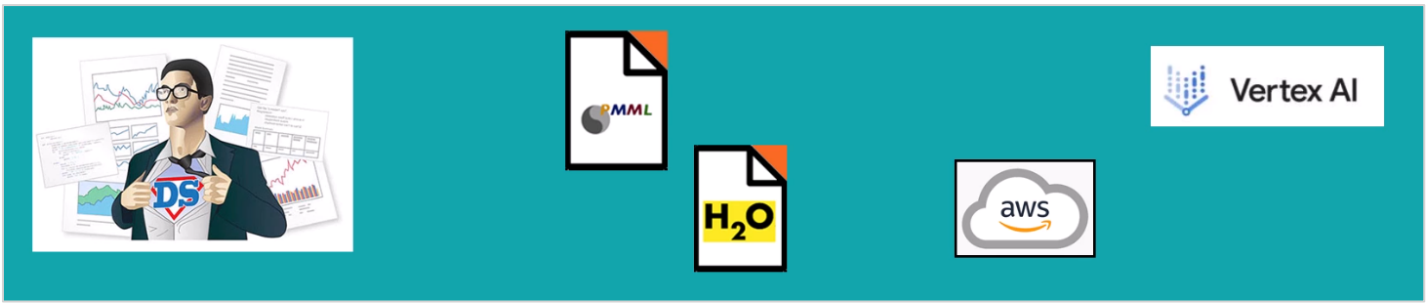
- Decision strategies to improve customer experience and deploy intelligent processes based on behavioral and operational data and data sets to read and write the data used in the decision strategies.
- You can use event strategies to detect patterns in data streams and react to them.
- And to ingest, process, and move data from one or more sources to one or more destinations, you can configure data flows as scalable and resilient data pipelines.

Decision management uses Pega AI to make predictions about customer behavior, successful case completion, the topic of an incoming message, or other subjects to make the decisions more relevant.

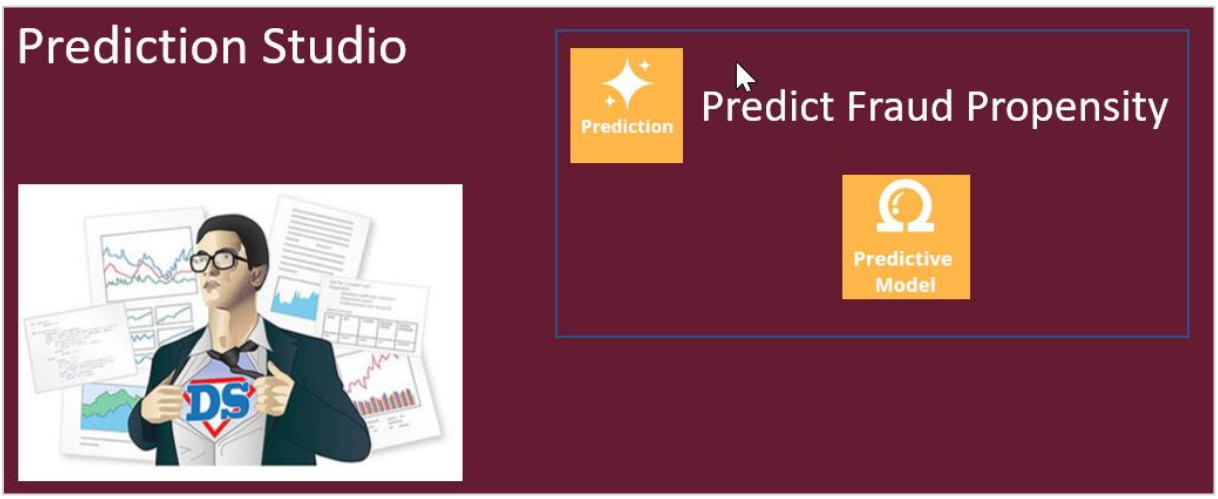
Decision management is a Pega Platform capability. You can apply decision management to any application that is built on Pega Platform.

Predictions differ to suit the domain they are used in, but one or more predictive models drive them all.

A data scientist can create a predictive model in Pega Platform or an external environment that can export the model as a PMML or H2O file. Another option is to connect to a machine learning service such as AWS SageMaker or Google Vertex AI.



If an insurance company wants to use Pega Process AI™ to route incoming claims that might be fraudulent to an expert based on the outcome of a predictive model the data scientist creates a fraud model to drive a new case management prediction in Prediction Studio.



Prediction Studio is the dedicated workspace where you manage the life cycle of predictive models and the predictions they drive.

A prediction is a hand-off to an application developer, who can then use the prediction in a decision step in the case type to route cases more accurately. This strengthens the separation of concerns.

You can use Pega Customer Decision Hub™ to make next-best-action decisions for your customers.

Customer Decision Hub predictions can predict customer behavior, such as which customer is about to churn or predict the likelihood that a customer clicks on a web banner to support the decision on which banner to show to a customer.

Pega Adaptive Decision Manager (ADM), a key component of the decision management capability allows a data scientist to configure self-learning, adaptive models that continuously improve predictions about business processes and customer behavior.

