Humana-Mays Healthcare Analytics 2020 Case Competition

Submission By:

Team Quark

Background:

Social determinants of health are the conditions in the environments in which people live, learn, work, play, worship and age that affect a wide range of health, functioning and quality-of-life outcomes and risks. Transportation challenges is one of these determinants.

- Using the data provided and potentially supplementing with public data, create a model to predict which Medicare members are most likely struggling with Transportation Challenges.

- Propose solutions for overcoming these barriers to accessing care and achieving their best health.

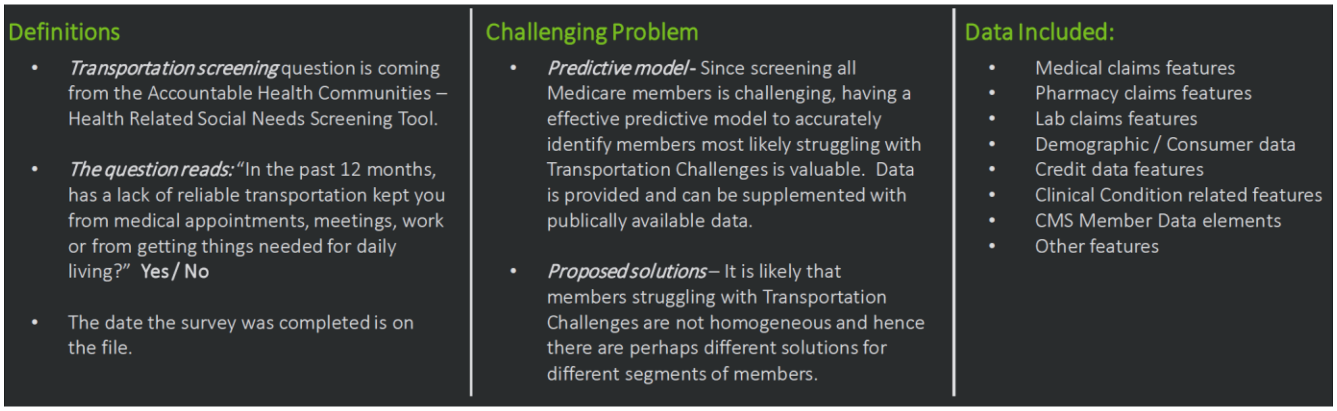
Case Requirements | Key Components:

Problem Statement:

Social determinants of health are the conditions in the environments in which people live, learn, work, play, worship and age that affect a wide range of health, functioning and quality-of-life outcomes and risks. Transportation is one of these determinants. Using the provided data and potentially supplementing with public data, create a model to predict if which members are likely struggling with Transportation.

Goal:

To identify Medicare members most at risk for a Transportation Challenge and propose solutions for them to overcome this barrier to accessing care and achieving their best health.



Definitions

- *Transportation screening* question is coming from the Accountable Health Communities – Health Related Social Needs Screening Tool.
- *The question reads:* "In the past 12 months, has a lack of reliable transportation kept you from medical appointments, meetings, work or from getting things needed for daily living?" **Yes / No**
- The date the survey was completed is on the file.

Challenging Problem

- *Predictive model-* Since screening all Medicare members is challenging, having a effective predictive model to accurately identify members most likely struggling with Transportation Challenges is valuable. Data is provided and can be supplemented with publically available data.
- *Proposed solutions*– It is likely that members struggling with Transportation Challenges are not homogeneous and hence there are perhaps different solutions for different segments of members.

Data Included:

- Medical claims features
- Pharmacy claims features
- Lab claims features
- Demographic / Consumer data
- Credit data features
- Clinical Condition related features
- CMS Member Data elements
- Other features

Data Preparation:

We looked into each feature variable and then grouped it into relevant sets based on the keywords used in naming the variable and the definition for each one of them in the documentation.

Like all the category of features represented as ‘submcc’ anomalies have been grouped based on their common denominator like submcc_bld, submcc_blr, submcc_ano, etc.

The procedure was repeated for many such features and we came up with some 57 groups and the same can be seen in our python notebook file.

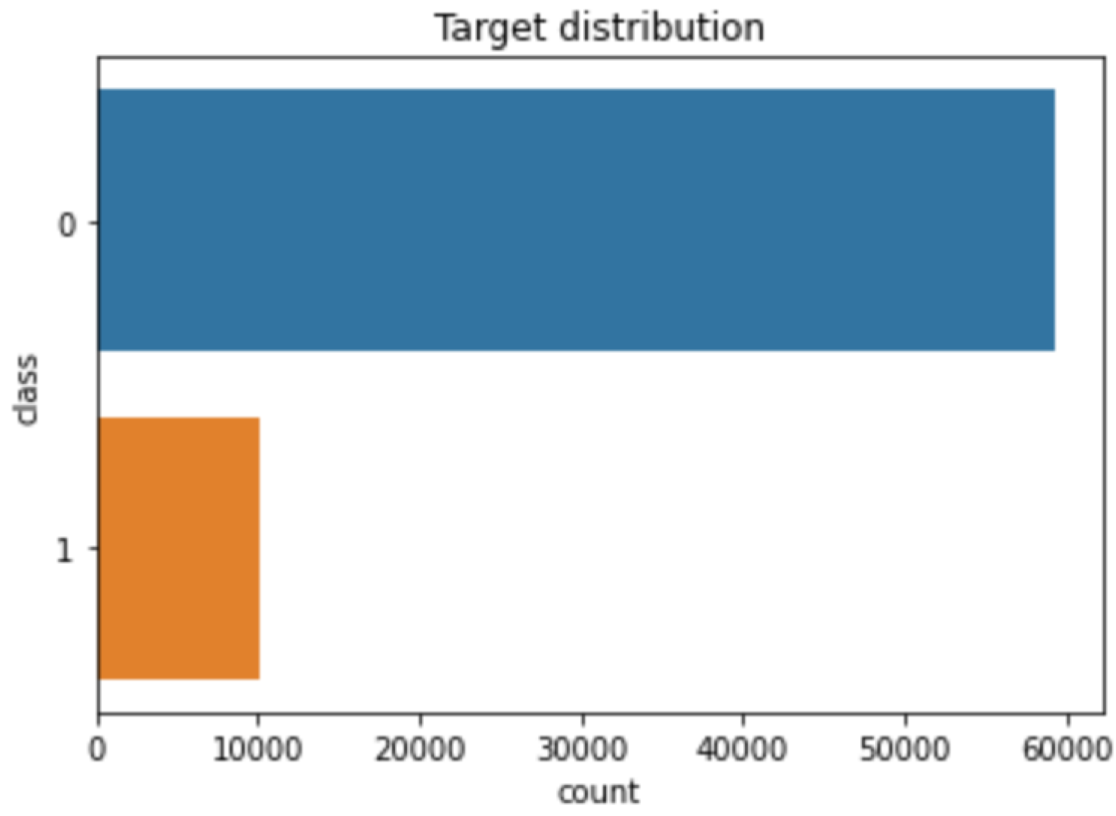
Then we ran multiple models and looked for the accuracy score and the mean error to see how close it is to the probable best fit condition (or line in case of simple regression).

