



Allstate[®]
You're in good hands.

Overview

Allstate provides customized insurance policies to its customers with personal attention and service. Marketing in insurance terms would mean to provide the perfect quote to your customer and reduce the quote processing time.

When a car owner inquires about the car-insurance plan, insurance agents propose several quotes with different coverage options. After considering all the possible options, car owner settles with a plan and purchases it.

About the data

The data contains transaction history for customers that ended up purchasing a policy. For each customer_ID, we have given their quote history. In the dataset we have the entire quote history, the last row of which contains the coverage options they purchased.

What is a customer?

Each customer has many shopping points, where a shopping point is defined by a customer with certain characteristics viewing a product and its associated cost at a particular time.

- Some customer characteristics may change over time (e.g. as the customer changes or provides new information), and the cost depends on both the product and the customer characteristics.
- A customer may represent a collection of people, as policies can cover more than one person.
- A customer may purchase a product that was not viewed!

Coverage options

Each product has 7 customizable options selected by customers, each with 2, 3, or 4 ordinal values possible. A policy is simply a vector with length 7 whose values are chosen from each of the options listed below. The cost of a product is a function of both the product options and customer characteristics:

Option name	Possible values
A	0, 1, 2
B	0, 1
C	1, 2, 3, 4
D	1, 2, 3
E	0, 1
F	0, 1, 2, 3
G	1, 2, 3, 4

Variable Descriptions

- **customer_ID** - A unique identifier for the customer
- **shopping_pt** - Unique identifier for the shopping point of a given customer
- **record_type** - 0=shopping point, 1=purchase point
- **day** - Day of the week (0-6, 0=Monday)
- **time** - Time of day (HH:MM)
- **state** - State where shopping point occurred
- **location** - Location ID where shopping point occurred
- **group_size** - How many people will be covered under the policy (1, 2, 3 or 4)
- **homeowner** - Whether the customer owns a home or not (0=no, 1=yes)
- **car_age** - Age of the customer's car

- **car_value** - How valuable was the customer's car when new
 - **risk_factor** - An ordinal assessment of how risky the customer is (1, 2, 3, 4)
 - **age_oldest** - Age of the oldest person in customer's group
 - **age_youngest** - Age of the youngest person in customer's group
 - **married_couple** - Does the customer group contain a married couple (0=no, 1=yes)
 - **C_previous** - What the customer formerly had or currently has for product option C (0=nothing, 1, 2, 3,4)
 - **duration_previous** - how long (in years) the customer was covered by their previous issuer
 - **A,B,C,D,E,F,G** - the coverage options
 - **cost** - cost of the quoted coverage options
-

Goal of the project

If the eventual purchase can be predicted sooner in the shopping window, the quoting process is shortened and the issuer is less likely to lose the customer's business. This application is likely be applied to any business in which customers are deciding between multiple products (or multiple options for the same product). If the business can gain an insight into which product or option a customer is likely to end up choosing, they could nudge the customer toward that product (in order to increase their conversion rate), or instead nudge the customer toward a slightly more expensive product (in order to maximize their profit from that sale).

- **Dockerizing an application** - [Docker File](#) describes the step by step implementation of creating an docker image.
- **Pipelining tasks** - [Start-Pipeline](#) uses Luigi to pipelining the tasks.
- **Data Download** - [Data-download](#) describes the step by step implementation of downloading the dataset provided on the [kaggle.com](#).
- **Data Wrangling & Preprocessing** - [Clean Data Script](#) describe the step by step process to clean and separate the data into 3 different files for purchases, quotes & last quotes.
- **Exploratory data analysis** - [EDA Notebook](#) performs Exploratory data analysis on the purchases and quotes. Data scientist view of [Power BI dashboard](#) to illustrate your key insights.
- **Classification Of Coverages** - Classify all the 7 coverages based on the customer's profile.
- **Predicting Cost** - Based on the possible coverages and customer's profile, predict the cost, customer is likely to pay.
- **Create a WebApp for user-interaction** - [Allstate Insurance](#) webapp takes the inputs from a customer and provides the best possible quote, with the option of customizing it.

Docker File

[Docker File](#) has following dependency.

```
FROM python
```

```
RUN apt-get update && \  
    apt-get clean && \  
        rm -rf /var/lib/apt/lists/*
```

Installation of dependencies

```
RUN pip install 'pandas' \  
    'numexpr' \  
    'numpy' \  
    'matplotlib' \  
    'scipy' \  
    'scikit-learn' \  
    'beautifulsoup4' \  
    'lxml' \  
    'html5lib' \  
    'boto' \  
    'luigi' \  
    'mechanicalsoup'
```

Copying scripts from local to docker machine

```
ADD clean_data.py clean_data.py  
ADD download_data.py download_data.py  
ADD start_pipeline.py start_pipeline.py
```

Luigi Pipeline

[Start Pipeline Script](#) description.

Start Task

Task Requirement

```
def requires(self):  
    return Clean_data()
```

Task Output

```
def output(self):  
    return luigi.LocalTarget('logs/start_pipeline.log')
```

Run Task

```
#take AWS keys from user  
  
#create connection to AWS  
conn = boto.connect_s3(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)  
bucket = conn.lookup(bucket_name)  
  
#check if files are already present
```

```

if not (bucket == None):
    for key in bucket.list(delimiter='/'):
        for f in bucket.list(key.name):
            if(f.name == filePath):
                #files are already on AWS

#upload files on AWS
k = Key(bucket)
k.key = file
k.set_contents_from_filename(file)

```

Clean_data Task

Task Requirement

```

def requires(self):
    return Download_data()

```

Task Output

```

def output(self):
    return{
        'output1' : luigi.LocalTarget('data/processed_data/purchase_data.csv'),\
        'output2' : luigi.LocalTarget('data/processed_data/quotes_data.csv'),\
        'output3' : luigi.LocalTarget('data/processed_data/la

```



```
st_quote_data.csv')
}
```

Run Task

```
#run clean data script
```

Download_data Task

Task Requirement

```
def requires(self):
    return None
```

Task Output

```
def output(self):
    return luigi.LocalTarget('luigi/download_data_completed.txt')
```

Run Task

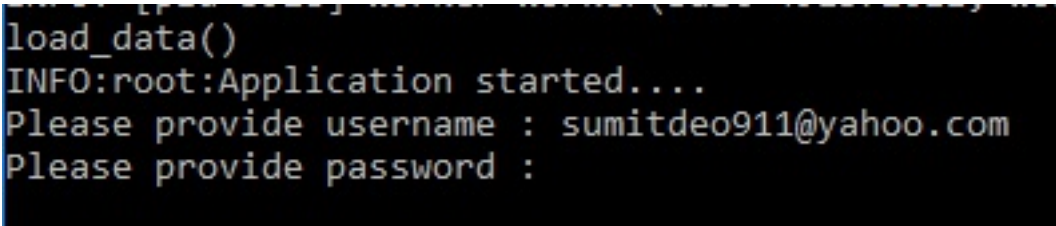
```
#run download data script
```

Data Download

[Data-download Script](#) description.

Get login credentials

```
username = input("Please provide username : ")
password = input("Please provide password : ")
```

A terminal window with a black background and green text. The text shows the script's execution: it starts with 'load_data()', followed by 'INFO:root:Application started....', then prompts for a username where 'sumitdeo911@yahoo.com' is entered, and finally prompts for a password.

```
load_data()
INFO:root:Application started....
Please provide username : sumitdeo911@yahoo.com
Please provide password :
```

Logging to the site

```
login_page = browser.get(url)
login_form = login_page.soup.find('form', {"id":"signin"}
)
login_form.find("input", {"id" : "UserName"})["value"] =
username
login_form.find("input", {"id" : "Password"})["value"] =
password
login_form.find("input", {"id" : "RememberMe"})["value"]
= True
response = browser.submit(login_form, login_page.url)
```

Check if username & password are

correct

```
if (response.url == 'https://www.kaggle.com/account/login
.action'):
    print(username + " logged in successfully!")
    --download files--
else:
    print("Either username or password or both are wrong.
.")
```

Download & unzip files

```
with urlopen(baseURL + extn) as zipresp:
    with ZipFile(BytesIO(zipresp.read())) as zfile:
        zfile.extractall(path)
```

Data Wrangling & Preprocessing

[Clean Data Script](#) description.

