Quick Surge Price Prediction for Cab aggregator

JanataHack: Mobility Analytics

Introduction

With the upcoming cab aggregators and demand for mobility solutions, the pa
has seen immense growth in data collected from commercial vehicles with ma
contributors such as Uber, Lyft and Ola to name a few.

There are loads of innovative data science and machine learning solutions bei
implemented using such data and that has led to tremendous business value fo
organizations.

This is a Hackathon relating to Mobility Business conducted by Analytics Vidhya.
presentation is about my attempt in tackling the challenge

Problem Statement

Welcome to Sigma Cab Private Limited - a cab aggregator service. Their custo
download their app on smartphones and book a cab from any where in the cit
operate in. They, in turn search for cabs from various service providers and prov
option to their client across available options. They have been in operation for l
a year now. During this period, they have captured surge_pricing_type from the
providers.

You have been hired by Sigma Cabs as a Data Scientist and have been asked
predictive model, which could help them in predicting the surge_pricing_type p
This would in turn help them in matching the right cabs with the right customers
efficiently.

Problem Statement

Welcome to Sigma Cab Private Limited - a cab aggregator service. Their custo
download their app on smartphones and book a cab from any where in the cit
operate in. They, in turn search for cabs from various service providers and prov
option to their client across available options. They have been in operation for l
a year now. During this period, they have captured surge_pricing_type from the
providers.

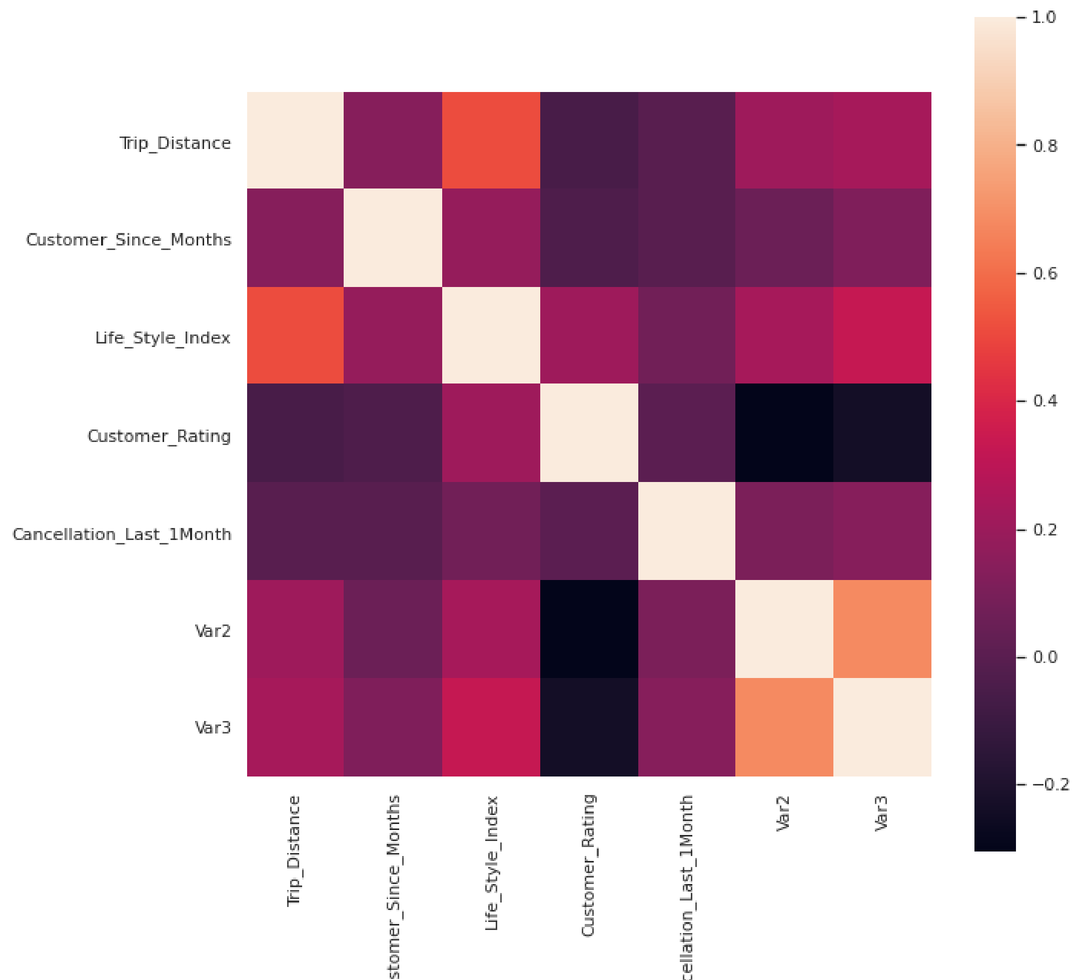
You have been hired by Sigma Cabs as a Data Scientist and have been asked
predictive model, which could help them in predicting the surge_pricing_type p
This would in turn help them in matching the right cabs with the right customers
efficiently.