An adaptive model rule typically represents many adaptive model instances because each unique combination of the model context generates a model.

In Customer Decision Hub, adaptive models drive many predictions that come with the product out of the box, such as the Predict Web Propensity prediction that predicts the likelihood that a customer clicks a web banner.

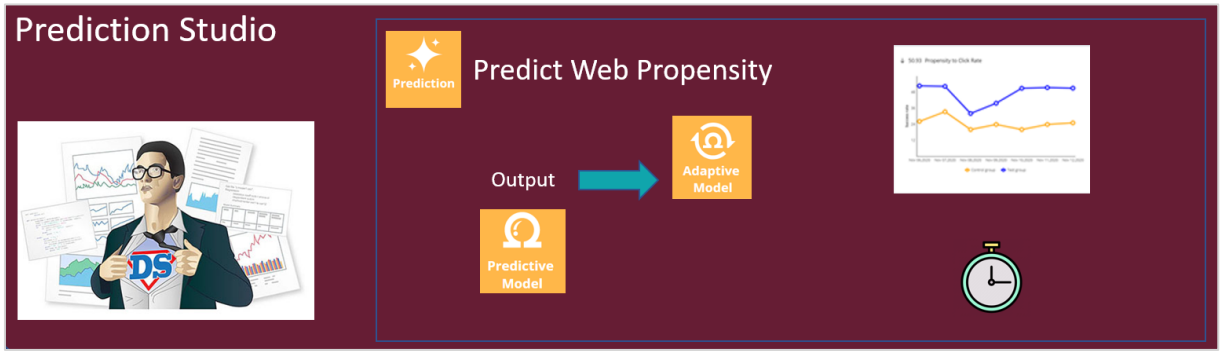
Customer Decision Hub predictions have several features specific for the domain such as a control group for which the prediction outputs a random propensity instead of the propensity that is generated by the adaptive model instance.

Comparison of the control group and the model propensity-based group allows you to measure the lift in a success rate that the AI generates, an important business metric.

Also, Customer Decision Hub predictions feature a response timeout setting. After the timeout expires, a negative response is recorded.

The response timeout setting depends on the use case. For example, in a web use case, several minutes suffice while in an outbound email campaign, the response timeout is set to several days to allow customers enough time to respond.

You can further enhance the prediction by using the output of a predictive model as a predictor in the adaptive model.



The Pega Customer Service™ application uses the natural language processing capability of decision management to analyze incoming text and route the messages based on the topics and entities detected.

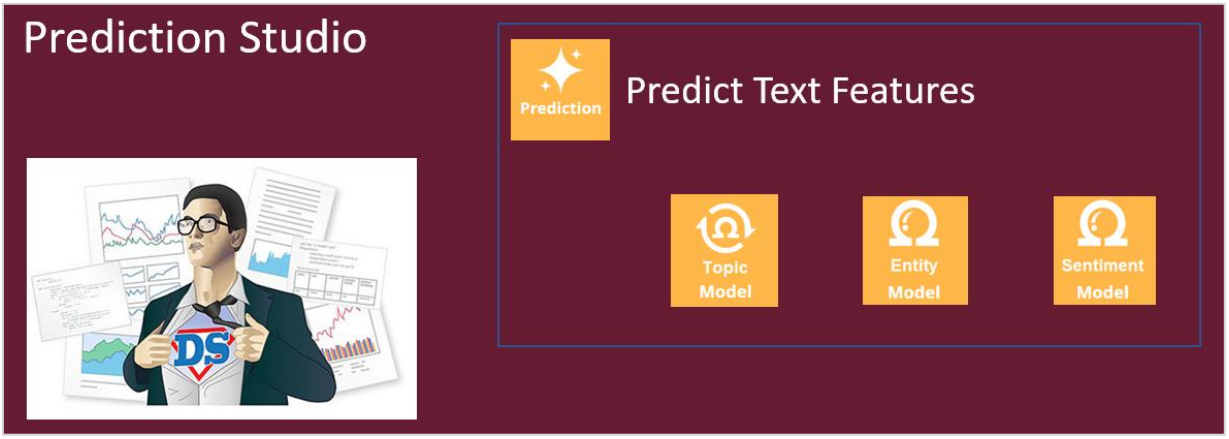
Pega Customer Service uses Text analytics predictions that are distinctly different from both case management predictions and Customer Decision Hub predictions.

Text analytics predictions use predictive models to detect the topic of an incoming message that the application can use to optimize the routing of the message to the relevant department.

Secondly, text analytics predictions use entity extraction models that qualify text as, for example, an account number, a postal ZIP code, or an address.

The application can use this information to fill relevant fields in a case automatically.

Finally, the text analytics predictions come with a sentiment model that can route or prioritize negative messages to improve the customer experience.



Feedback on the detected topics, entities, and sentiment by CSRs improves the performance of the text analytics prediction over time.

This video has concluded. What did it show you?

- Pega AI allows you to improve business processes and customer engagement by using predictions.
- Predictive models drive the predictions.
- The predictive models can be static or adaptive.
- Predictions are managed in Prediction Studio.