Fill missing data

Below function will fill all the NaNs with some default values.

```
def funFillMissingData(df):
    df['car_value'].fillna('z', inplace=True)
    df['C_previous'].fillna(df['C'], inplace=True)
```

```
df['duration_previous'].fillna(0, inplace=True)

return df
```

Derive new columns

Below function will derive new features from the existing features.

```
def funDeriveColumns(df):

    df['plan'] = df['A'].map(str) + df['B'].map(str) + df
['C'].map(str) + df['D'].map(str) + df['E'].map(str) + df
['F'].map(str) + df['G'].map(str)
    df['hour'] = df['time'].dt.hour
    df['is_weekend'] = np.where(df['day'].isin(list(range
(0, 4)))) , 0, 1)
    df['is_family'] = np.where(((df['group_size'] > 2) &
(df['age_youngest'] < 25) & (df['married_couple'] == 1)),
0, 1)
    df['agediff'] = df['age_oldest'] - df['age_youngest']
    df['is_individual'] = np.where(((df['agediff'] == 0)
& (df['group_size'] == 1)), 1, 0)
    return df
```

Change data types

Below function will change the data type of columns to category.

```
def funChangeDataTypes(df):
```

```
df['state_code'] = df['state'].astype('category')
df['state_code'] = df['state_code'].cat.codes
df['car_value_code'] = df['car_value'].astype('category')
df['car_value_code'] = df['car_value_code'].cat.codes

columns = #columns to be changed

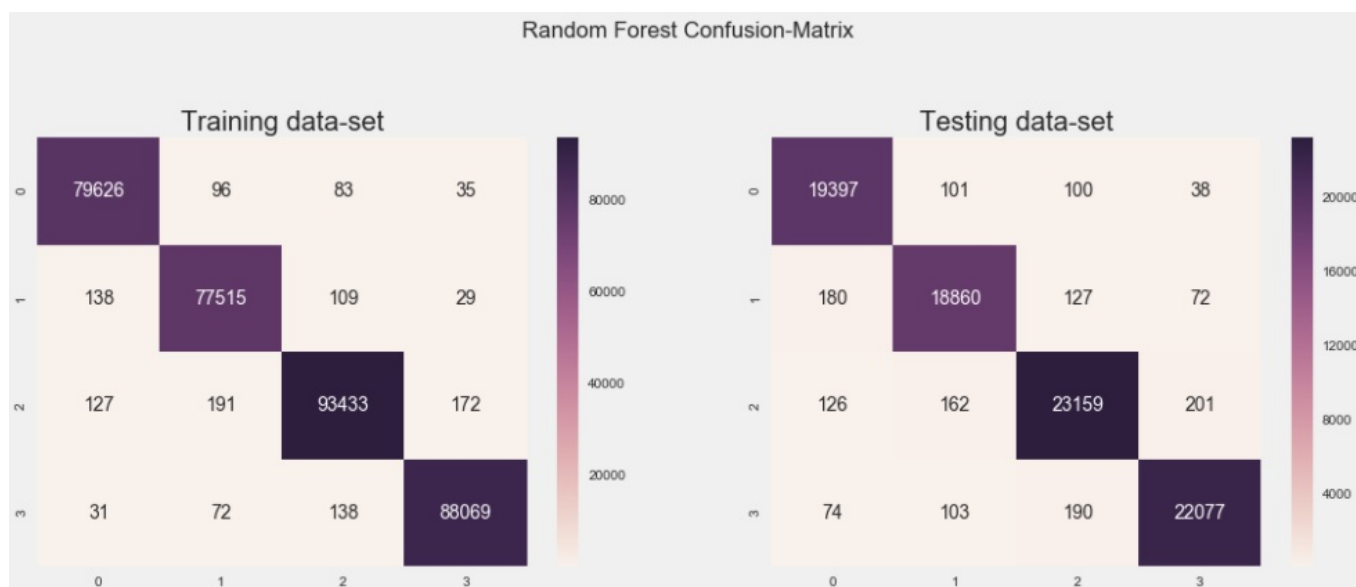
for col in columns:
    df[col] = df[col].astype('category')
return df
```

Predict Risk Factor

In the raw data, we have found that almost in 33% of the rows, 'risk_factor' is missing. After carefully looking into the data, we came to the conclusion that, instead of filling these NaN by mean/max/min, we can predict it with the help of other available features.

We have used Random Forest Multiclass Classifier to classify the Risk Factor.

Confusion Metric



Accuracy Score

```
#calculating accuracy score
random_forest_acc_matrix_train = metrics.accuracy_score(train_y, train_y_predicted)
random_forest_acc_matrix_test = metrics.accuracy_score(test_y, test_y_predicted)
print('Random Forest accuracy')
print('Training data : ',random_forest_acc_matrix_train)
print('Testing data : ',random_forest_acc_matrix_test)
```

```
Random Forest accuracy
Training data : 0.996407386484
Testing data : 0.982652088458
```

```
def funPredictNFillRiskFactor(df):
    dfrisk = df[~ pd.isnull(df['risk_factor'])]
    dfnorisk = df[pd.isnull(df['risk_factor'])]
    columns = ['location', 'car_age', 'age_oldest', 'age_
youngest', 'state_code']
    train_X = dfrisk[columns]
    train_y = dfrisk['risk_factor']
    model = RandomForestClassifier(n_jobs=2)
    model.fit(train_X, train_y)
    test_X = dfnorisk[columns]
    dfnorisk['risk_factor'] = model.predict(test_X)
    test_X = dfnorisk[columns]
    dfnorisk['risk_factor'] = model.predict(test_X)
```

```
df = pd.concat([dfrisk, dfnorisk])  
  
return df
```

Write data to CSVs in chunks

```
def chunker(seq, size):  
    return (seq[pos:pos + size] for pos in range(0, len(s  
eq), size))
```

Snapshot of cleaned data

customer_ID	shopping_pt	record_type	day	state	location	group_size	homeowner	car_age	car_value	risk_factor	age_oldest	age_youngest
10000000	9	1	0	IN	10001	2	0	2	g	3	46	42
10000005	6	1	3	NY	10006	1	0	10	e	4	28	28
10000013	4	1	4	WV	10014	2	1	3	d	3	62	60
10000014	6	1	1	MO	10015	1	0	5	d	3	32	28
10000023	7	1	1	PA	10025	1	1	2	f	3	63	63
10000025	7	1	0	OH	10027	2	1	15	d	2	52	17
10000026	4	1	3	OH	10028	2	1	5	d	3	59	57
10000027	6	1	2	IN	10029	1	0	17	e	4	50	50
10000028	6	1	6	FL	10030	2	1	1	d	1	44	20