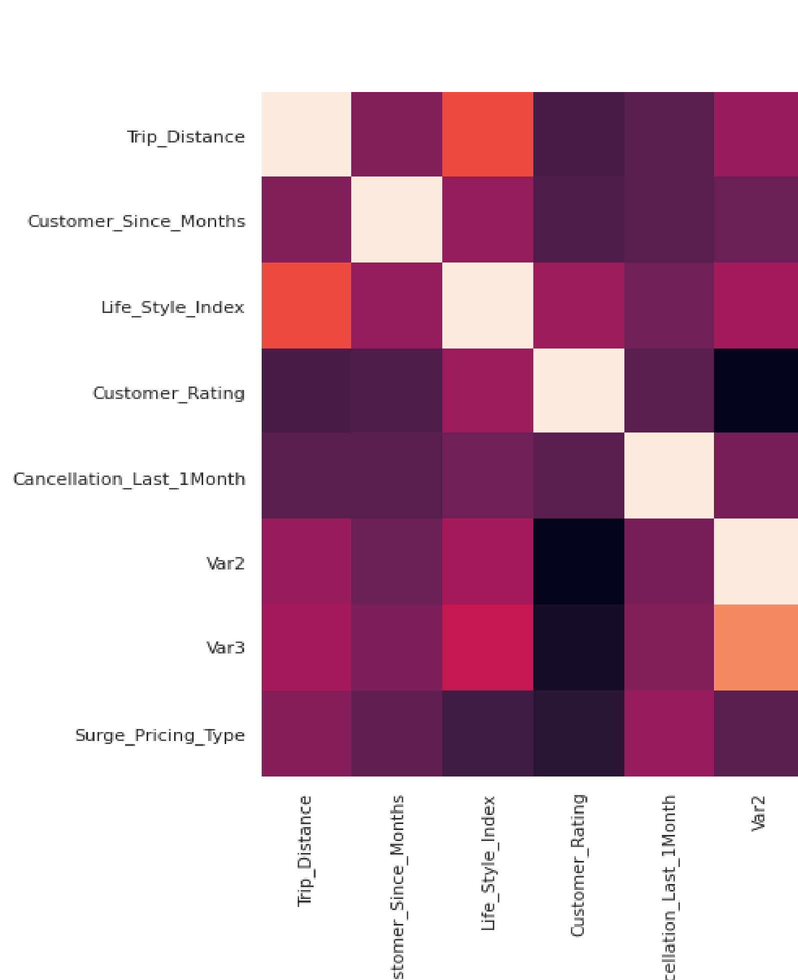
Preliminary Understanding

- Train and Test data shows similar pattern in their mean and quartile distribution. This is great.
  that the test data is similar to that of train and predictions on Train might work on Test
- Train and Test have no empty Train_ID, Train_Distance
- We have few NaN in Type_of_Cab for both Train and Test. Lets create a new category 'F' with all the NaN values
- Customer_Since_Months has few NaN values and replace them with 0. They are the newbie
  ices.
- Life_Style_Index, Confidence_Life_Style_Index. This is a propritery value by the cab compan
  o idea how it is derived. Can think of omitting the NaN rows. Since, replacing them with 0 m
  hing different. Or, can perform EDA and decide later.
- Destination_Type, Customer_Rating, Cancellation_Last_1Month have no missing values.
- Var1 is masked by the company and is very sparse. We definitely cant remove all records w
  nd neither assume them to be 0. we could take a call on this after EDA.

