MGMT-571
KAGGLE COMPETITION PRESENTATION

Team Data Drillers

Our Team: Data Drillers

Mohinder Goyal

Maharshi Dutta

Zaid Ahmed

Talent wins games, but teamwork and intelligence win championships.

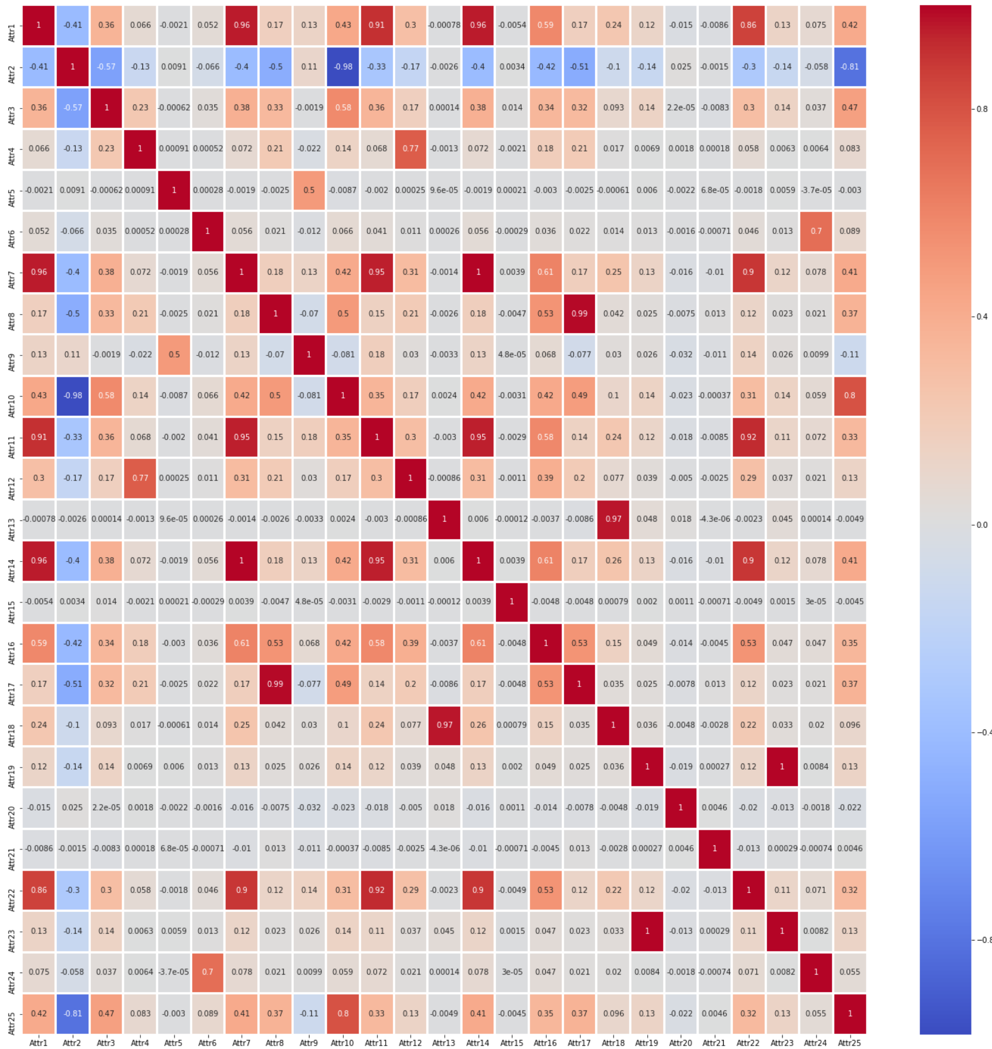
-Michael Jordan

COMPETITION BACKGROUND:

- **Problem Statement:** To develop a predictive model that combines various econometric measures to foresee a financial condition (Bankruptcy or not) of a firm.
- **Data Description:**
  - Training Data- 10000 observations
  - Test Data – 5000 observations