Exploratory data analysis

Installation

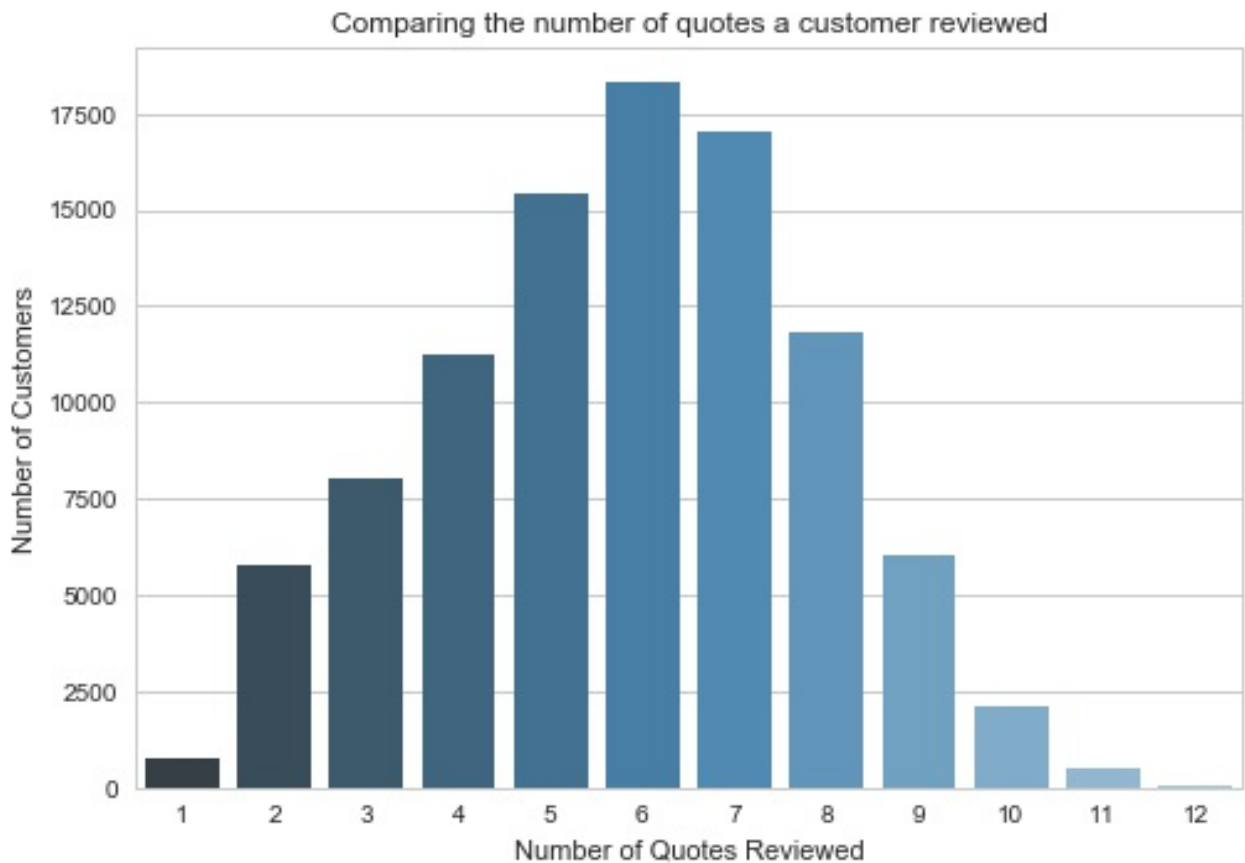
[EDA Notebook](#) has following dependency.

```
$ pip install pandas  
$ pip install numpy  
$ pip install seaborn  
$ pip install plotly
```

Number of shopping points

While working on exploratory data analysis, we found that more than 80% of purchases are finalized after considering more than 5 quotes. If the eventual purchase can be predicted sooner in the shopping window, the quoting process is shortened and the issuer is less likely to lose the customer's business.

```
#by shopping_pt and counting the total number of records
seriesCount = dfLastQuote['customer_ID'].groupby(dfLastQuote['shopping_pt']).count()
columns=['Number of Quotes Reviewed', 'Count of Customers']
dfSummary = pd.DataFrame({'Number of Quotes Reviewed':seriesCount.index,'Count of Customers': seriesCount})
```

Predictive power of final quote before purchase

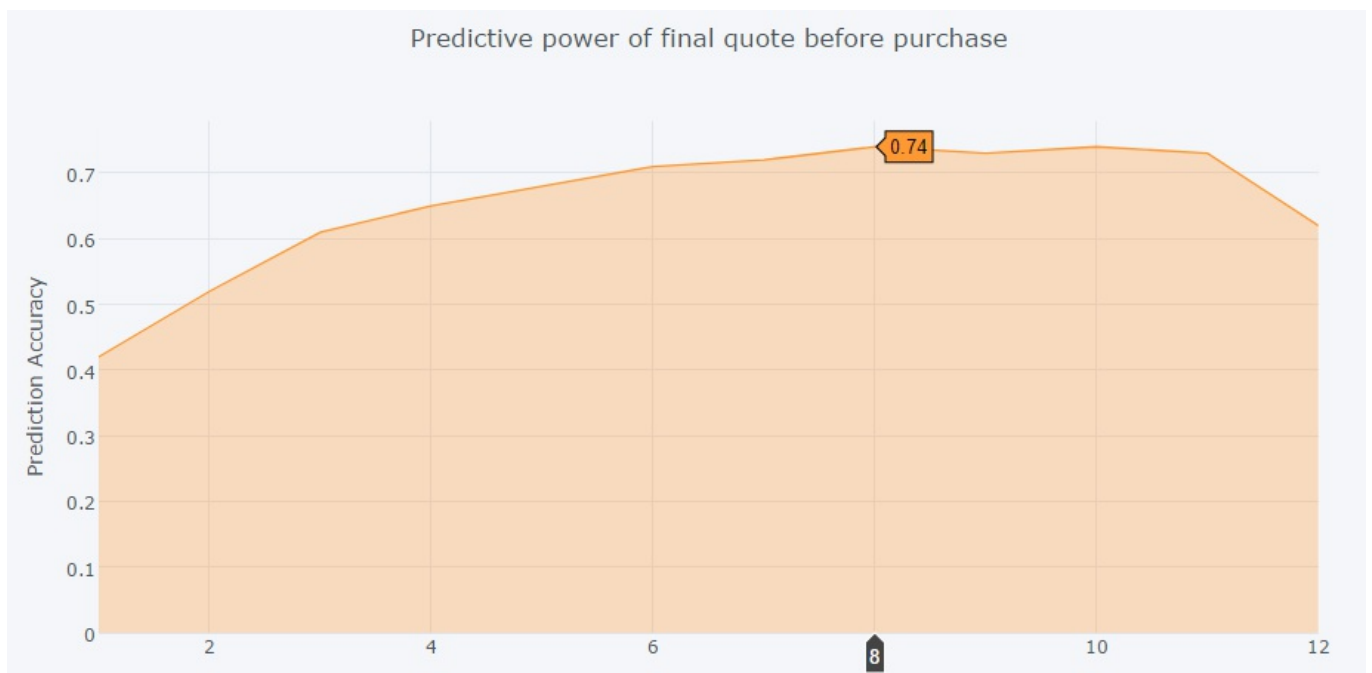
As seen in the plot below, the final quote a customer requests before the “purchase point” is hugely predictive of which options they will actually purchase. The final quote correctly predicted the purchased options 50% to 75% of the time, with that percentage steadily increasing as customers review more quotes. This proves that customer continues to take quotes unless and until, he is 100% satisfied with the coverages & cost.

```
#by shopping_pt and counting the total number of records  
# df = dfLastQuote[dfLastQuote['is_changed'] == 0]
```

```

seriesCount = dfLastQuote[dfLastQuote['is_changed'] == 0]
['customer_ID'].groupby(dfLastQuote['shopping_pt']).count
()
columns=['Number of Quotes Reviewed', 'Prediction Accuracy']
dfSummary1 = pd.DataFrame({'Number of Quotes Reviewed':seriesCount.index,'Prediction Accuracy': seriesCount})

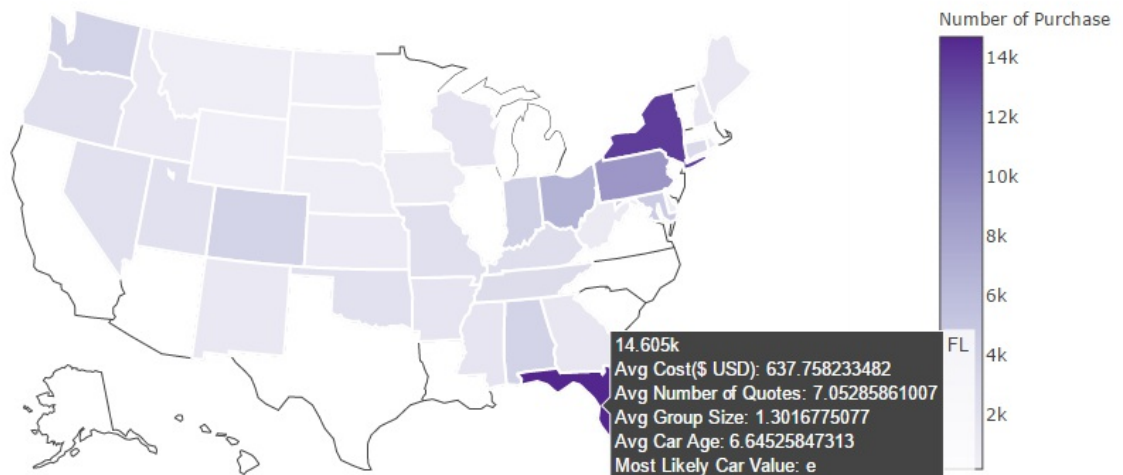
```