Arbitrating across business issues

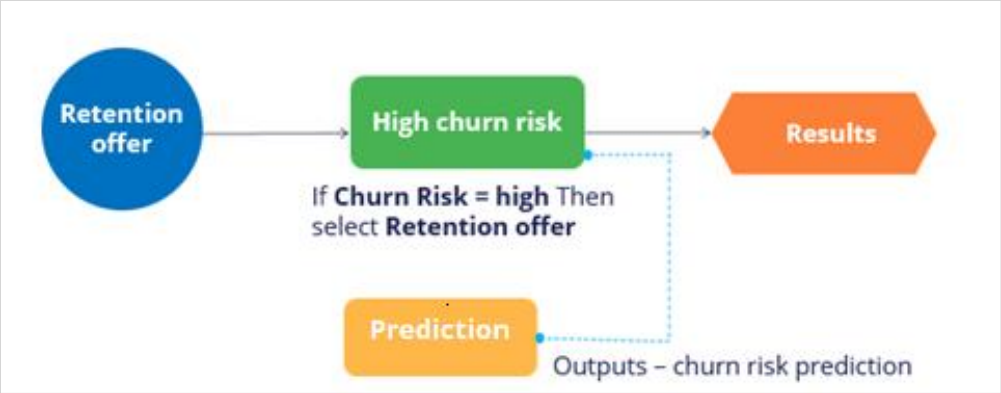
Pega Customer Decision Hub™ combines analytics, business rules and customer data to make intelligent decisions. Every next best action weighs customer needs and business objectives to optimize decisions.

To arbitrate across multiple business issues, you create a decision strategy and use 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.

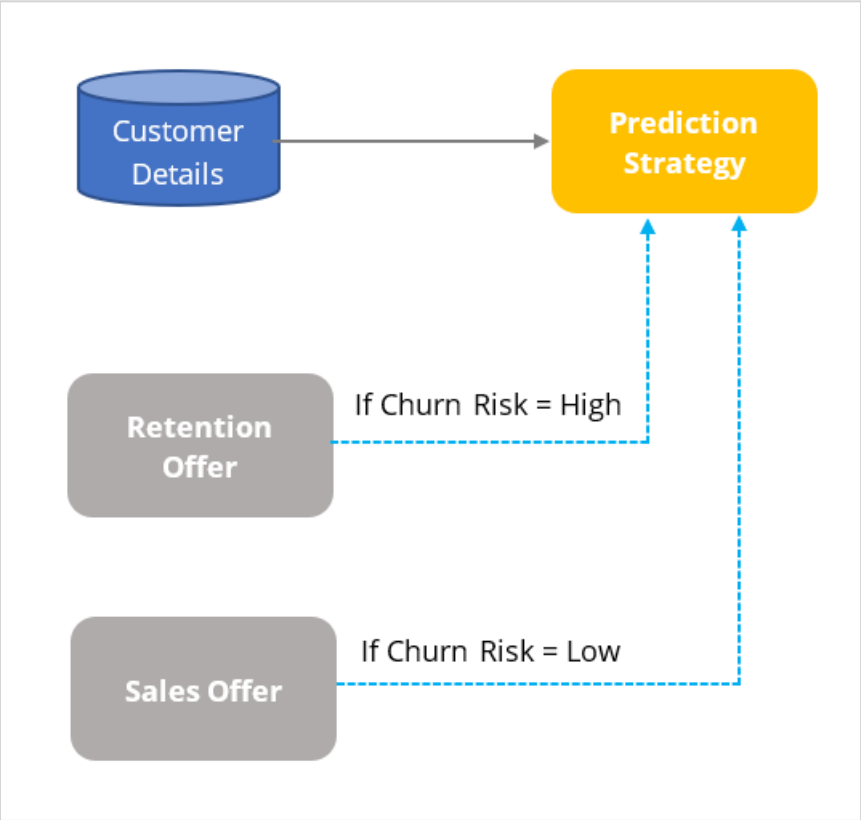
You create a decision strategy that differentiates between high risk and low risk customers that references the churn prediction.