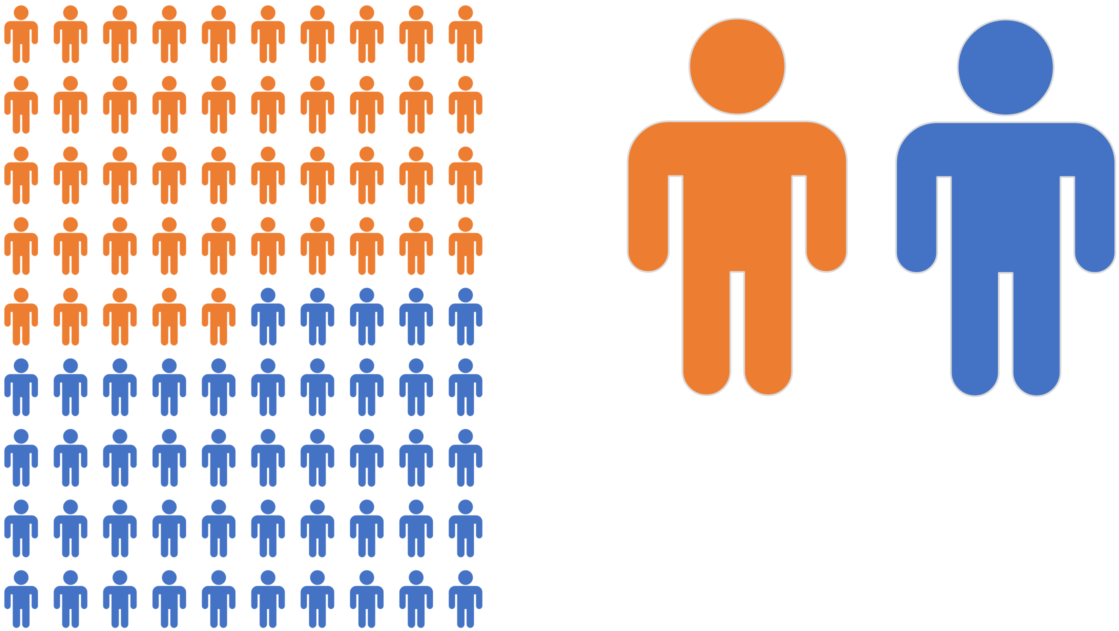
DATA DESCRIPTION:

- 64 Attributes
- High correlation was expected



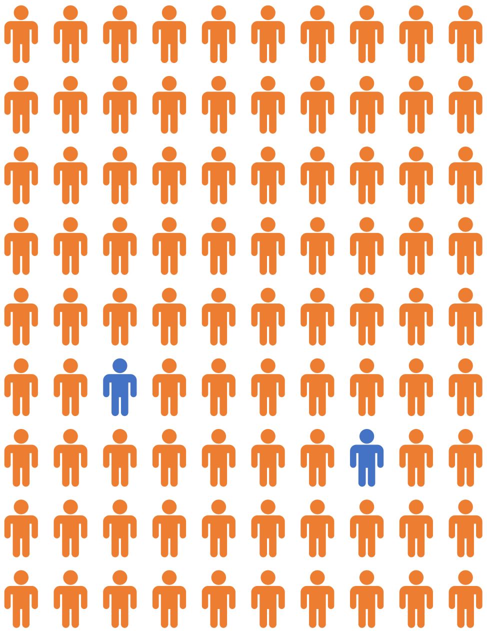
Team: Data Drillers

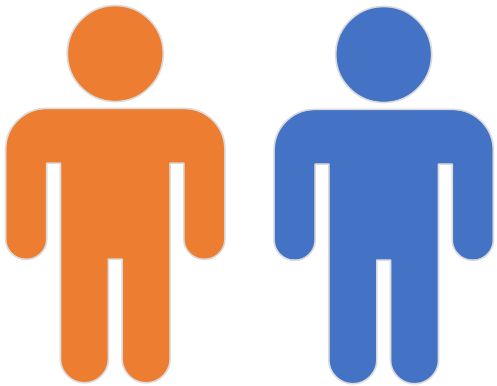
EDA: Imbalance Data



EDA: Imbalance Data

Problem: The Train Data provided to us was highly skewed.