Summarizing data by States

- Summary of purchases are displayed on the plot using plotly in offline mode.
- Group the data by State and take average/mode of the required columns.
- Create a dictionary of the summary and display on the plotly plot.
- Hover over each state to get the summary.

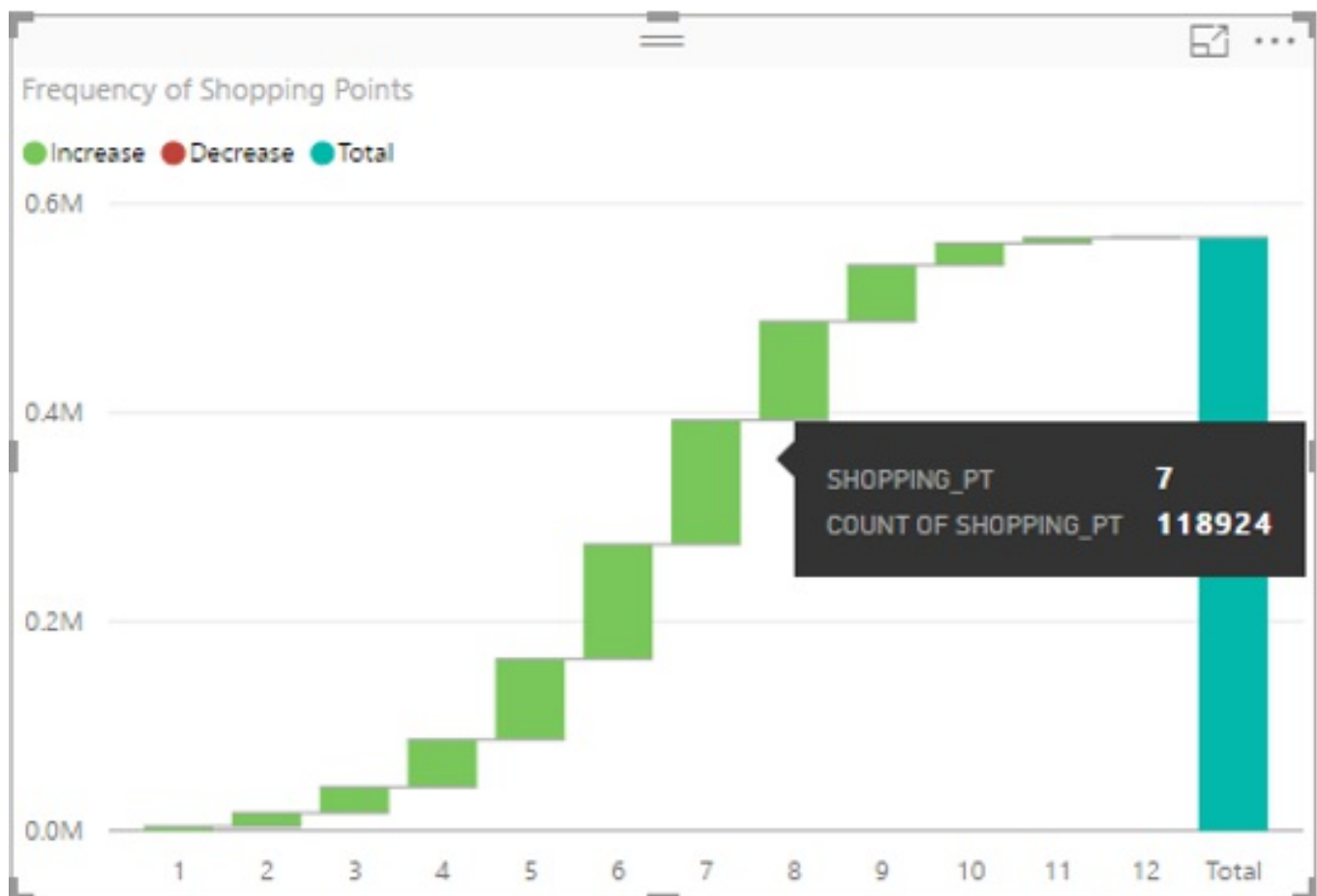
Total number of purchase by state
(Hover over state for other metrics)



Power BI Dashboard

PowerBI Dashboard

Allstate Auto insurance agent posts several quotes to the users before user agreeing on one quote. Our objective is to reduce the number of quotes to minimum to save customer's time and increase customer satisfaction. Below, we have visualized users' shopping point (a number that indicates how many quotes does a common user go through) before making a decision. As we can see, majority of the users have taken from 6-8 turns before deciding on a quote. Our application aims a minimizing these turns so that we can come up with an optimal quote that the user.



User quote selection pattern using shopping point

Popular plans and frequency Analysis

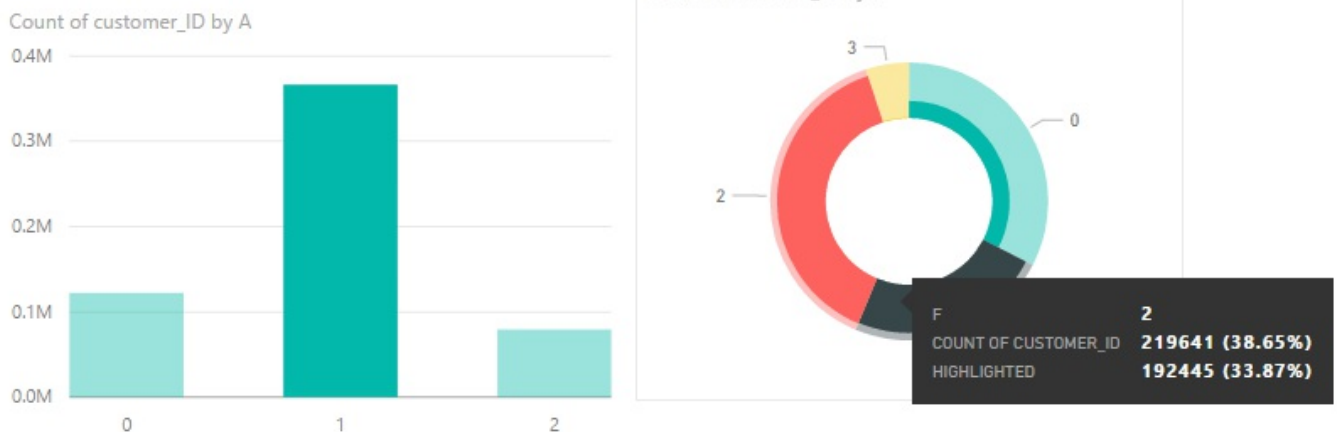
We have ranked the plans based on the number of sales by customers. Most popular plan is 1133123 with a count of 59 followed by others. On the right-hand side, we have analyzed the shopping frequency of each plan and found out highest percentage of people 32.20 % choose that plan on the first go. That sums up our application objective well.