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

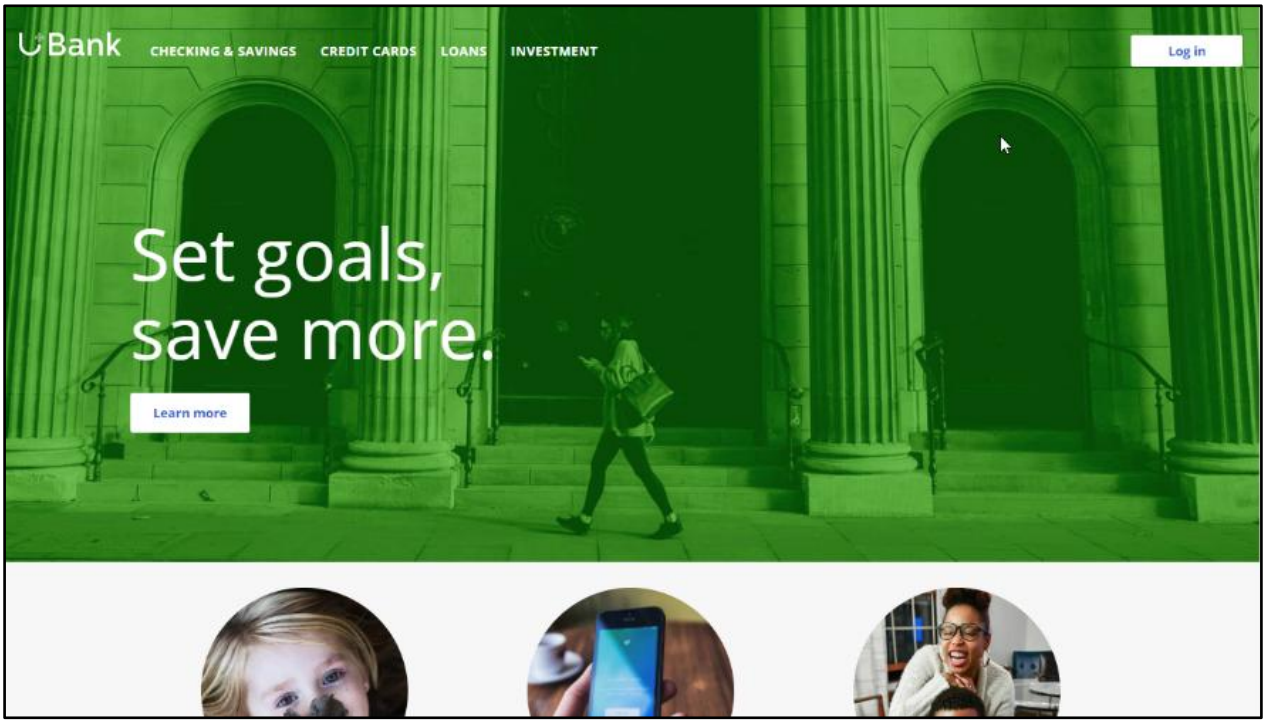
Introduction

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.

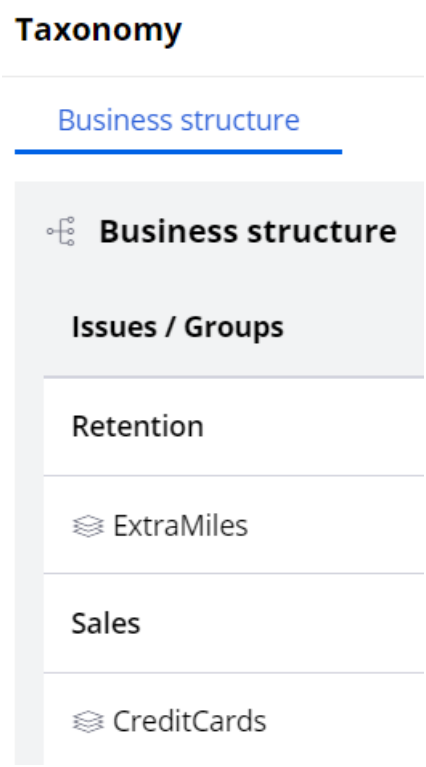
Transcript

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

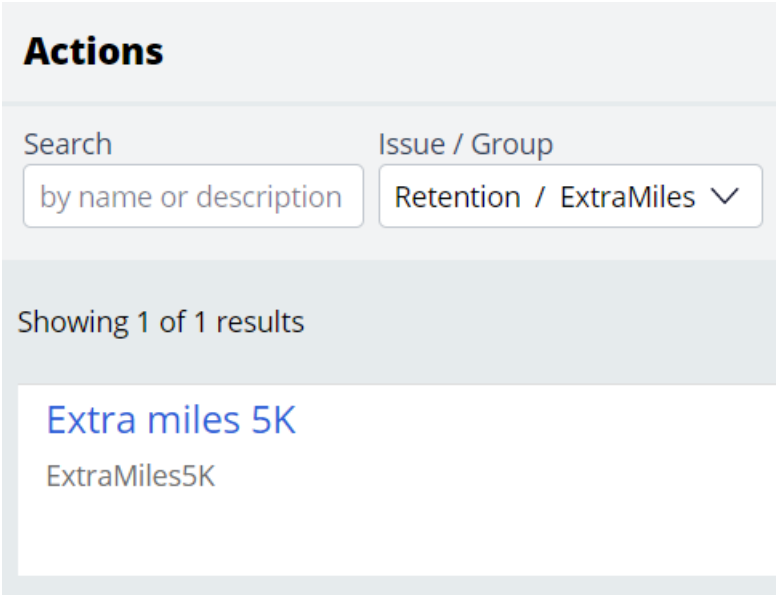
Currently, U+ Bank is cross-selling on the web by showing various credit cards to eligible customers who log in to its website. The bank now wants to show a retention offer, instead of a credit card offer, to customers who are likely to churn in the near future. The credit card offers are shown only 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.