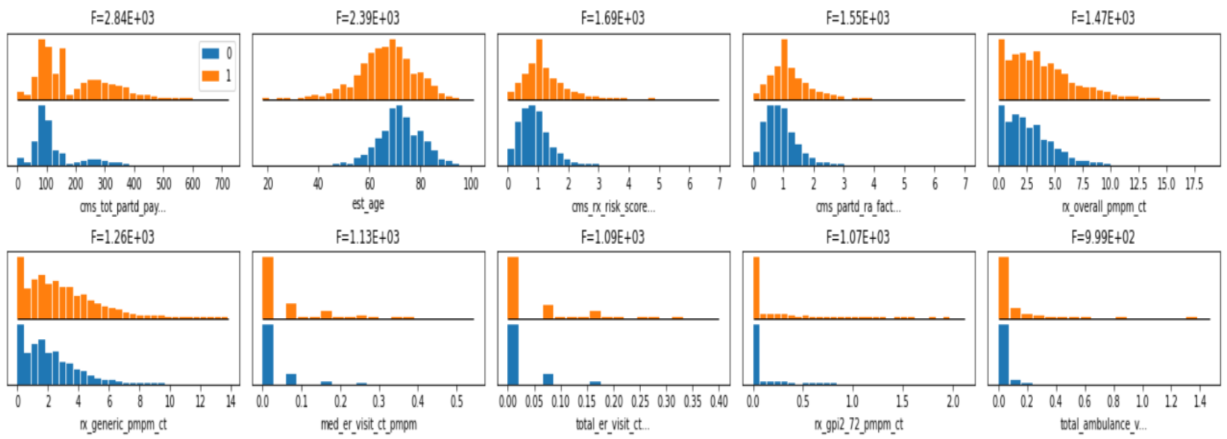
Following are the steps we undertook for the data wrangling part:

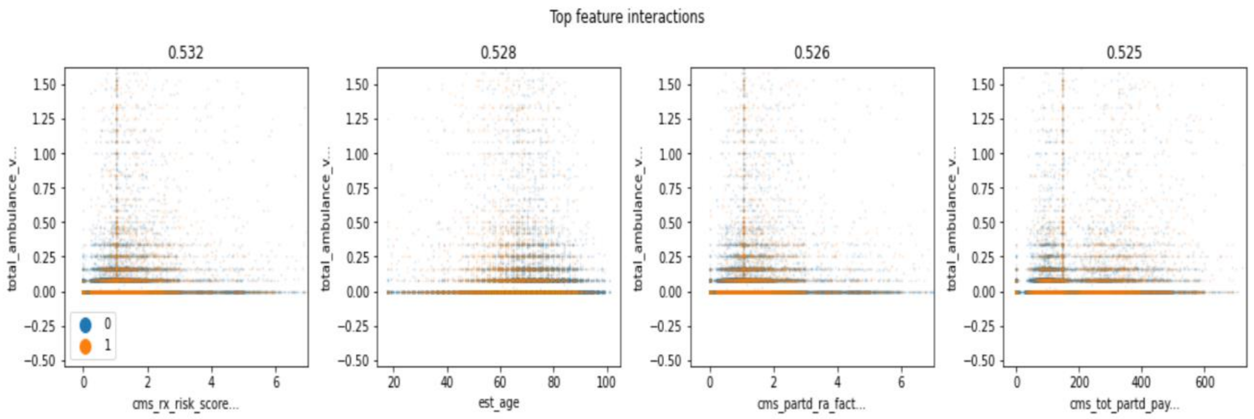
1. For first step we looked at the co-relations between each of the features in the file
2. Then we clean the data to remove all the NAs. We remove any feature which had over 80% of the values as either null or NAs.
3. For the remaining ones, we filled the value with what will a good average representation.

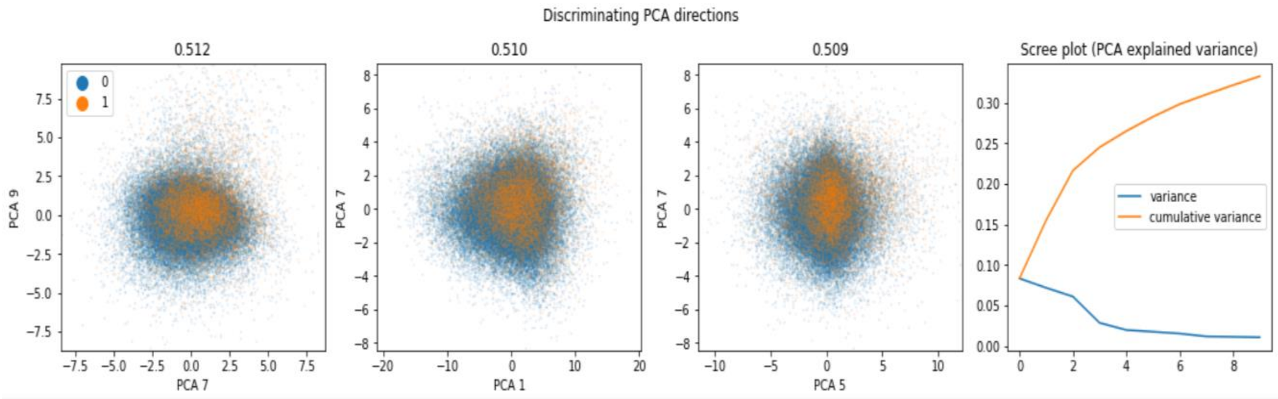
Exploratory Data Analysis:

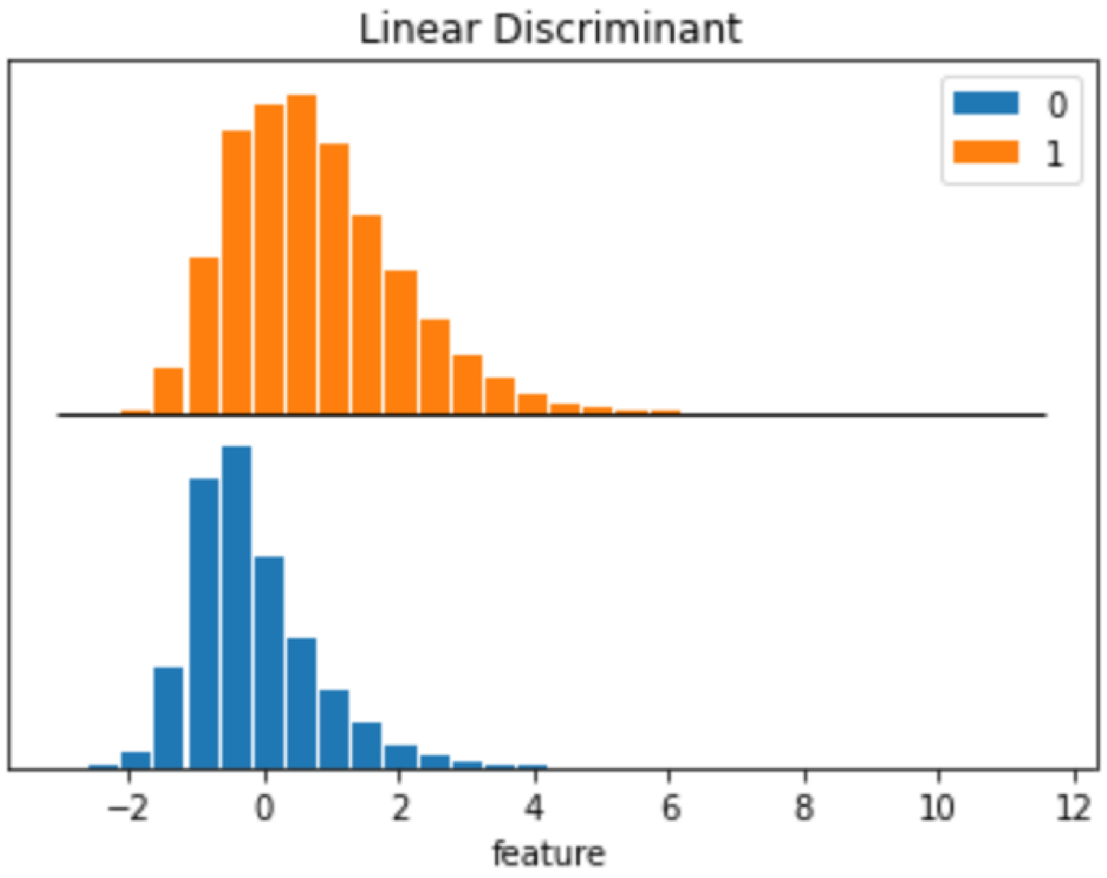
We used the ‘dab;’ package to have a quick understanding of the features v/s our predictor variable which in this case was ‘transportation issues’.

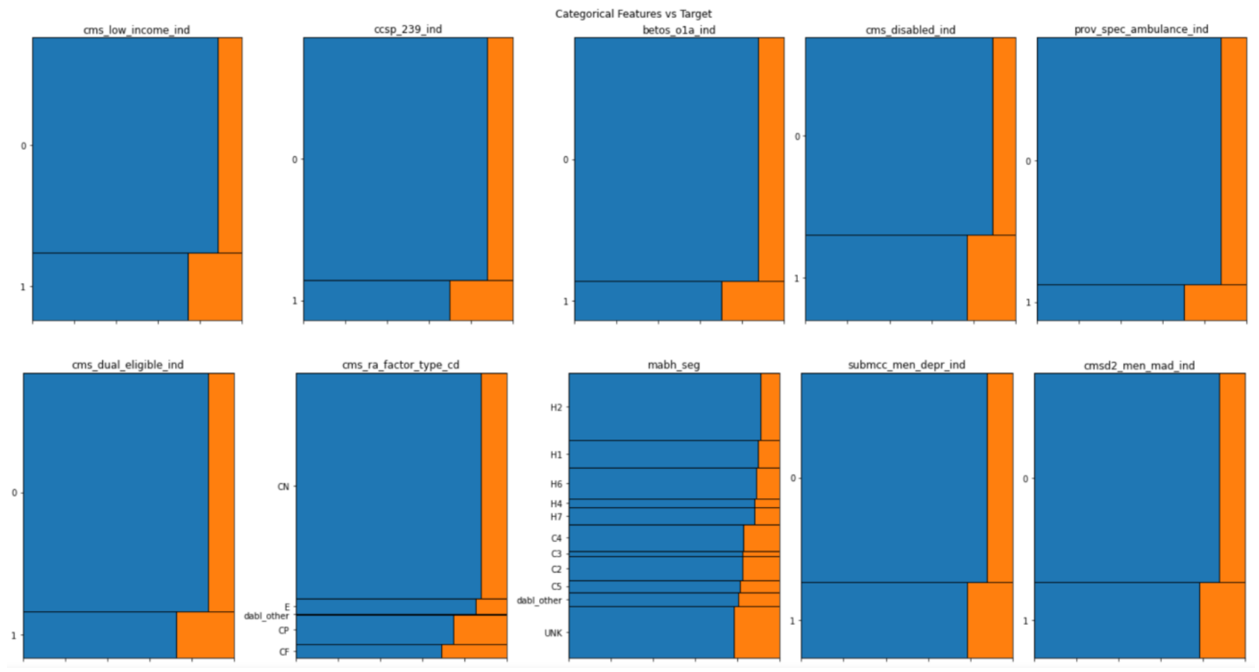





Top feature interactions
0.532
0.528
0.526
0.525
total_ambulance_v...
cms_rx_risk_score...
est_age
cms_partd_ra_fact...
cms_tot_partd_pay...
0
1


Discriminating PCA directions
0.512
0.510
0.509
Scree plot (PCA explained variance)
PCA 9
PCA 7
PCA 1
PCA 5
0
1
variance
cumulative variance