Popular quotes with count shopping frequencies

Correlation Analysis between Different Plans

We have calculated the correlation between different insurance plans and found out some to be highly correlative. Below is the correlation analysis for most correlated plan C and D. Below bar chart explains if a user goes for Plan C type.



We have also analyzed other plans like Plans A and D, B and E and found them to be highly correlated. As we can see above, if customer is take Plan A Type 1 then he has opted for Plan F type 2. Out of 2 million customers 1.9 have went for the same plan. That proves our assumption to be correct.

Insurance cost and Risk Factor using Age distribution

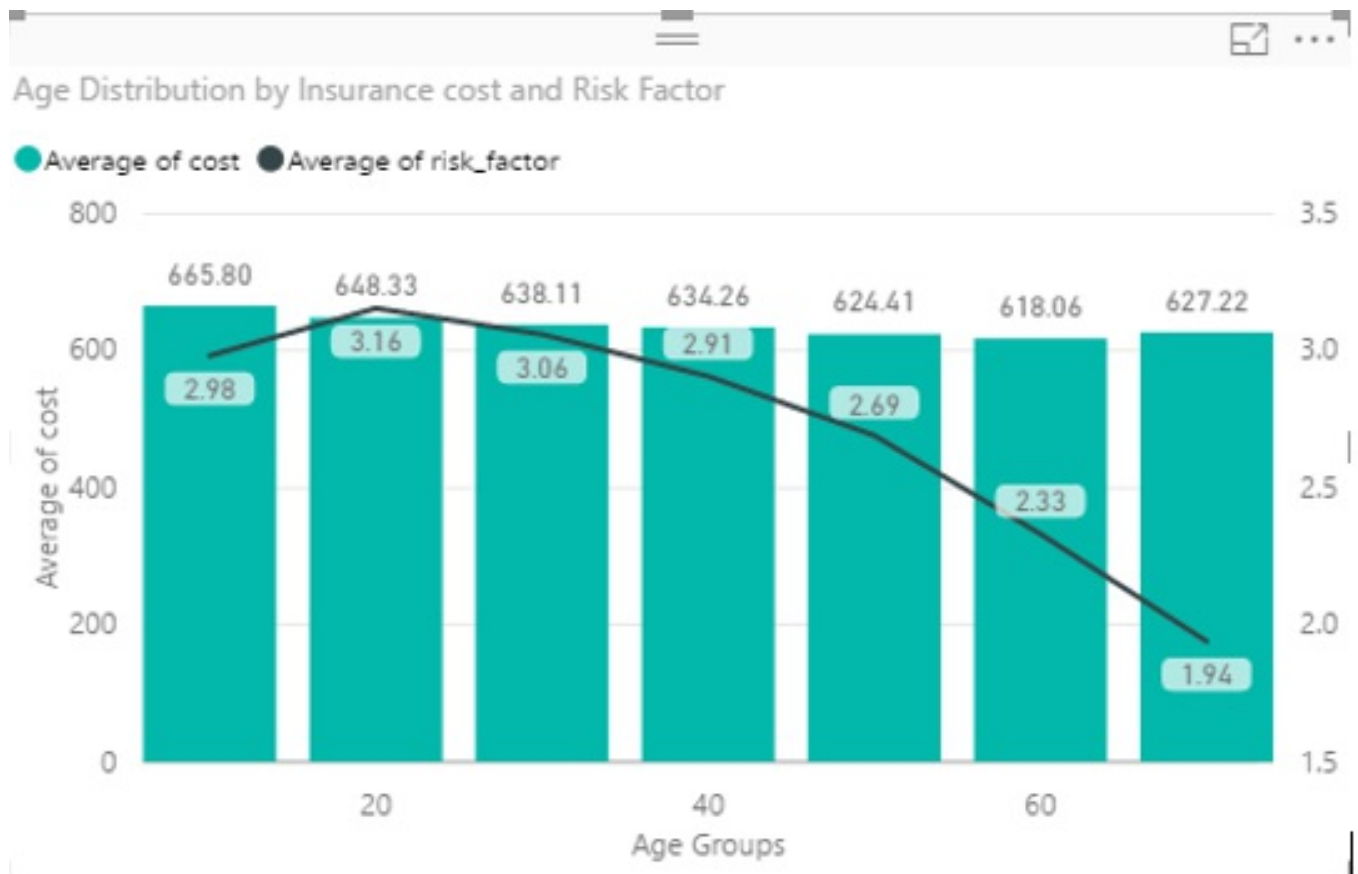
We have divided the users age data in different bins and analyzed the average insurance cost and risk factor. We have figured out that the average risk score has fallen as the age group increases that gives an idea how younger aged people tends to have higher risk score as compared to adults. The 20's age group has the highest average risk score 2.89. Even the average insurance cost is a bit on the higher side for the younger people with \$647 as compared to adults with \$630 average score.



Risk Factor and Car Age distribution analysis

Below line chart explains the trend on how the risk factor increases with the car age. X-axis represents the Car Age bin whereas the y-axis represent the risk factor. Risk factor is minimum where the car age is 1 year and it goes on increasing till it reaches max of 2.92 when the car

age is 15 years. This shows a trend how risk factor is directly related to the car age.



Azure Machine Learning

We have created total 3 API from the Azure ML.

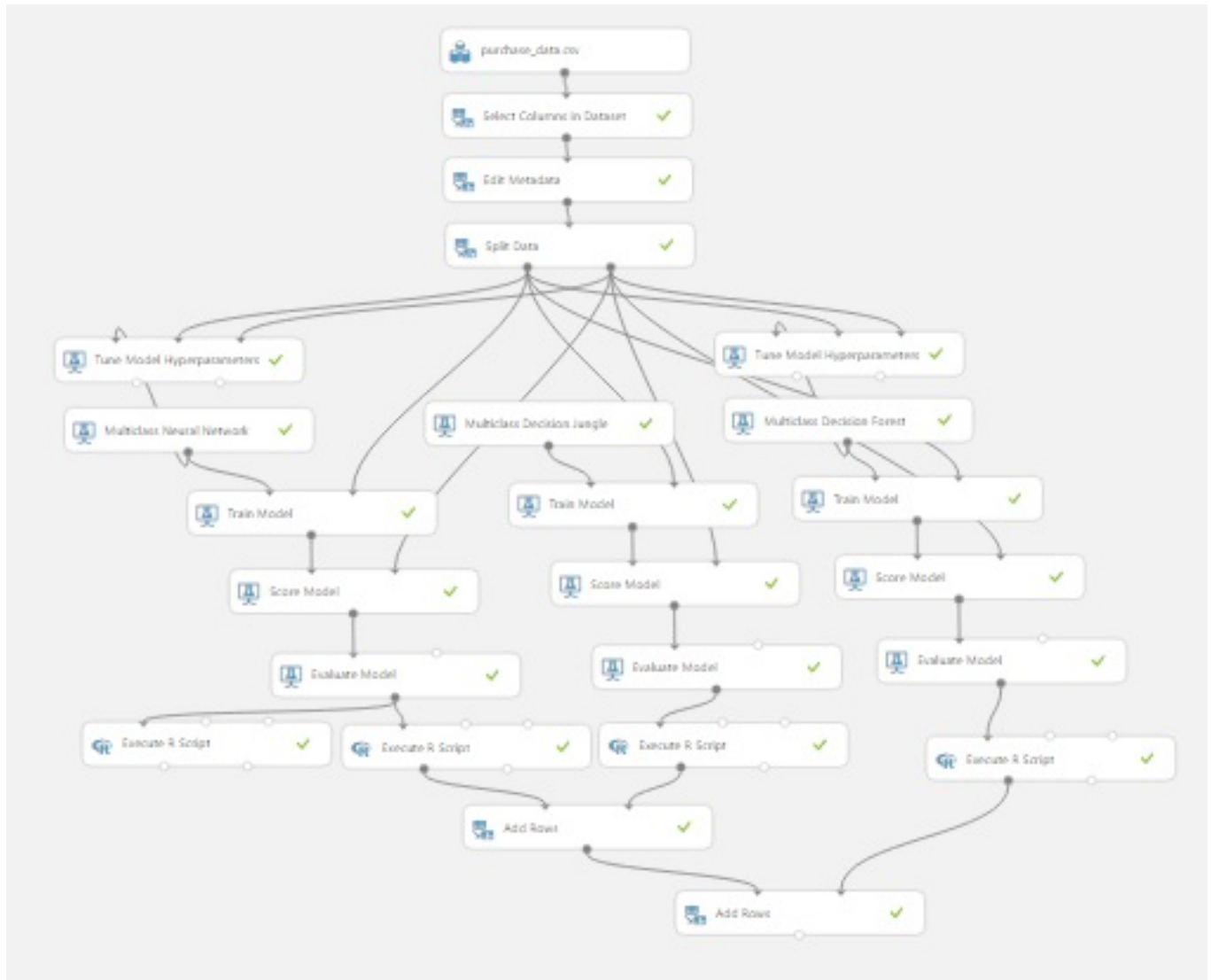
- To classify the risk factor.
- To classify all the 7 coverages.
- To predict the cost of the policy.