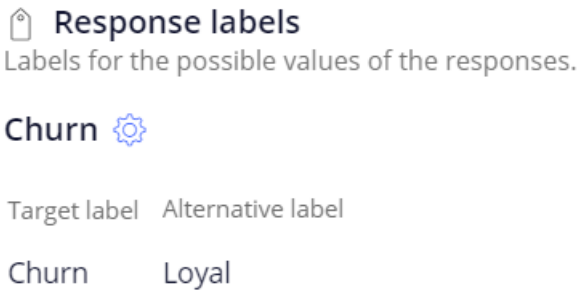


This ExtraMiles group contains a retention offer, Extra Miles 5K.

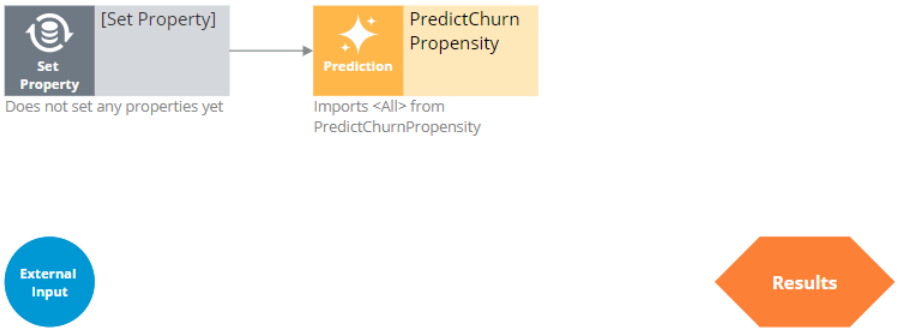


The next step is to create an applicability condition that makes a customer qualify for a retention offer when there is a high likelihood that the customer might churn. A data scientist has created a prediction that identifies these high-risk customers. When you open the prediction in Prediction Studio, notice that the possible response labels are **Churn** and

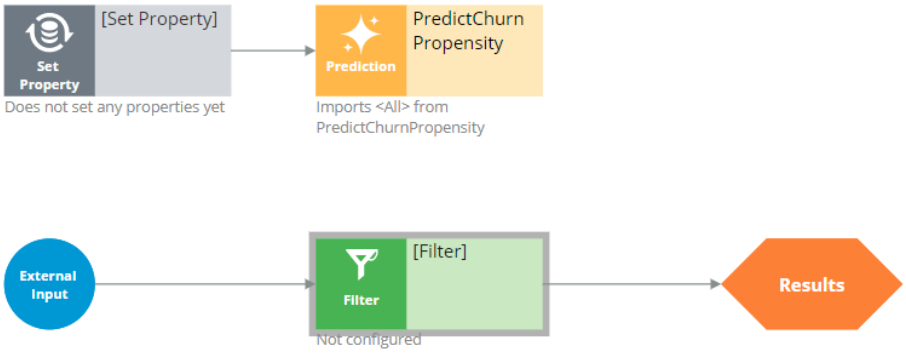
Loyal to predict customer behavior. The result of the prediction is stored in the pxSegment property.



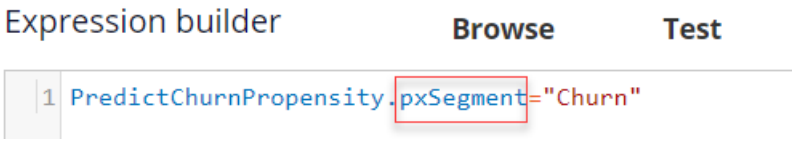
To define the applicability condition, you create a decision strategy to output a retention offer only if the response label of the prediction is **Churn**. Add a **Prediction** component to the canvas and configure it to reference the churn prediction. Add a **Set Property** component and connect it to the Prediction component. You can configure the Set Property component at a later point to accommodate parameterized fields.

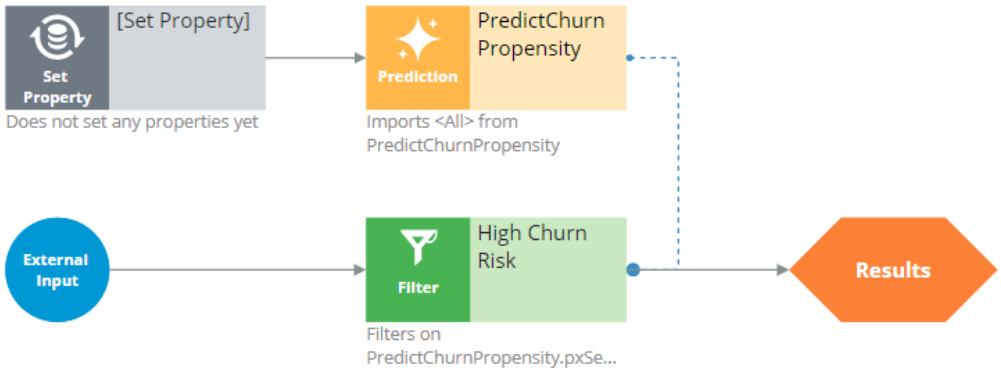


Next, add a filter component to filter out the loyal customers and pass retention offers to high churn risk customers only.