Linear Discriminant
0
1
−2
0
2
4
6
8
10
12
feature

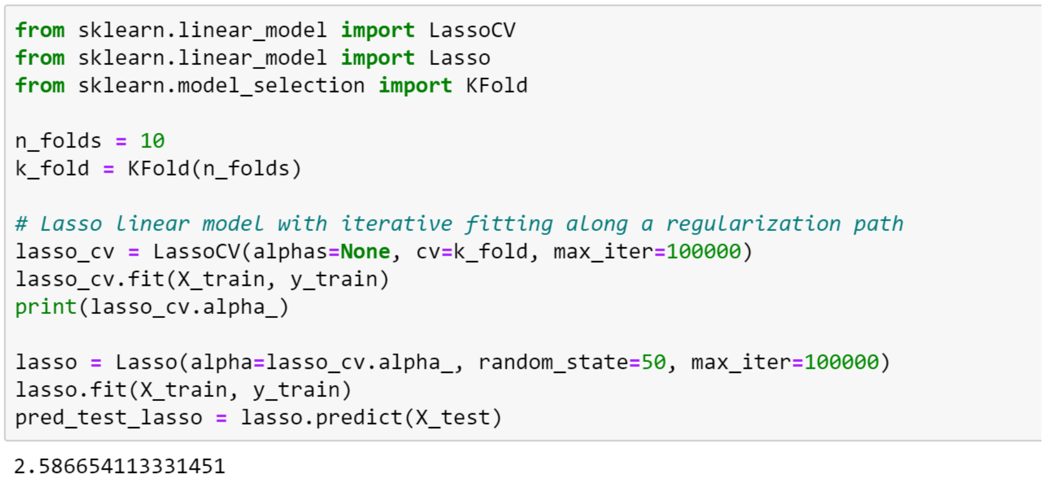

Categorical Features vs Target
cms_low_income_ind
ccsp_239_ind
betos_o1a_ind
cms_disabled_ind
prov_spec_ambulance_ind
cms_dual_eligible_ind
cms_ra_factor_type_cd
mabh_seg
submcc_men_depr_ind
cmsd2_men_mad_ind
0
1
CN
E
dabl_other
CP
CF
H2
H1
H6
H4
H7
C4
C3
C2
C5
dabl_other
UNK

Modeling:

We ran multiple models on the dataset and divided it into random sample of 75% for training set and 25% for test set. Following is the summary of our result:

1. Lasso Regression

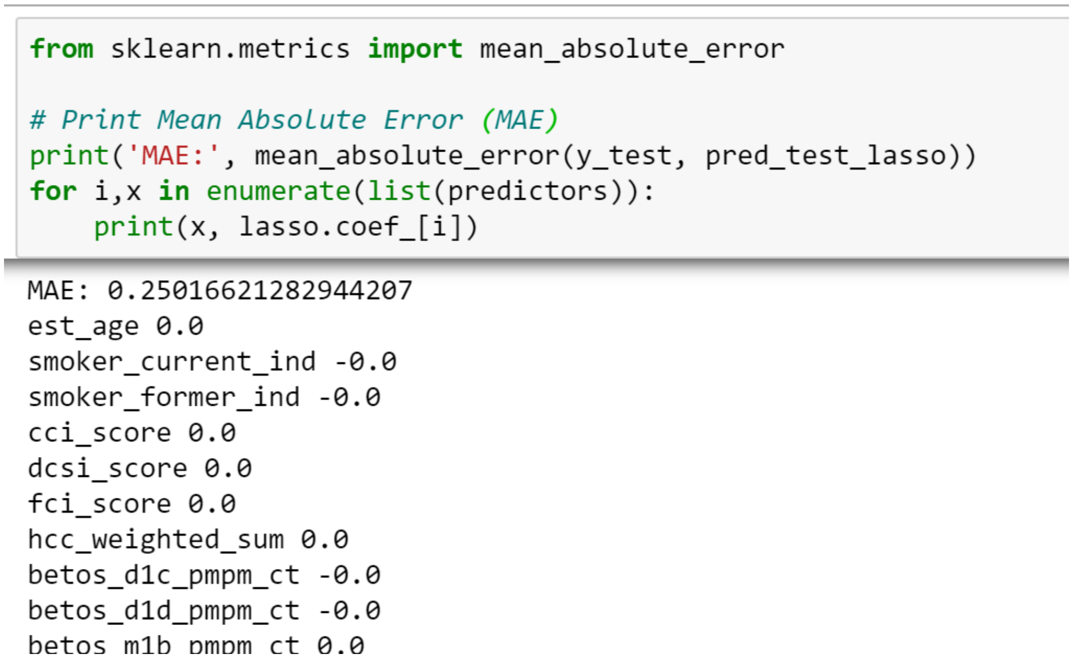
a) The lasso prediction score



```python
from sklearn.linear_model import LassoCV
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold

n_folds = 10
k_fold = KFold(n_folds)

# Lasso linear model with iterative fitting along a regularization path
lasso_cv = LassoCV(alphas=None, cv=k_fold, max_iter=100000)
lasso_cv.fit(X_train, y_train)
print(lasso_cv.alpha_)

lasso = Lasso(alpha=lasso_cv.alpha_, random_state=50, max_iter=100000)
lasso.fit(X_train, y_train)
pred_test_lasso = lasso.predict(X_test)
```

```
2.586654113331451
```

b) The lasso coefficients



```python
from sklearn.metrics import mean_absolute_error

# Print Mean Absolute Error (MAE)
print('MAE:', mean_absolute_error(y_test, pred_test_lasso))
for i,x in enumerate(list(predictors)):
    print(x, lasso.coef_[i])
```

```
MAE: 0.25016621282944207
est_age 0.0
smoker_current_ind -0.0
smoker_former_ind -0.0
cci_score 0.0
dcsi_score 0.0
fci_score 0.0
hcc_weighted_sum 0.0
betos_d1c_pmpm_ct -0.0
betos_d1d_pmpm_ct -0.0
betos_m1b_pmpm_ct 0.0
```

2. Logistic Regression

a) Setting up the model

```
from sklearn.model_selection import train_test_split
X = df.drop ('transportation_issues', axis=1)
Y = df['transportation_issues']
X_train, X_Valid, y_train, y_Valid = train_test_split( X, Y, test_size=0.25, random_state=25, stratify= Y)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Fit the logistic regression on the training set
model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

b) Model evaluation