Classify Risk Factor

Staging

Stage_Classify_Risk_Factor is a staging experiment to find out the best model for the classification.

We have compared the Multiclass Neural Network, Multiclass Decision Jungle & Multiclass Decision Forest for the risk factor classification.



Based on the below results we have decided to deploy the Random Forest model and create an API.

rows

3

columns

3

view as

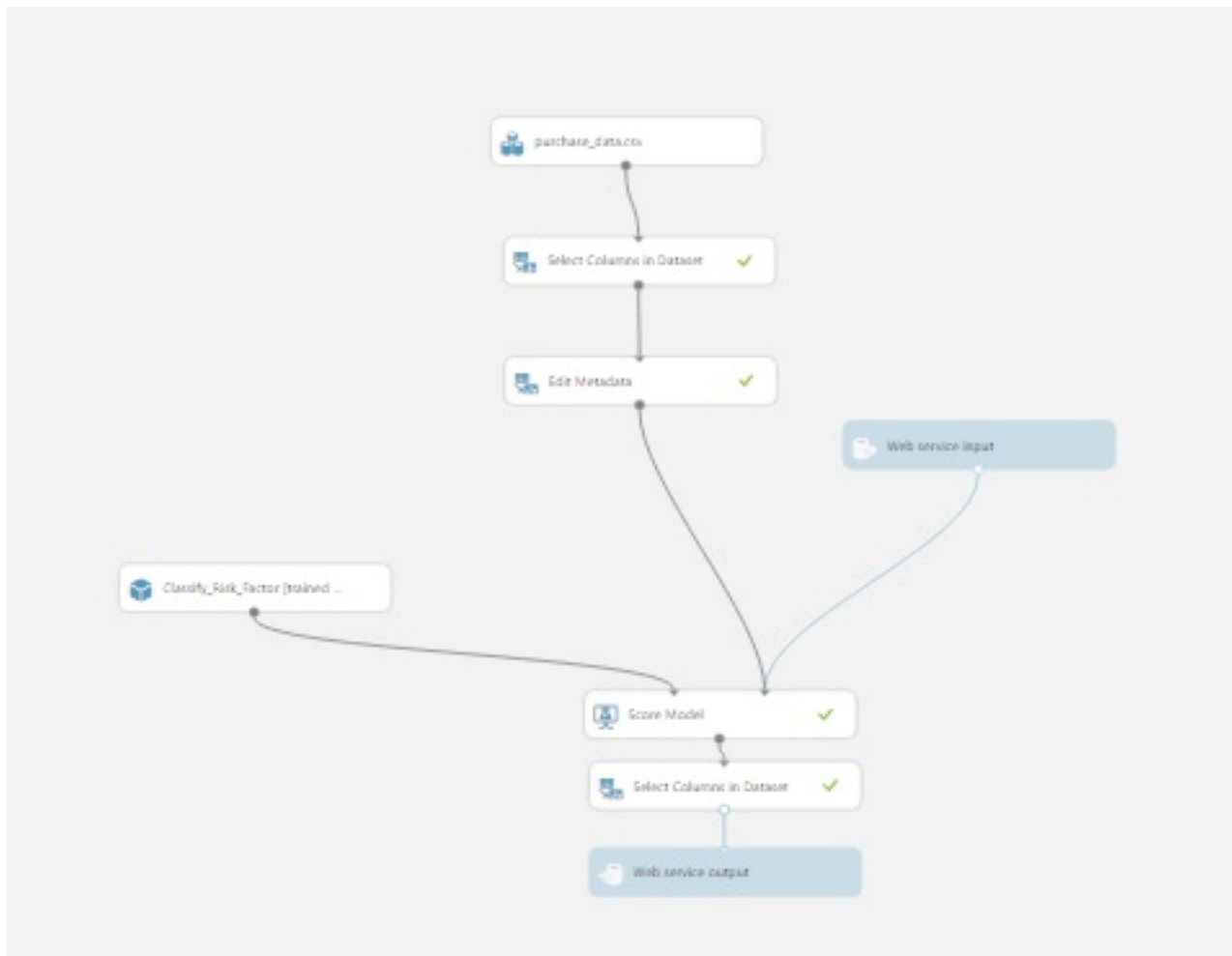


Algorithm	MacroPrecision	MacroRecall
Neural Network	0.443089	0.441688
Decision Jungle	0.458928	0.458209
Decision Forest	0.449083	0.453019

		Predicted Class			
		1	2	3	4
Actual Class	1	48.3%	21.7%	17.2%	12.9%
	2	25.9%	26.1%	24.8%	23.3%
	3	10.5%	12.7%	46.2%	30.6%
	4	6.6%	8.9%	23.9%	60.6%

Deployment

Classify_Risk_Factor is a Random Forest model to classify the Risk Factor.