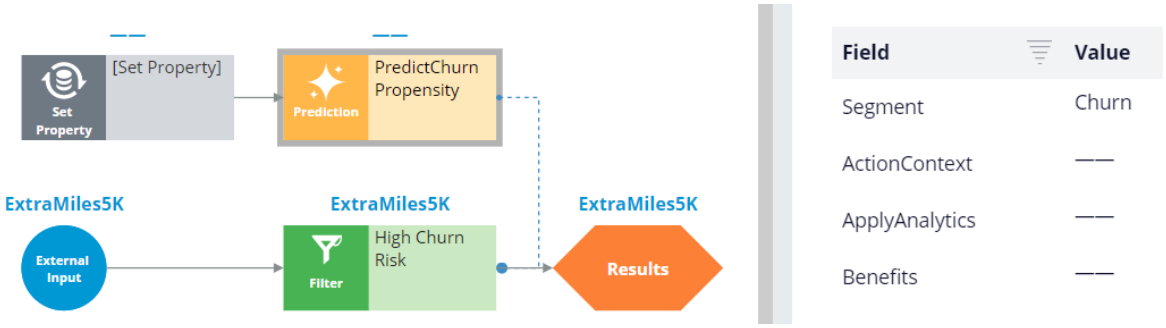


The filter condition is defined to output a retention offer when the pxSegment property of the prediction is equal to **Churn**.

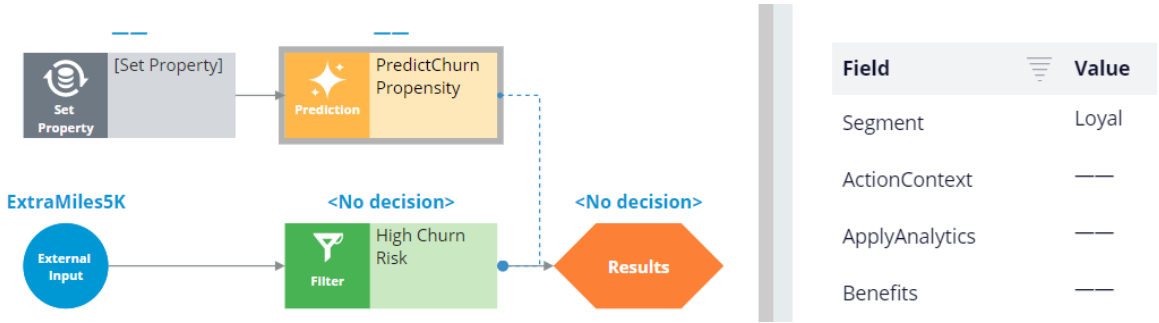




Next, test the strategy using two customer profiles, **Troy** and **Barbara**. For external inputs, consider all available retention offers. The strategy outputs a result for **Troy** because the result of the prediction is **Churn**.



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



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.

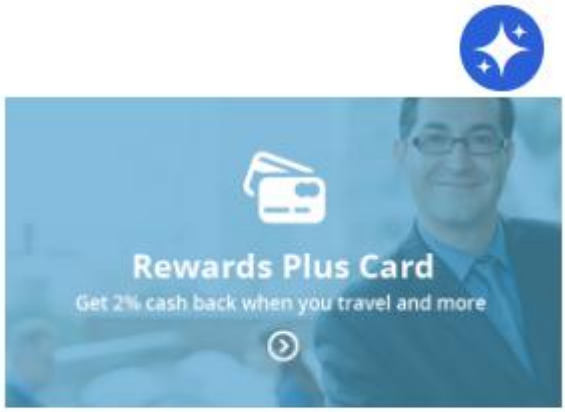
The first business rule you need to implement is that the **ExtraMiles** group is applicable only to high churn risk customers. To implement this rule, in the **Applicability** section, define a condition for the customer field. Select the **RetentionStrategy**. The condition is: the RetentionStrategy 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.

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

Pega Customer Decision Hub™ uses Adaptive Models that use a wide range of predictors to optimize customer interactions. You can make input fields that are not directly available in the customer Data Model accessible to the models by configuring these fields as parameterized predictors. Learn how to create parameterized predictors for Expressions, offline model scores, and online model scores.

Transcript

This demo shows you how to create parameterized predictors to make input fields that are not directly available in the customer Data Model accessible to Adaptive Models. We will cover three examples: Expressions, offline model scores, and online model scores.

U+ Bank is cross-selling its credit cards on the web by using Pega Customer Decision Hub™. All customer data, including financial clickstream summary attributes, such as webpage visits, is available to the Adaptive Models that determine which offer to display for a particular customer.