```
# Model evaluation:
y_predict = model.predict(X_Valid)
```

```
# Accuracy using confusion matrix
confusion_matrix(y_Valid, model.predict(X_Valid))
```

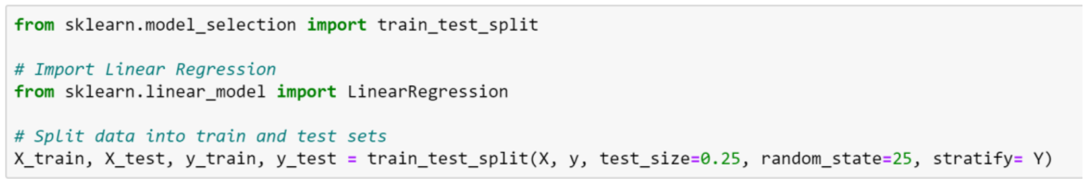
```
array([[14735,   109],
       [ 2442,   107]], dtype=int64)
```

```
# Classification for validation set
print(classification_report(y_Valid, y_predict))
```

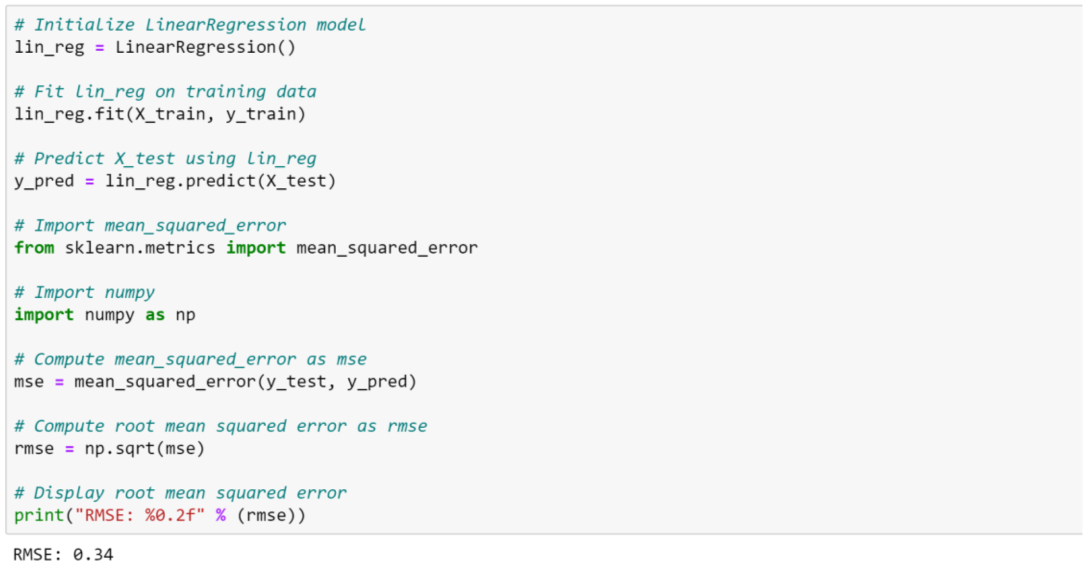
```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92     14844
           1       0.50      0.04      0.08      2549

    accuracy                           0.85     17393
   macro avg       0.68      0.52      0.50     17393
weighted avg       0.80      0.85      0.80     17393
```

3. XGBoost: Regression and Cross Validation

a) Setting up the model and calculating RMSE (Root Mean Square Error)



```
from sklearn.model_selection import train_test_split

# Import Linear Regression
from sklearn.linear_model import LinearRegression

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=25, stratify= Y)
```



```
# Initialize LinearRegression model
lin_reg = LinearRegression()

# Fit lin_reg on training data
lin_reg.fit(X_train, y_train)

# Predict X_test using lin_reg
y_pred = lin_reg.predict(X_test)

# Import mean_squared_error
from sklearn.metrics import mean_squared_error

# Import numpy
import numpy as np

# Compute mean_squared_error as mse
mse = mean_squared_error(y_test, y_pred)

# Compute root mean squared error as rmse
rmse = np.sqrt(mse)

# Display root mean squared error
print("RMSE: %0.2f" % (rmse))
```

```
RMSE: 0.34
```

b) XGBoost Regressor for Linear model

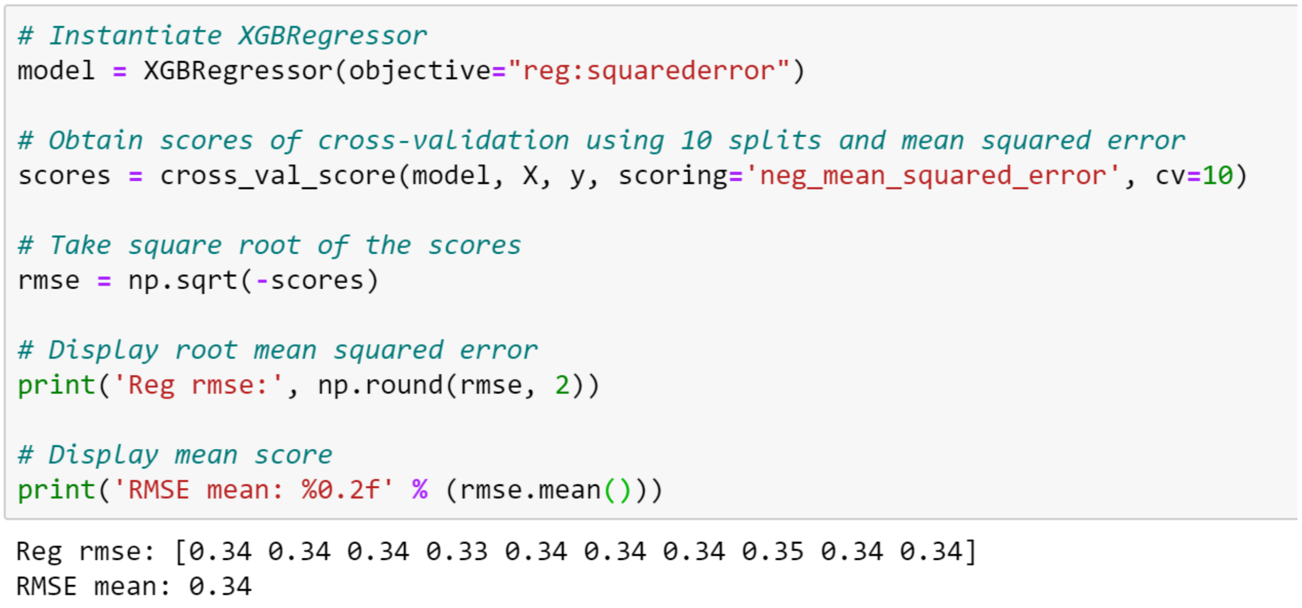
```
# Import XGBRegressor
from xgboost import XGBRegressor