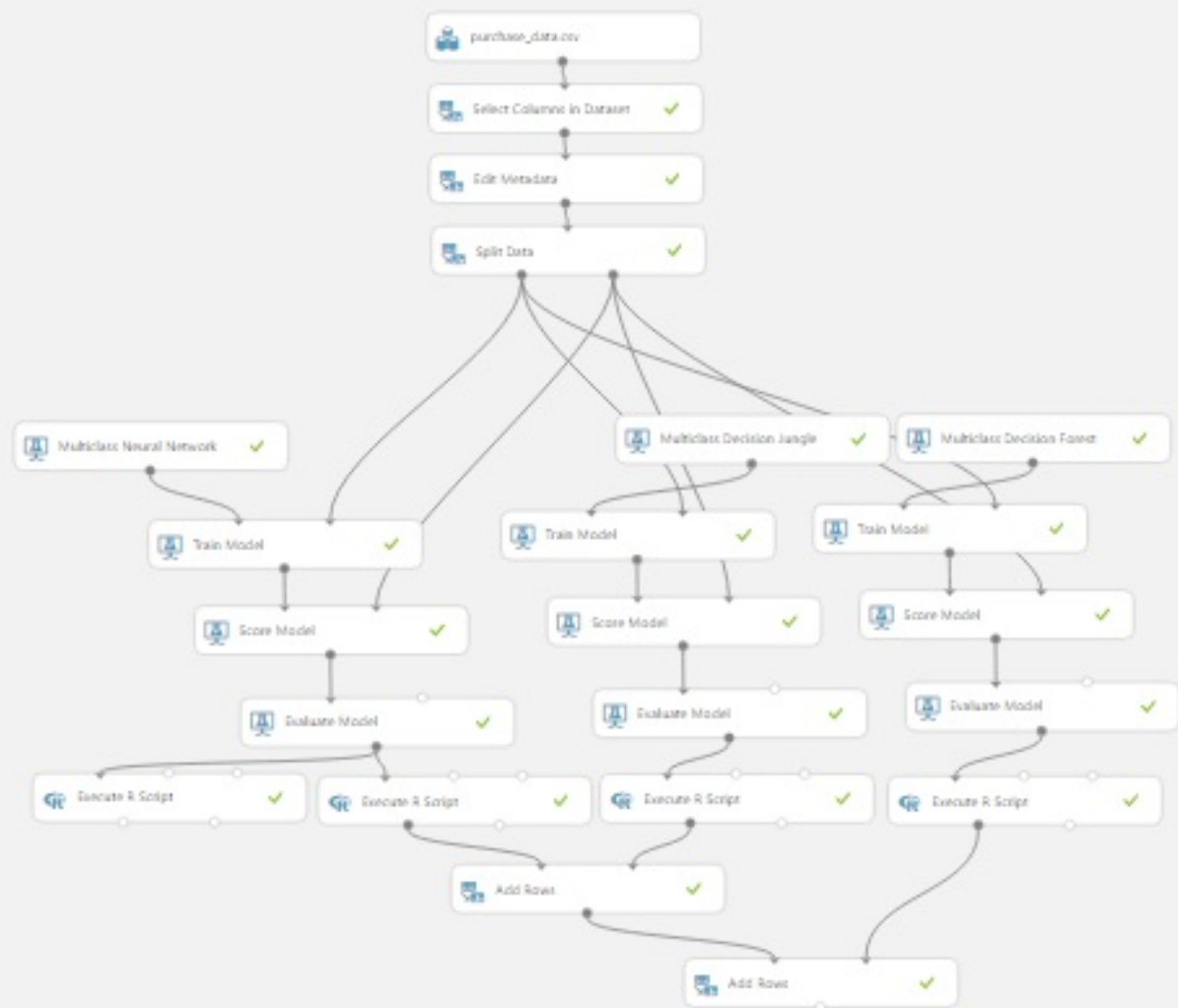
Above experiment we have deployed as an [Classify_Risk_Factor](#) API.

Classify Coverages

Staging

[Stage_Classify_A](#) is a staging experiment to find out the best model for the classification of coverage A.

We have compared the Multiclass Neural Network, Multiclass Decision Jungle & Multiclass Decision Forest for the risk factor classification.



Based on the below results we have decided to deploy the Multiclass decision jungle and create an API.

rows
3

columns
3

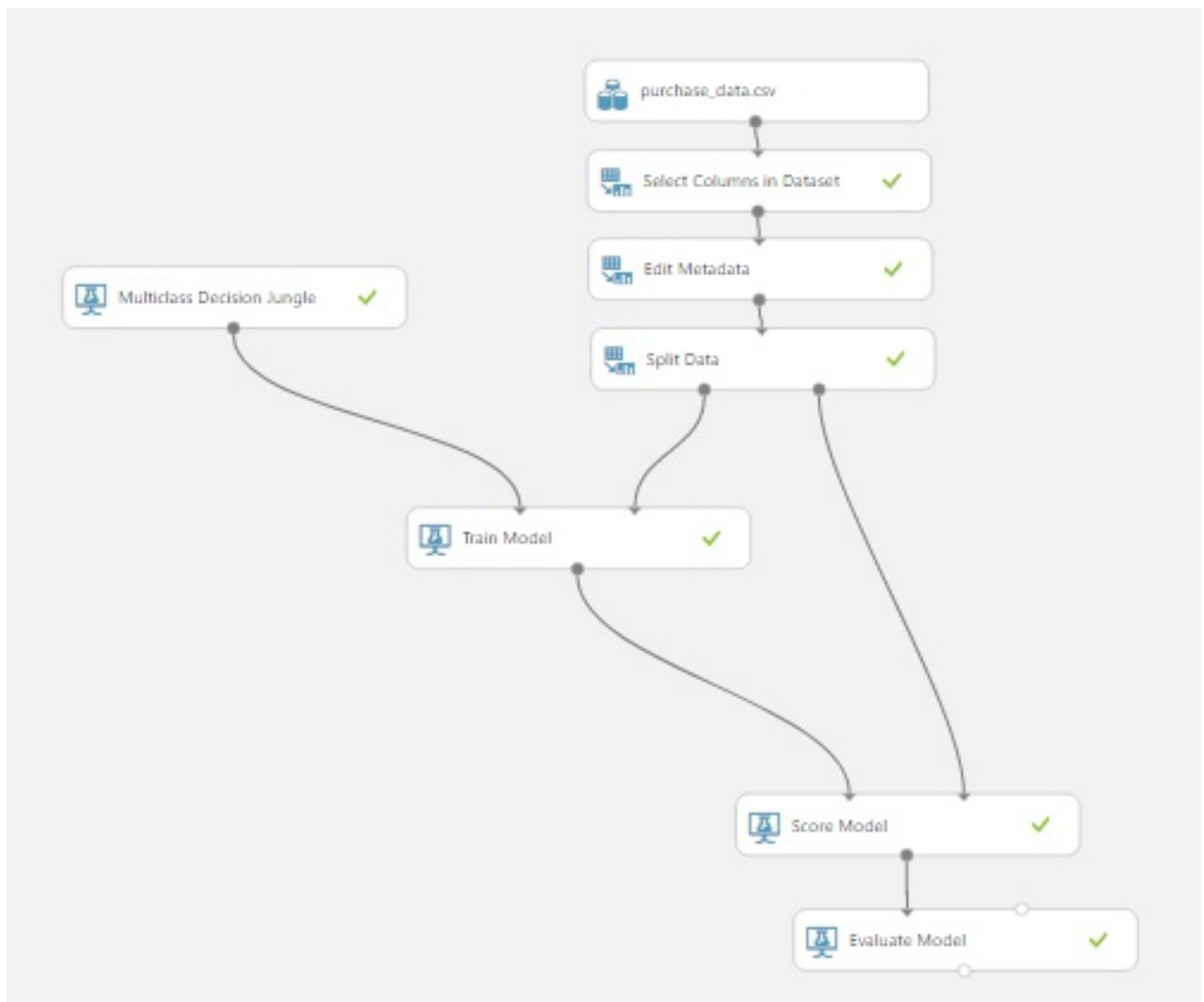
view as
 

Algorithm	MacroPrecision	MacroRecall
Neural Network	0.62279	0.583849
Decision Jungle	0.674376	0.582727
Decision Forest	0.626267	0.59485

		Predicted Class		
		0	1	2
Actual Class	0	78.1%	20.2%	1.7%
	1	8.7%	86.6%	4.8%
	2	9.1%	77.2%	13.8%

Deployment

Classify_A is a Multiclass Descision Jungle model to classify the Risk Factor.



Similarly we have deployed all the models for the coverages B-G.

All the above experiments for coverages A to G are deployed in only one [first_quote](#) API. We have done this to predict the coverages based on other coverages and customer's profile.