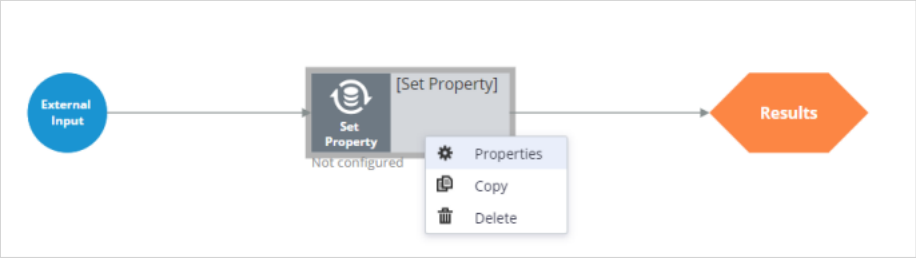
The first parametrized predictor is the ratio of two clickstream summary attributes that denote the number of visits to the U+ Bank Investment webpage in the last 30 days and the last 90 days. An increasing ratio might indicate growing customer interest.

The *Financial Services Clickstream* Data Set contains the clickstream data, which is a record of the user's activity on the website. These entries reflect the *Investment* webpage visits of the customer.

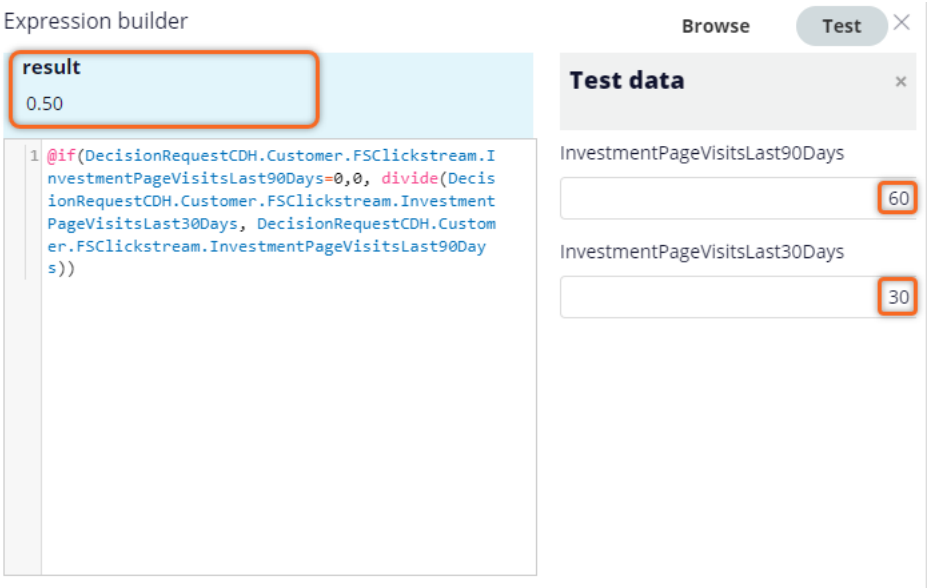
Investment page visits last 1 hour	Investment page visits last 30 days	Investment page visits last 7 days	Investment page visits last 90 days
0.0	0.0	0.0	0.0
2.0	2.0	2.0	2.0

To populate a parameterized predictor, you configure the *NBA Pre-Processing Extension Point* Decision strategy that is included in a change request. The pre-processing strategy runs before the Prediction strategy so that the output of the extension strategy is available as input to the Prediction.

The first example of a parameterized predictor is an Expression that references two clickstream fields. To add a field, you add a Set Property component to the strategy and create a new property in the *Strategy Results* class. Set the property type to **Decimal** to store the ratio of customer visits to a web page in the last 30 days and in the last 90 days.



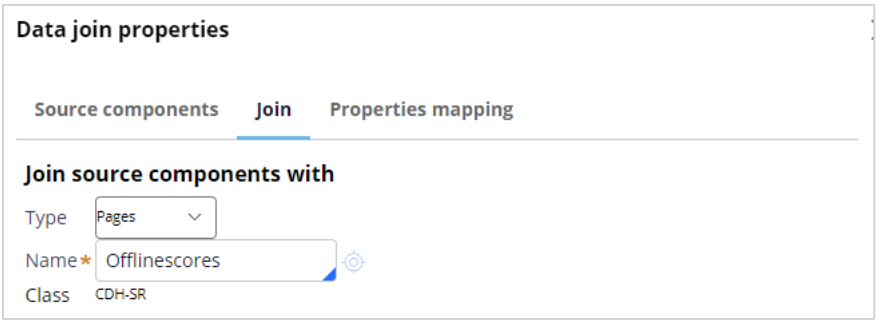
This Expression returns a value of zero when the number of Investment page visits in the last 90 days is zero. Otherwise, it returns the ratio of the Investment page visits in the last 30 days to the last 90 days. A high value might indicate the increasing interest of the customer in the content of this page.



The second example of a parameterized predictor is offline model scores. A Data Set that is associated with the customer context makes the data available as a data source. Each customer can have multiple scores for different categories or different channels.