# Instantiate the XGBRegressor, xg_reg
xg_reg = XGBRegressor()

# Fit xg_reg to training set
xg_reg.fit(X_train, y_train)

# Predict labels of test set, y_pred
y_pred = xg_reg.predict(X_test)

# Compute the mean_squared_error, mse
mse = mean_squared_error(y_test, y_pred)

# Compute the root mean squared error, rmse
rmse = np.sqrt(mse)

# Display the root mean squared error
print("RMSE: %0.2f" % (rmse))
```

```
RMSE: 0.34
```

```
from sklearn.model_selection import cross_val_score

# Instantiate Linear Regression
model = LinearRegression()

# Obtain scores of cross-validation using 10 splits and mean squared error
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=10)

# Take square root of the scores
rmse = np.sqrt(-scores)

# Display root mean squared error
print('Reg rmse:', np.round(rmse, 2))

# Display mean score
print('RMSE mean: %0.2f' % (rmse.mean()))
```

```
Reg rmse: [0.33 2.87 0.37 0.33 0.34 0.34 0.34 0.34 0.33 0.33]
RMSE mean: 0.59
```

c) Cross validation score with 10 splits



```
# Instantiate XGBRegressor
model = XGBRegressor(objective="reg:squarederror")

# Obtain scores of cross-validation using 10 splits and mean squared error
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=10)

# Take square root of the scores
rmse = np.sqrt(-scores)

# Display root mean squared error
print('Reg rmse:', np.round(rmse, 2))

# Display mean score
print('RMSE mean: %0.2f' % (rmse.mean()))
```

```
Reg rmse: [0.34 0.34 0.34 0.33 0.34 0.34 0.34 0.35 0.34 0.34]
RMSE mean: 0.34
```

d) Cross validation score

```
# Import cross_val_score
from sklearn.model_selection import cross_val_score

# Define cross_val function with classifer and num_splits as input
def cross_val(classifier, num_splits=10):

    # Initialize classifier
    model = classifier

    # Obtain scores of cross-validation
    scores = cross_val_score(model, X, y, cv=num_splits)

    # Display accuracy
    print('Accuracy:', np.round(scores, 2))

    # Display mean accuracy
    print('Accuracy mean: %0.2f' % (scores.mean()))
```

```
# Use cross_val function to score LogisticRegression
cross_val(LogisticRegression())
```

```
C:\Users\its_t\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\s
iled to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Accuracy: [0.85 0.86 0.86 0.86 0.85 0.85 0.85 0.85 0.85 0.85]
Accuracy mean: 0.85
```

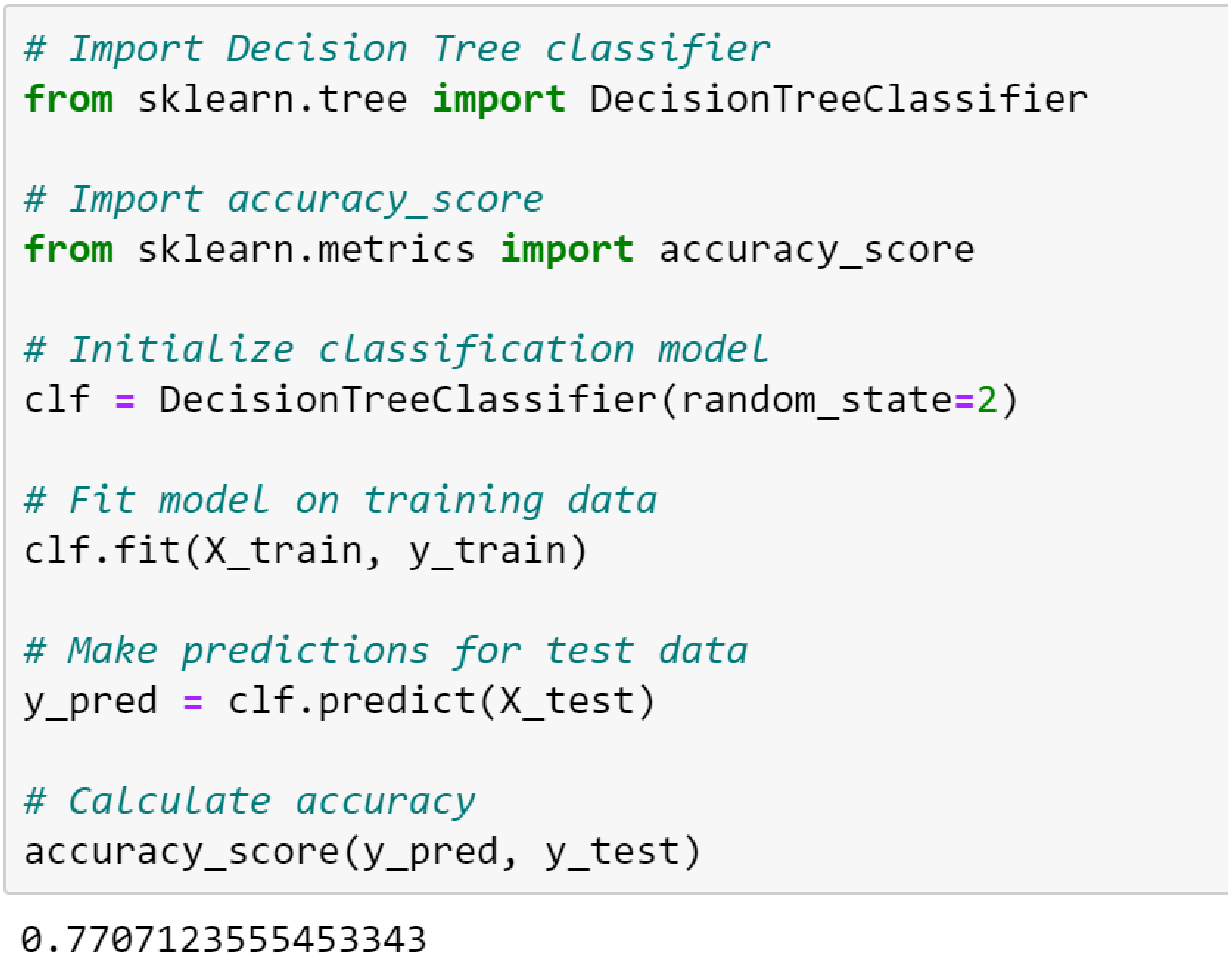
```
from xgboost import XGBClassifier
```