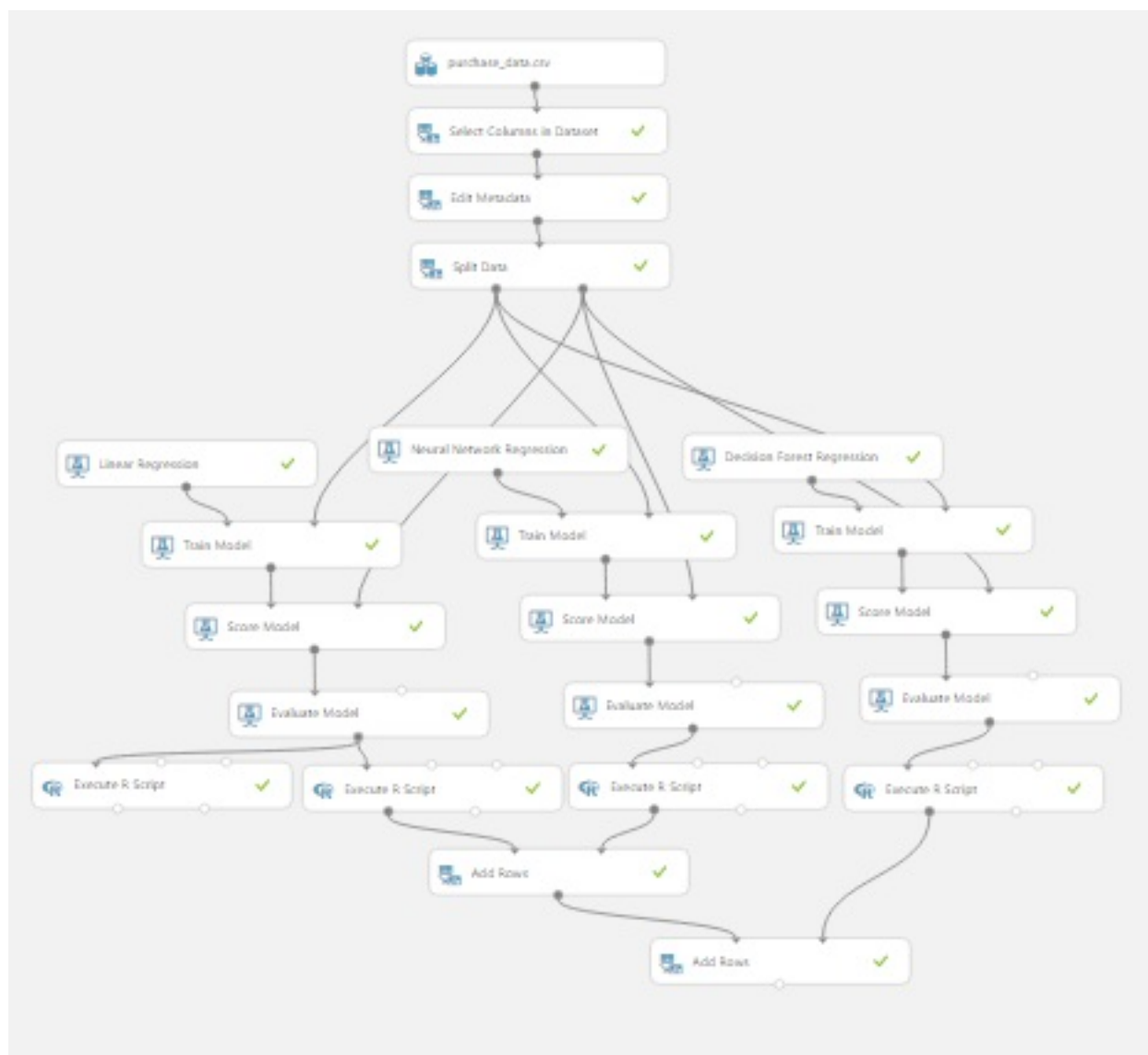
Predict Cost

Staging

[Stage_Predict_Cost](#) is a staging experiment to find out the best model

for the prediction of cost.

We have compared the Neural Network, Linear Regression & Decision Forest for the cost regression.









Based on the below results we have decided to deploy the decision forest regression and create an API.

Stage_Predict_Cost > Add Rows > Results dataset

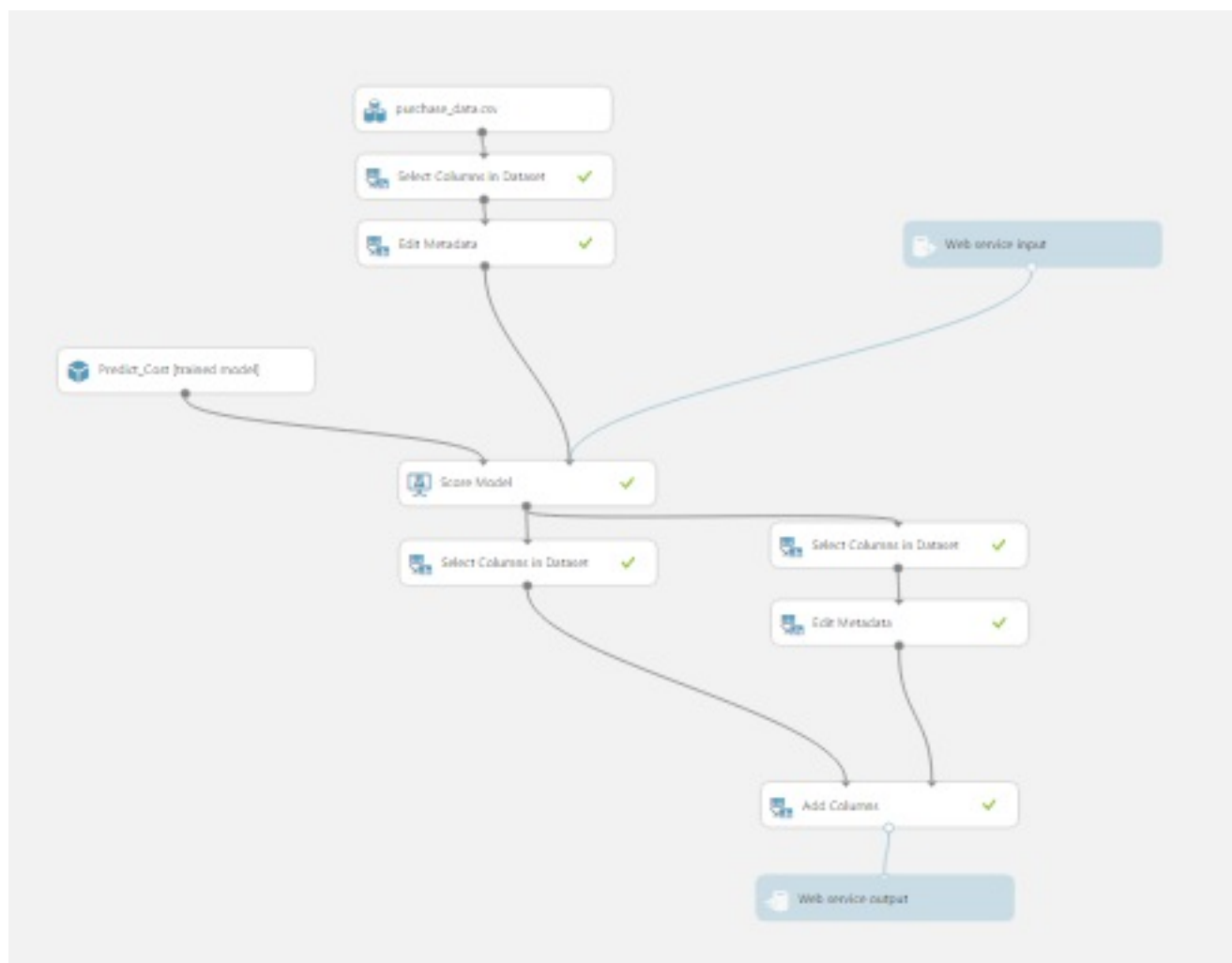
rows
3

columns
5

	Algorithm	MeanAbsoluteError	RootMeanSquaredError	RelativeAbsoluteError	RelativeSquaredError
view as 					
	Linear Regression	24.830726	31.733584	0.728342	0.545178
	Neural Network	28.18666	35.201434	0.826779	0.670843
	Random Forest	24.011497	30.954087	0.704312	0.518724

Deployment

predict_cost is a Decision Forest Regression model to predict the cost.



Above experiment for the prediction of cost is deployed as [predict_cost](#) API.

How to use this application

pull docker image

```
docker build -t sumit91188/final .
```

run docker image

```
docker run -d sumit91188/final tail -f /dev/null
```

check the container running

```
docker ps -a
```

run the pipeline through container

```
docker exec -it <container_id> python start_pipeline.py s  
tart_task --local-scheduler
```

to go inside the container and check results

```
docker exec -it <container_id> /bin/bash
```