AdventureWorks SkiResort

Demonstration

Contents

[OVERVIEW 3](#_Toc474498536)

[SKIRESORT TOUR (Xamarin) 4](#_Toc474498537)

[SKIRESORT TOUR (Web) 9](#_Toc474498538)

[SKI RENTALS (Forecasting) 13](#_Toc474498539)

[LIFTS (Anomaly detection) 23](#_Toc474498540)

[DINING (Tailored experiences) 30](#_Toc474498541)

# OVERVIEW

Applications show intelligence when they can spot trends, react to unusual events, predict outcomes or recommend choices. Learn how to introduce intelligence traits into your apps including: establishing feedback loops, applying big data and machine learning techniques to classify, predict or otherwise analyze explicit and implicit signals, and operationalizing the full stack into the regular usage flow of the app.

Most every day apps, from consumer to enterprise can deliver greater customer or business benefit by learning from user behavior and other signals. In this session we’ll take a pragmatic look at introducing real, useful data-driven intelligence into apps by walking through services, code and data needed to make it happen.

*Disclaimer:*

*To deploy this demo and use it, we expect the user to have an intermediate knowledge of Microsoft Azure.*

# SKIRESORT TOUR (Xamarin)

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
| **TOUR** | |  |
|  | 1. Open the AdventureWorks.SkiResort solution. 2. Set the Andoid project as StartUp project. | If you think about adding intelligence to an app, let's use this ski app.  For those of us who don't ski, this app is pretty straightforward. At pretty much any resort you go to, you'll find an app like this.  **NOTE**: If you prefer you could ise the iOS project if you have a Mac. |
|  | 1. Choose the Microsoft Android Phone emulator. 2. Build the project. 3. Click on F5. | You can use your own Android device if it´s connected. |
|  |  |  |
|  |  | You can look up lift status.  You can check snow conditions.  You can look at dining options and stuff like that.  But let's just walk through three scenarios on an app that looks just like  this with a before and after. |
|  | 1. Click on the menu. |  |
|  | 1. Click on “Lift Status”. | Shows open/closed status.  **With intelligence**  Predicts wait time.  Gives warnings based on traffic.  Identifies anomalies. |
| 3 | 1. Click on “Rental Reservations”. | Displays products.  Handles transactions. |
|  | 1. Depending on the selected start day, the app shows this message. | **With intelligence**  Predicts demand to help customers know what to expect. |
| 7 8 | 1. Click on “Dining”. | Shares menu.  Takes reservations. |
|  |  | **With intelligence**  Knows where you want to eat next. |

# SKIRESORT TOUR (Web)

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
| **TOUR** | |  |
|  | 1. Open the AdventureWorks.SkiResort solution. 2. Set the web project as StartUp project. | If you think about adding intelligence to an app, let's use this ski app.  For those of us who don't ski, this app is pretty straightforward. At pretty much any resort you go to, you'll find an app like this. |
|  | 1. Click on F5 in the “IIS Express” button. |  |
|  |  | You can look up lift status.  You can check snow conditions.  You can look at dining options and stuff like that.  But let's just walk through three scenarios on an app that looks just like  this with a before and after. |
|  | 1. Click on “Lift Status”. | Shows open/closed status.  **With intelligence**  Predicts wait time.  Gives warnings based on traffic.  Identifies anomalies. |
|  | 1. Click on “Rental Reservations”. | Displays products.  Handles transactions. |
|  | 1. Depending on the selected start day, the app shows this message. | **With intelligence**  Predicts demand to help customers know what to expect. |
|  | 1. Click on “Dining”. | Shares menu.  Takes reservations. |
|  |  | **With intelligence**  Knows where you want to eat next. |

# SKI RENTALS (Forecasting)

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
| **SKI RENTALS** | |  |
|  | 1. Open the AdventureWorks.SkiResort solution. 2. Set the web project as StartUp project. 3. Click on F5 or the “IIS Express” button. 4. Click on “Rentals”. 5. Depending on the selected start day, the app shows this message. | What we want to achieve is to know how busy it is going to be tomorrow.  And what we know is all of our past history.  So the interesting thing about intelligent apps is that they're intelligent not because  of being particularly sophisticated, but because they learn  from past experience.  In this case, past experience is the record of all of our history of rentals.  We have all of our years worth  of rentals in a table somewhere.  This is our transactional system.  You can see there in step one, that we have the the complete rental history. |
|  | 1. Click on “File>New>File”. |  |
|  | 1. Create a new “SQL” script. |  |
|  | 1. Click on “Connect”. |  |
|  | 1. Connect using SQL Server Authentication.   Server: <YOUR\_SQL\_VM>  DatabaseName: AdventureWorks.SkiResort  Username: skiresort  Password: P2ssw0rd@1 | The data is in SQL server 2016. |
|  | 1. List the existing rentals in the database.   “Select top 10 \* from rental”   1. Click on Execute. | Let's take a look at  the rental stable real quick.  This shows rentals,  what types of skis, what shoe sizes there are, etc.  From the predictive model standpoint, I don't need this  level of detail.  What I want to know everyday, is how many rentals did you have in the past and then what happened? What was the day? Did it snow the day before? How was the weather?  An interesting thing about the newer version of SQL Server, particularly in 2016 is that a lot of work that we've been doing is around allowing you to do analytical work directly in your transactional store. |
|  | 1. Expand the “SkiResort.Infrastructure” project. 2. Open the SkiResortDataInitializer.cs file. |  |
|  | 1. Go to the “RentalFeatures” View definition. |  |
|  | 1. Go to the SQL script. 2. Check the View Content.   “Select top 10 \* from RentalFeatures”   1. Click on Execute. |  |
|  | 1. Expand the “SkiResort.Infrastructure” project. 2. Open the SkiResortDataInitializer.cs file. 3. Go to the “INDEX” definition. |  |
|  | 1. Go to the SQL script. 2. Click on “include actual execution plan”. | Allows me to do a show plan. |
|  | 1. Select the “RentalFeatures” query. 2. Click on Execute. |  |
|  | 1. Go to the Execution plan window and search for the “Columnstore Index Scan”. |  |
|  | 1. Open the “demo items” solution folder. 2. Open the R script. | But today we're using SQL Server with R. |
|  | 1. Select “library(‘rpart.plot’)” 2. Click on Execute In Interactive.   If you get an error due to the library not being recognized, try to install it.  Install.packages(‘rpart.plot’) |  |
|  | 1. Select the “query”. 2. Click on Execute In Interactive. | So, now I'm gonna pull some data from the SQL Server.  In this case, I'm gonna  just pull the data down to the client.  So if you look at this, this is the view we prepared before. |
|  | 1. Select the different queries that exist in the file and evaluate the results. |  |
|  |  |  |
|  | 1. Open the SQL Server Object Explorer in Visual Studio. 2. Connect using SQL Server Authentication.   Server: <YOUR\_SQL\_VM>  DatabaseName: AdventureWorks.SkiResort  Username: skiresort  Password: P2ssw0rd@1   1. Open the stored procedures folder. |  |
|  | 1. Open the **PredictRentals** procedure. |  |
|  | 1. Go to the Visual Studio solution. 2. Expand the SkiResort.API project. 3. Click on “RentalsController”. 4. Go to the “CheckHighDemandAsync” controller. |  |
|  | 1. Open the SkiResort.Infrastructure project. 2. Open the SkiResortContext.cs class. 3. Go to the “EstimateRentalsAsync” controller. |  |