```
# Use cross_val function to score XGBoost
cross_val(XGBClassifier(n_estimators=5))
```

```
Accuracy: [0.86 0.86 0.86 0.86 0.86 0.85 0.86 0.85 0.85 0.86]
Accuracy mean: 0.86
```

4. Decision Trees

a) Setting up the model and getting the accuracy score



```python
# Import Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# Import accuracy_score
from sklearn.metrics import accuracy_score

# Initialize classification model
clf = DecisionTreeClassifier(random_state=2)

# Fit model on training data
clf.fit(X_train, y_train)

# Make predictions for test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy_score(y_pred, y_test)
```

```
0.7707123555453343
```

b) Checking train and test score:

```
# Choose max_depth hyperparameters
params = {'max_depth':[None,2,3,4,6,8,10,20]}

# Initialize regression model as reg
reg = DecisionTreeRegressor(random_state=2)

# Initialize GridSearchCV as grid_reg
grid_reg = GridSearchCV(reg, params, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)

# Fit grid_reg on X_train and y_train
grid_reg.fit(X_train, y_train)

# Extract best parameters
best_params = grid_reg.best_params_

# Print best hyperparameters
print("Best params:", best_params)
```

```
Best params: {'max_depth': 4}
```

```
# Compute best score
best_score = np.sqrt(-grid_reg.best_score_)

# Print best score
print("Training score: {:.3f}".format(best_score))
```

```
Training score: 0.337
```

```
# Extract best model
best_model = grid_reg.best_estimator_

# Predict test set labels
y_pred = best_model.predict(X_test)

# Import mean_squared_error from sklearn.metrics as MSE
from sklearn.metrics import mean_squared_error

# Compute rmse_test
rmse_test = mean_squared_error(y_test, y_pred)**0.5

# Print rmse_test
print('Test score: {:.3f}'.format(rmse_test))
```

```
Test score: 0.337
```

c) Grid search method to find best parameters

```python
# Create grid_search function
def grid_search(params, reg=DecisionTreeRegressor(random_state=2)):

    # Instantiate GridSearchCV as grid_reg
    grid_reg = GridSearchCV(reg, params, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)

    # Fit grid_reg on X_train and y_train
    grid_reg.fit(X_train, y_train)

    # Extract best params
    best_params = grid_reg.best_params_

    # Print best params
    print("Best params:", best_params)

    # Compute best score
    best_score = np.sqrt(-grid_reg.best_score_)

    # Print best score
    print("Training score: {:.3f}".format(best_score))

    # Predict test set labels
    y_pred = grid_reg.predict(X_test)

    # Compute rmse_test
    rmse_test = mean_squared_error(y_test, y_pred)**0.5

    # Print rmse_test
    print('Test score: {:.3f}'.format(rmse_test))
```

```python
grid_search(params={'min_samples_leaf':[1,2,4,6,8,10,20,30]})
```

```
Best params: {'min_samples_leaf': 30}
Training score: 0.366
Test score: 0.366
```

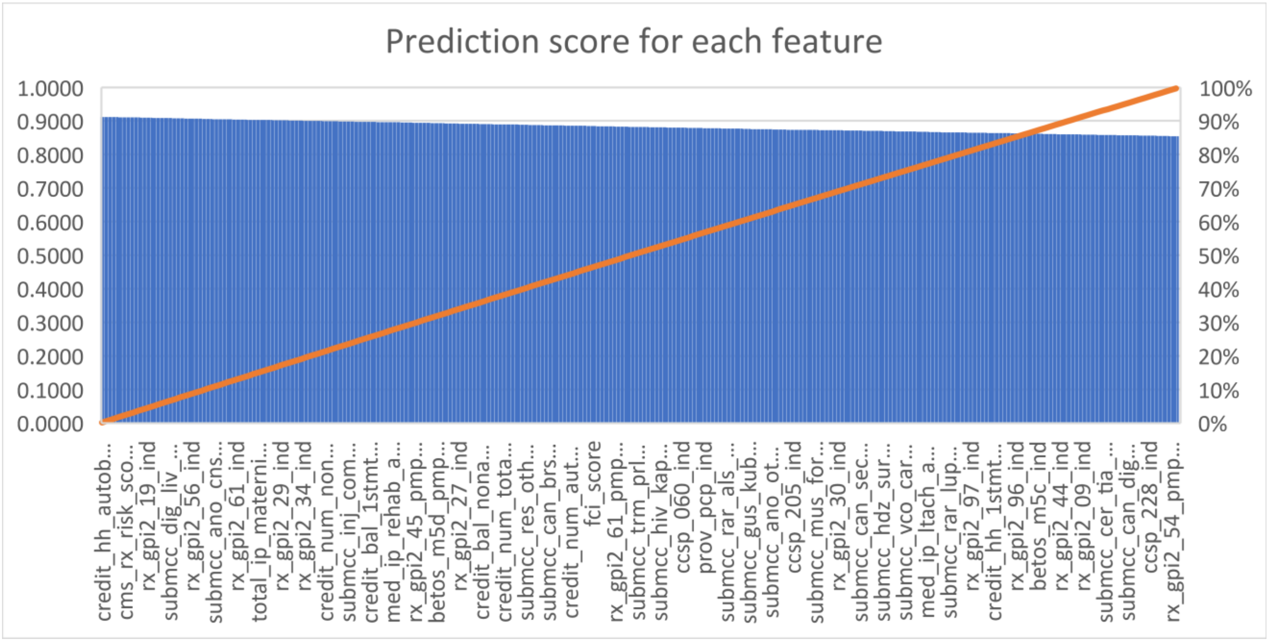
```python
grid_search(params={'max_depth':[None,2,3,4,6,8,10,20],'min_samples_leaf':[1,2,4,6,8,10,20,30]})
```