Correlation check

Can see Var2 and Var3 correlated in both Test and Train and have removed Var2



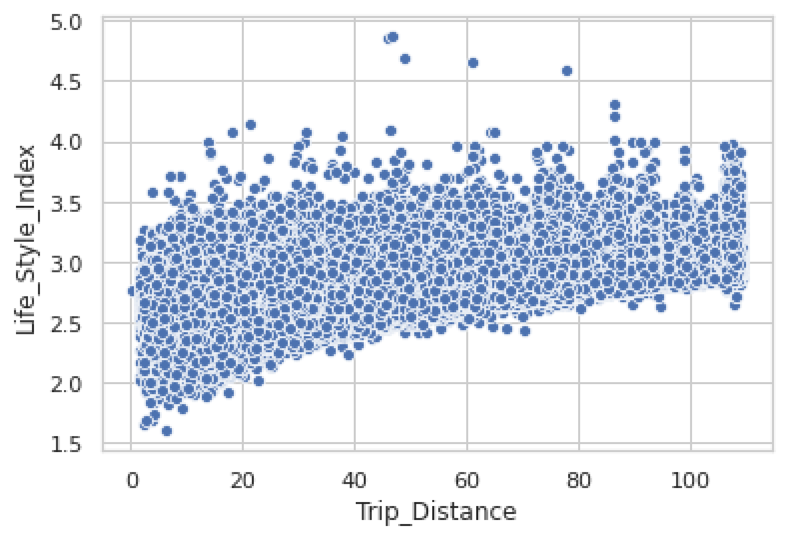


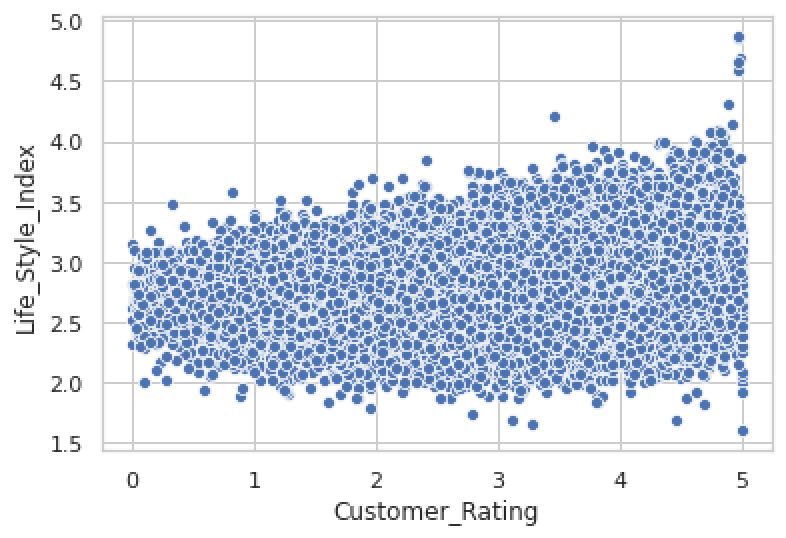
Preliminary Understanding

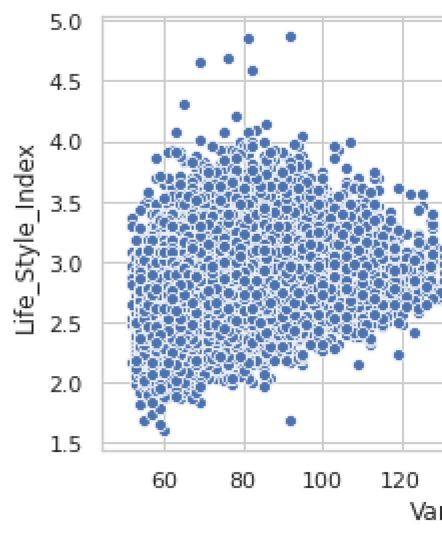
- Train and Test data shows similar pattern in their mean and quartile distribution. This is great. that the test data is similar to that of train and predictions on Train might work on Test
- Train and Test have no empty Train_ID, Train_Distance
- We have few NaN in Type_of_Cab for both Train and Test. Lets create a new category 'F' with all the NaN values
- Customer_Since_Months has few NaN values and replace them with 0. They are the newbie ices.
- Life_Style_Index, Confidence_Life_Style_Index. This is a propritery value by the cab compan o idea how it is derived. Can think of omitting the NaN rows. Since, replacing them with 0 m hing different. Or, can perform EDA and decide later.
- Destination_Type, Customer_Rating, Cancellation_Last_1Month have no missing values.
- Var1 is masked by the company and is very sparse. We definitely cant remove all records w nd neither assume them to be 0. we could take a call on this after EDA.