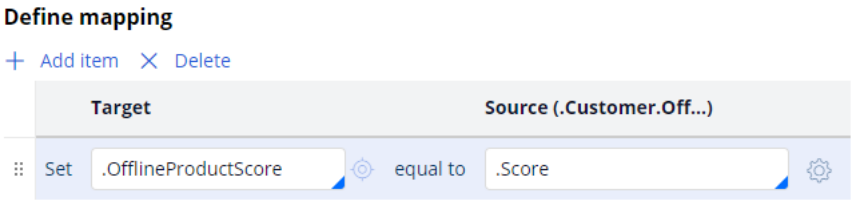
To add the model scores as potential predictors to the Adaptive Models, you add a Data Join component to the extension strategy.

The component joins the *Offlinescores* page that contains the model scores to the available data.



To output only relevant scores, you specify the channel and the product category to match the Prediction.

You then create a Decimal property and map it to accommodate the model scores.



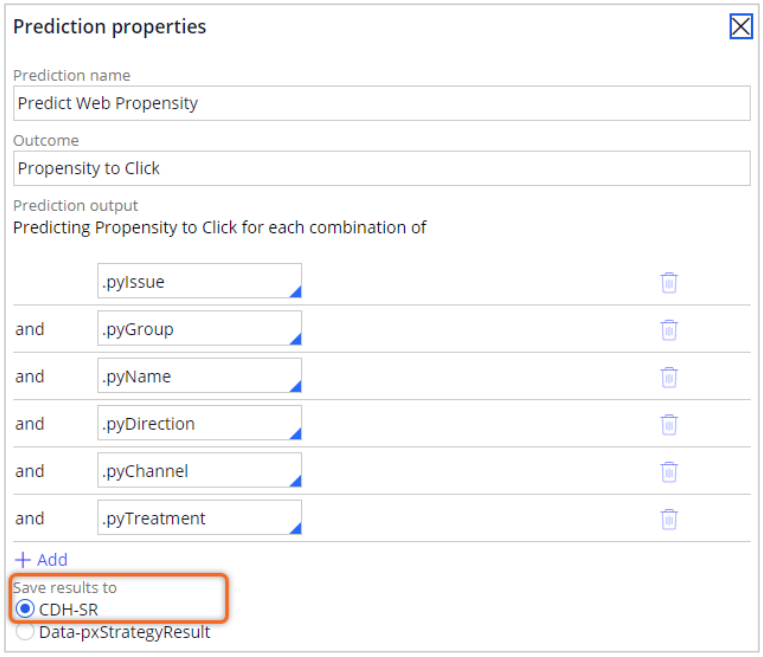
The third example of a parameterized predictor is the score of a churn Prediction that runs in Customer Decision Hub. The Data Scientist team develops Predictive Models based on the latest insights and data that drive the churn Prediction.

To make the churn score available to the Adaptive Models as a parameterized predictor, add a Prediction component to the extension strategy. Reference the churn Prediction in the *Customer* class.

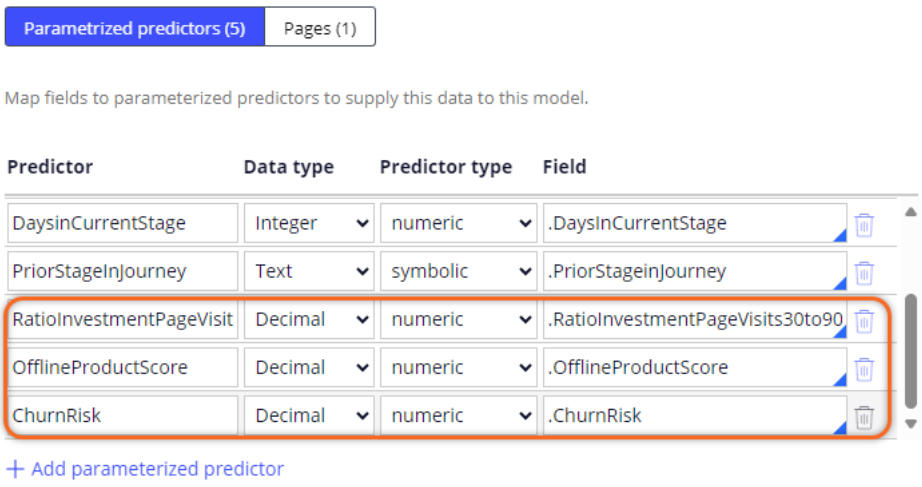
Add a Set Property component to the extension strategy to create a new property and map it to the propensity calculated by the churn Prediction.

Run the strategy by using a test customer profile to verify that the strategy outputs include the real-time churn risk score, the offline product scores, and the Expression.

Next, you use Prediction Studio to map three new parameterized predictors in the Prediction. The Predict Web Propensity Prediction calculates the likelihood that a customer clicks on a web banner. Like the new properties, the system saves the Prediction results to the *CDH-SR* class.



You then create three new parameterized predictors and map them to the new properties that represent customer behavior, offline scores, and online scores.



When you submit your work for deployment, the application saves all changes made in Predictions to a *Change prediction* change request.

After a review by the Team Lead, a Revision Manager deploys the changes as part of a revision.

You have reached the end of this video. You have learned:

- How to configure the pre-processing extension strategy for parameterized predictors.
- How to add properties as parameterized predictors to the Prediction.
- How to submit the changes for deployment.