```
Best params: {'max_depth': 4, 'min_samples_leaf': 30}
Training score: 0.337
Test score: 0.337
```

Final Analysis:

We ran the models for each of those features by group and individually and we found that:

1. Variables that tell about credit balance had a significant affect on the transportation issues.
2. Next came the sub-category of major clinical category
3. Features like BETOS, GPI and HEDIS had a comparatively smaller influence than the above two categories.
4.

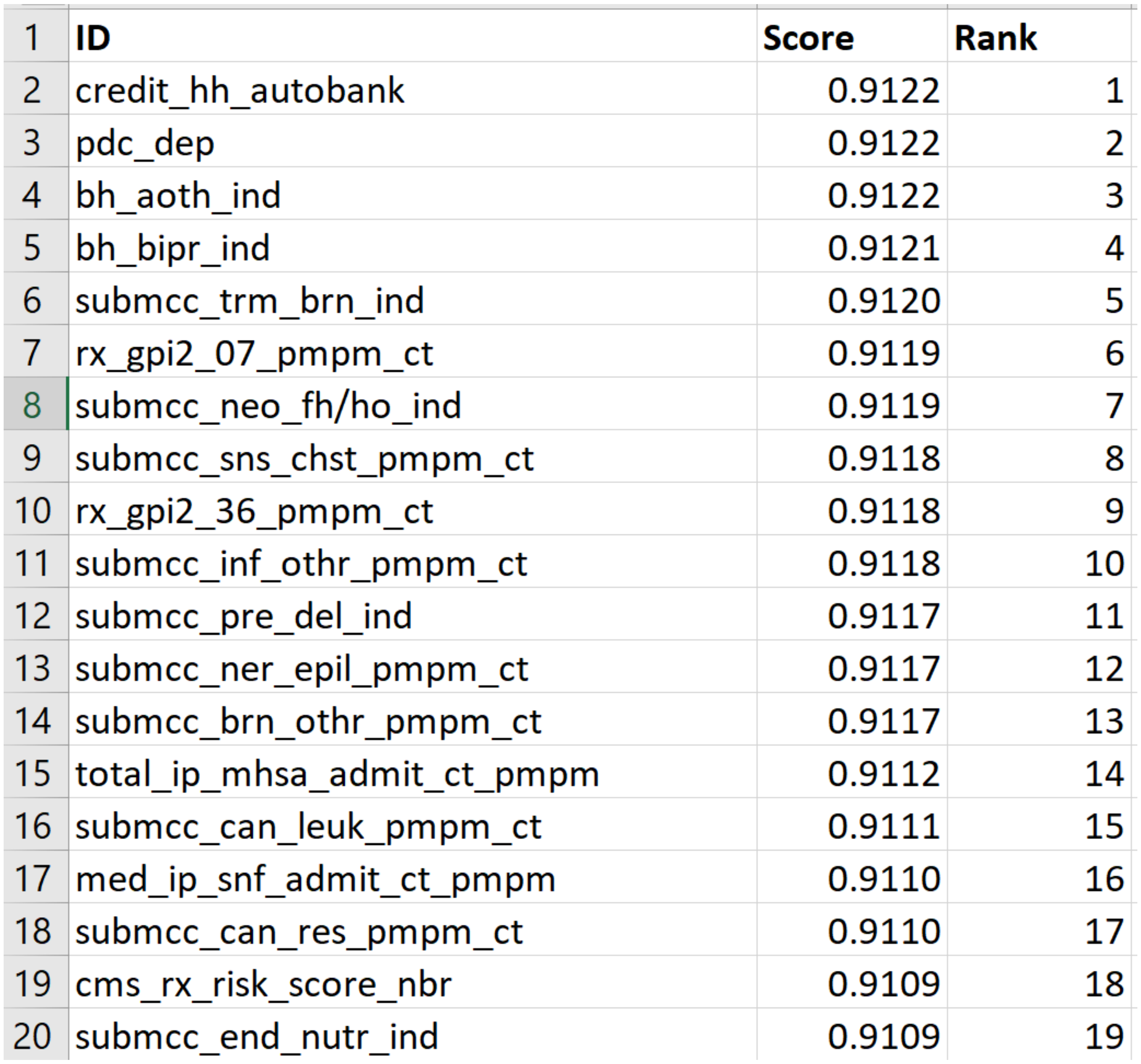


Credit_hh has the highest influence on deciding the transportation issue.

(The overall list with score and rank is shared in the holdout result file)

Top 20 features:

Following is the list of top 20 features that came out with high accuracy from the model.



1	ID	Score	Rank
2	credit_hh_autobank	0.9122	1
3	pdc_dep	0.9122	2
4	bh_aoth_ind	0.9122	3
5	bh_bipr_ind	0.9121	4
6	submcc_trm_brn_ind	0.9120	5
7	rx_gpi2_07_pmpm_ct	0.9119	6
8	submcc_neo_fh/ho_ind	0.9119	7
9	submcc_sns_chst_pmpm_ct	0.9118	8
10	rx_gpi2_36_pmpm_ct	0.9118	9
11	submcc_inf_othr_pmpm_ct	0.9118	10
12	submcc_pre_del_ind	0.9117	11
13	submcc_ner_epil_pmpm_ct	0.9117	12
14	submcc_brn_othr_pmpm_ct	0.9117	13
15	total_ip_mhsa_admit_ct_pmpm	0.9112	14
16	submcc_can_leuk_pmpm_ct	0.9111	15
17	med_ip_snf_admit_ct_pmpm	0.9110	16
18	submcc_can_res_pmpm_ct	0.9110	17
19	cms_rx_risk_score_nbr	0.9109	18
20	submcc_end_nutr_ind	0.9109	19

Key Takeaways:

It seems that transportation is an issue for the majority of the clinical problems while not so much services codes, or behavioral health concern.

The bank balance or financial standing of the patients also had a major say in terms of transportation issue.

We also categorized the data based on age and following were our sets:

1. Patients between 18 and 30 who can be attributed to youth.
2. Patients between 31 and 40 who can be attributed to mid-life.
3. Patients between 41 and 60 who can be attributed to senior.
4. Patients above 60 who can be attributed to veteran.

We observed as expectedly that patients in the higher age group i.e., veteran seem to have a bigger issue with transportation issue.

We must analyze categories which have higher influence in the social determinant and try to look for reasons behind them but interacting with the right audience and take necessary steps to bridge the gap.