EDA to understand Life_Style_Index

From the 3 scatter plots, we can notice that most of the values of Life_style_index is distributed b
For simplicity, we fill assume NaN values with mode values for both Train and Test(2.7)

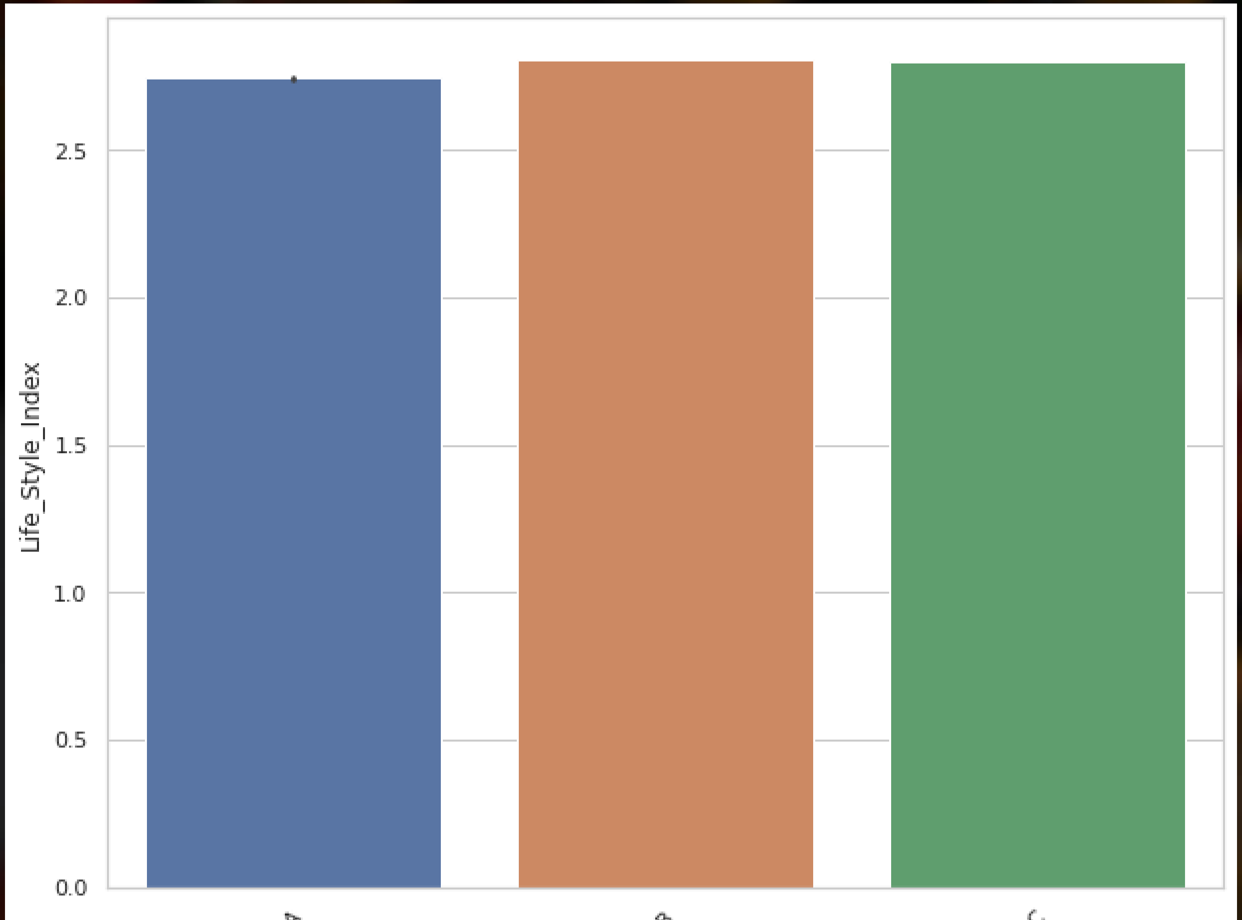






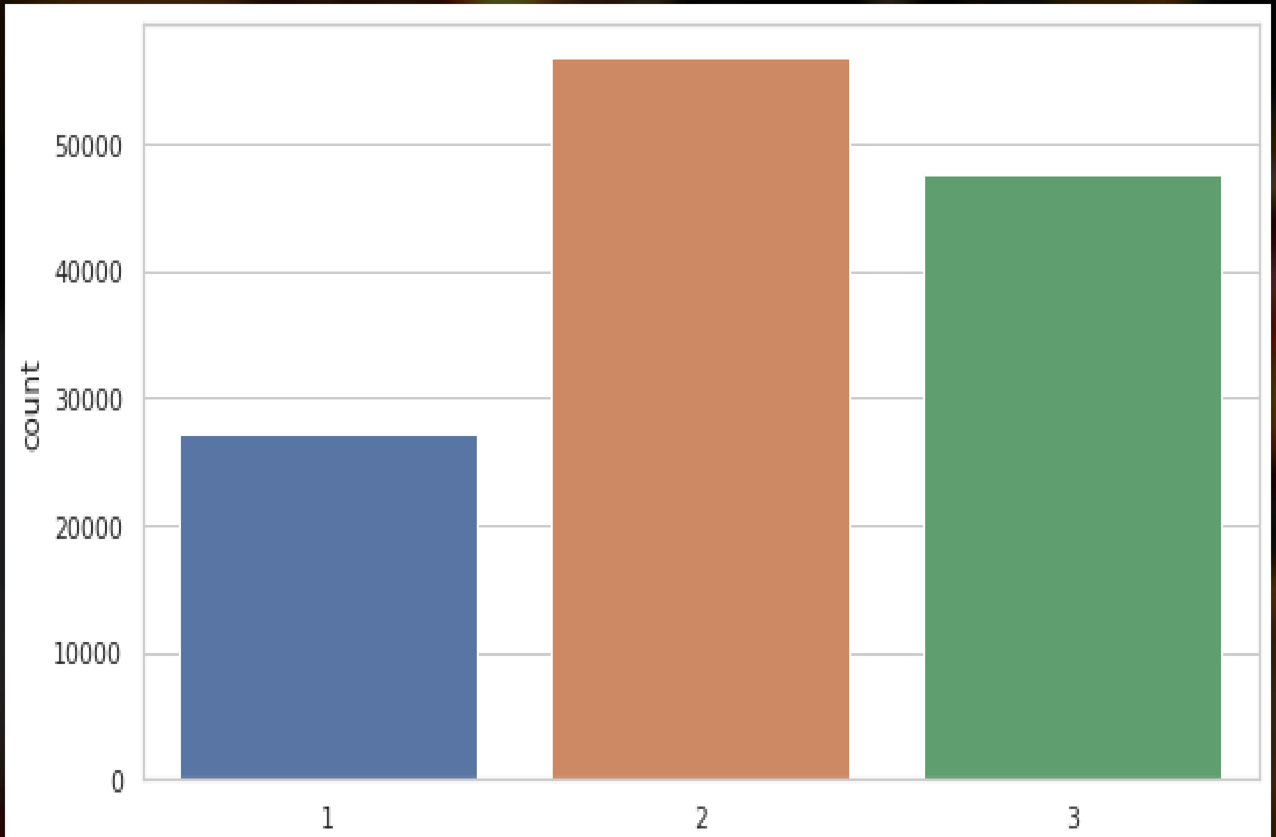
EDA to understand Confidence_Life_Style_Index

Look like the Confidence_Life_Style_Index is randomly assigned with equal distribution. For simpl
assign A,B,C to the NaNs in the field.



EDA on Surge_Pricing_Type

Not a large difference between the target values, and Sampling isn't required.



Modeling: RandomForestClassifier

1. Simple RandomForest gives Accuracy 0.685

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_model = RandomForestClassifier()
```

```
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
predictions_rf = rf_model.predict(X_test)
```

```
accuracy_score=metrics.accuracy_score(y_test, predictions_rf)
accuracy_score
```

```
0.6857175407283637
```

Modeling: XGBoost

1. Simple XGBoost gives Accuracy 0.6835

```
clf.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=27, n_jobs=1,
              nthread=None, num_classes=3, objective='multi:softprob',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=None, subsample=1, verbosity=1)
```

```
pred=clf.predict(X_test)
```

```
pred
```

```
array([2, 1, 2, ..., 2, 2, 3])
```

```
accuracy_score=metrics.accuracy_score(y_test, pred)
accuracy_score
```

```
0.6835149812022937
```

Modeling: XGBoost with GridSearchCV

1. XGBoost with GridSearchCV tuning gives Accuracy 0.696616 on Train data and 0.7015

Double-click (or enter) to edit

```
[ ]  xgb_model = xgb.XGBClassifier()
     optimization_dict = {'max_depth': [2,4,6,None],
                          'n_estimators': [50,100,200,None]}
```

```
[ ]  model = GridSearchCV(xgb_model, optimization_dict,
                          scoring='accuracy', verbose=1)
```

```
[ ]  model.fit(X_train, y_train)
```

```
[ ]  print(model.best_score_)
     print(model.best_params_)
```

```
0.6996933449514483
{'max_depth': 6, 'n_estimators': 200}
```

```
pred=model.predict(X_test)
accuracy_score=metrics.accuracy_score(y_test, pred)
accuracy_score
```