# LIFTS (Anomaly detection)

| Screen | Click Steps | Demoscript |
| --- | --- | --- |
| **LIFTS** | |  |
|  |  | Shows open/closed status.  **With intelligence**  Predicts wait time.  Gives warnings based on traffic.  Identifies anomalies. |
|  | 1. Open **SkiResort.DataGeneration.sln** solution. (src\SkiResort.DataGeneration directory) 2. Click on “Ctrl + F5” to run the application without debugging. | gen-skilocations simulates skiers movement around the mountain by continuously moving them from the bottom of chairlifts to the top and back. The location events are pushed to an event hub that’s then consumed by Stream Analytics. |
|  | 1. Go to the AdventureWorks.SkiResort solution. 2. Expand the SkiResort.Infrastructure project. 3. Open the Lift.cs file to check the Lift properties. | Imagine the skier sending their latitude, longitude and event time continuously…(the gen-skilocations app is running) |
|  | 1. Go to the Microsoft Azure portal. 2. Search for your Resource Group. 3. Click on the Stream Analytics Services. | I created an event hub and then I hooked up Azure stream analytics into that event hub. |
|  | 1. Click on the output to view the configuration. | I can pull data that is being generated from all these places and land it neatly organized in one place. |
|  | 1. Click on “Query”. | I look at the query. |
|  |  | I'm storing this in BLOBs. We will have all of the location history. |
|  | 1. Open the Visual Studio Cloud Explorer. 2. Open your Resource Group. 3. Open the Storage Service. 4. Open the blob and tables to check the outputs of the Stream Analytics services. | Allows us to see the storage account. |
|  | 1. View the content of the archive container. | So, you can see how stream analytics has taken all of these, and packed them in single files in sub CSVs. |
|  |  | So look at this query for a minute.  So what we're saying is, is that we're quitting the location events data stream.  In stream analytics, we support pulling in reference data. In this case, my reference data set is the locations of all of the bases of the chair lifts. |
|  | 1. View the content of liftlines table. |  |
|  |  |  |
|  | 1. View the content of liftlinearchive table. |  |
|  | 1. Click on SkiResort.API. 2. Click on LiftController.cs 3. Go to the “GetNearByAsync” controller. | We can start to predict wait times, and maybe even detect anomalies to redirect them to different part of the mountain. |
|  | 1. Go to the definition of the “SlowChairLiftAsync”. | Here I use the anomaly services that are available in the data services marketplace. |

# DINING (Tailored experiences)

| Screen | Click Steps | Demo script |
| --- | --- | --- |
| **DINING** | |  |
|  |  | Shares menu.  Takes reservations. |
|  |  | **With intelligence**  Knows where you want to eat next. |
|  | 1. Open the SQL script, the same as you opened in the previous step. 2. List the restaurants.   Select \* from restaurant   1. Click on execute. |  |
|  | 1. Open **SkiResort.DataGeneration.sln** solution. 2. Expand the gen-recomodel project. 3. Opent the program.cs file. | geo-recomodel: given 2 inputs (restaurant list, user restaurant attendance) creates a recommendations model in the pre-built AzureML recommendations API. It uses the recommendations data service created in the first steps. |
|  |  |  |
|  | 1. Expand the geo-restaurantsearch project. 2. Opent the program.cs file. | geo-restaurantsearch runs all restaurants by the recommendations API, retrieves a list of recommended restaurants for each one, and pushes the list of recommended/related restaurants to the Azure Search index. |
|  | 1. Go to the Azure Portal. 2. Go to the Resource Group. 3. Click on the Search Service. | I use a Search Service in Azure. |
|  | 1. Click on the “restaurant” index. | I have this restaurant index that has all of the typical things you would find from a search index for restaurants. |
|  | 1. Click on “Search explorer”. |  |
|  | 1. Write “grill” and click on search. | Try out the search index. |