>99%   <1%

*Source: [Data Provided]

EDA: Row Duplication

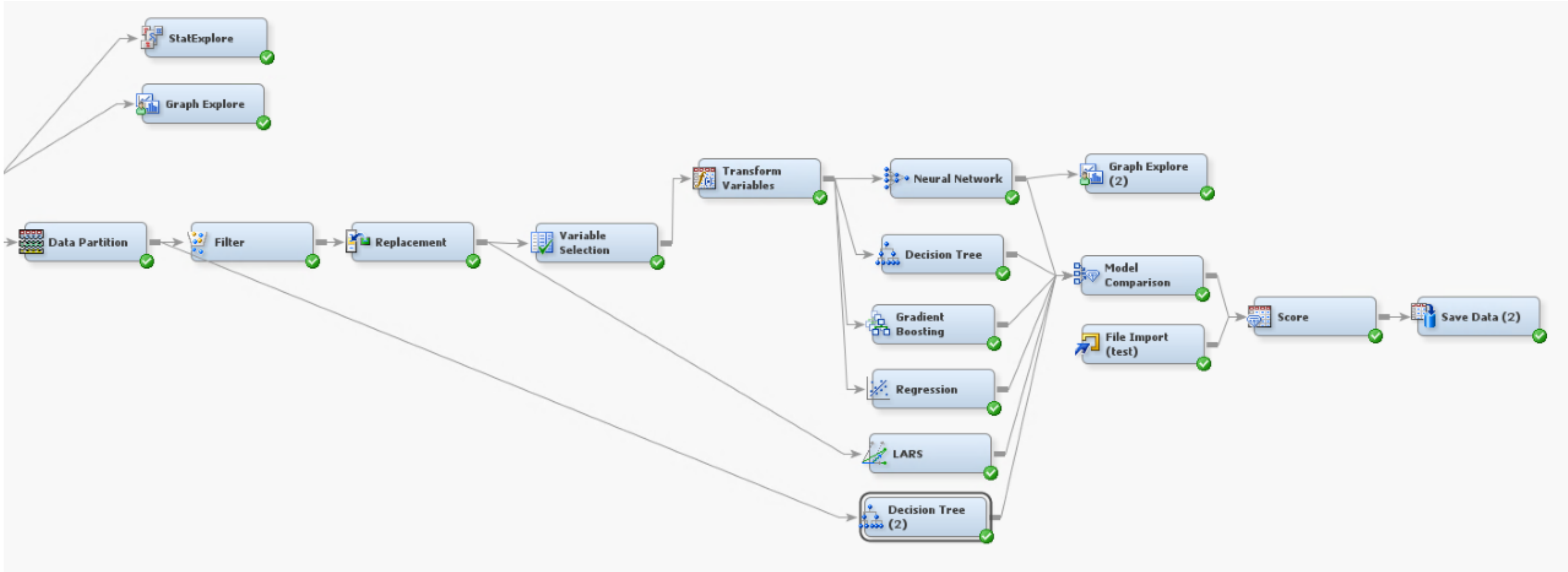
- Surprisingly there were multiple duplication of rows in our train data set.
- We used drop_duplicate to clean up the data.

MODELS USED:

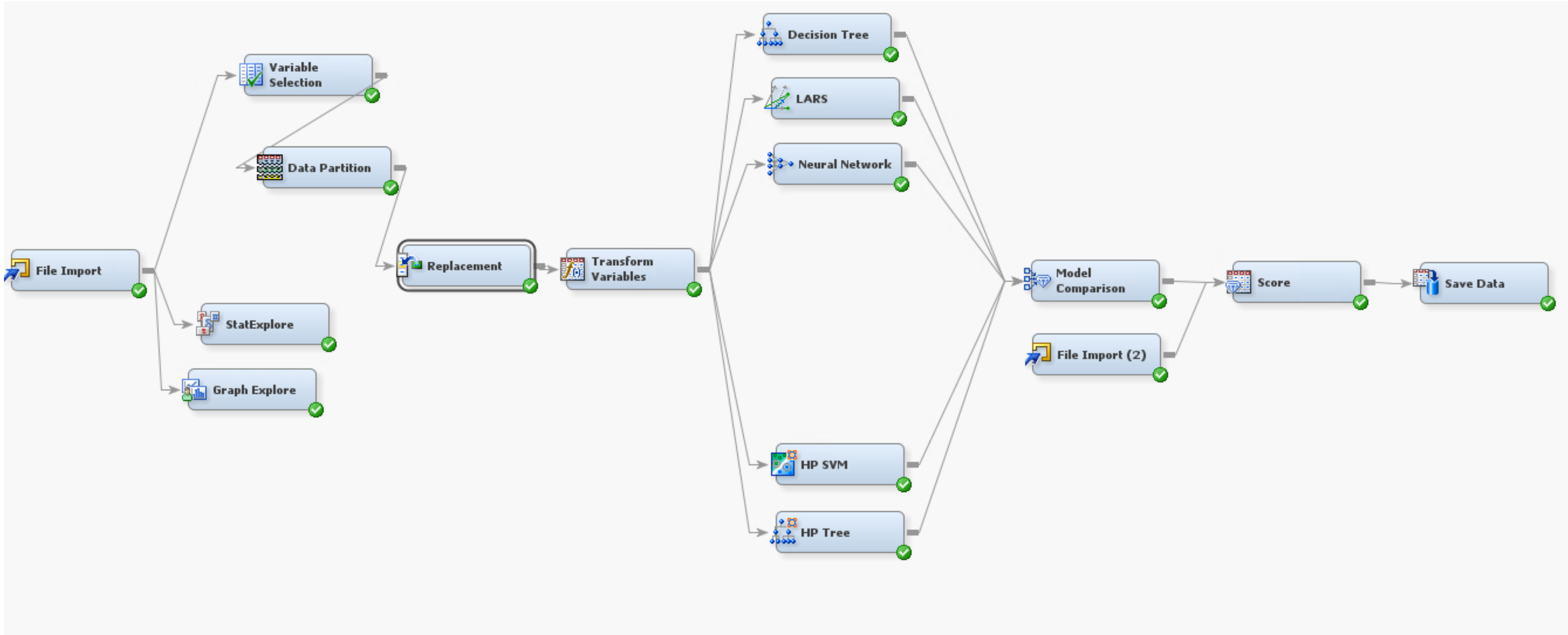
01

Enterprise Miner

- Neural Networks
- Decision Tree
- Gradient Boosting
- Logistic Regression
- LARS - Lasso
- Decision Tree
- LARS- Adaptive Lasso
- HP SVM
- HP Tree



ENTERPRISE MINER - 1



ENTERPRISE MINER - 2

MODELS USED:

01

Enterprise Miner

- Neural Networks
- Decision Tree
- Gradient Boosting
- Logistic Regression
- LARS - Lasso
- Decision Tree
- LARS- Adaptive Lasso
- HP SVM
- HP Tree

02

R-Studio: H2O

- Deep Learning
- Random Forest
- Gradient Boosting
- H20- Auto ML

```r
rf <- h2o.randomForest(x, y, train)    #Random Forest
gbm<- h2o.gbm(x,y,train, nfolds=4)     #Gradient Boosting
dl <- h2o.deeplearning(x, y, train)    #Deep Learning
```

```r
# make predictions
p <- h2o.predict(dl, bc_test)
p2 <- h2o.predict(rf, bc_test)
p3<- h2o.predict(gbm,bc_test)
```

```r
###########################################################################
# train with AutoML - specify how long you are willing to wait
auto <- h2o.automl(x, y, train, max_runtime_secs=3600, keep_cross_validation_predictions=TRUE,
                   nfolds = 5, balance_classes = TRUE)
automl<- h2o.predict(auto, bc_test)
automl<-as.data.frame(automl)
###########################################################################
```

R-Studio: H20

MODELS USED:

01 Enterprise Miner

- Neural Networks
- Decision Tree
- Gradient Boosting
- Logistic Regression
- LARS - Lasso
- Decision Tree
- LARS- Adaptive Lasso
- HP SVM
- HP Tree

02 R-Studio: H2O

- Deep Learning
- Random Forest
- Gradient Boosting
- H20- Auto ML

03 Python: AutoML & PCA

- Auto ML
- PCA

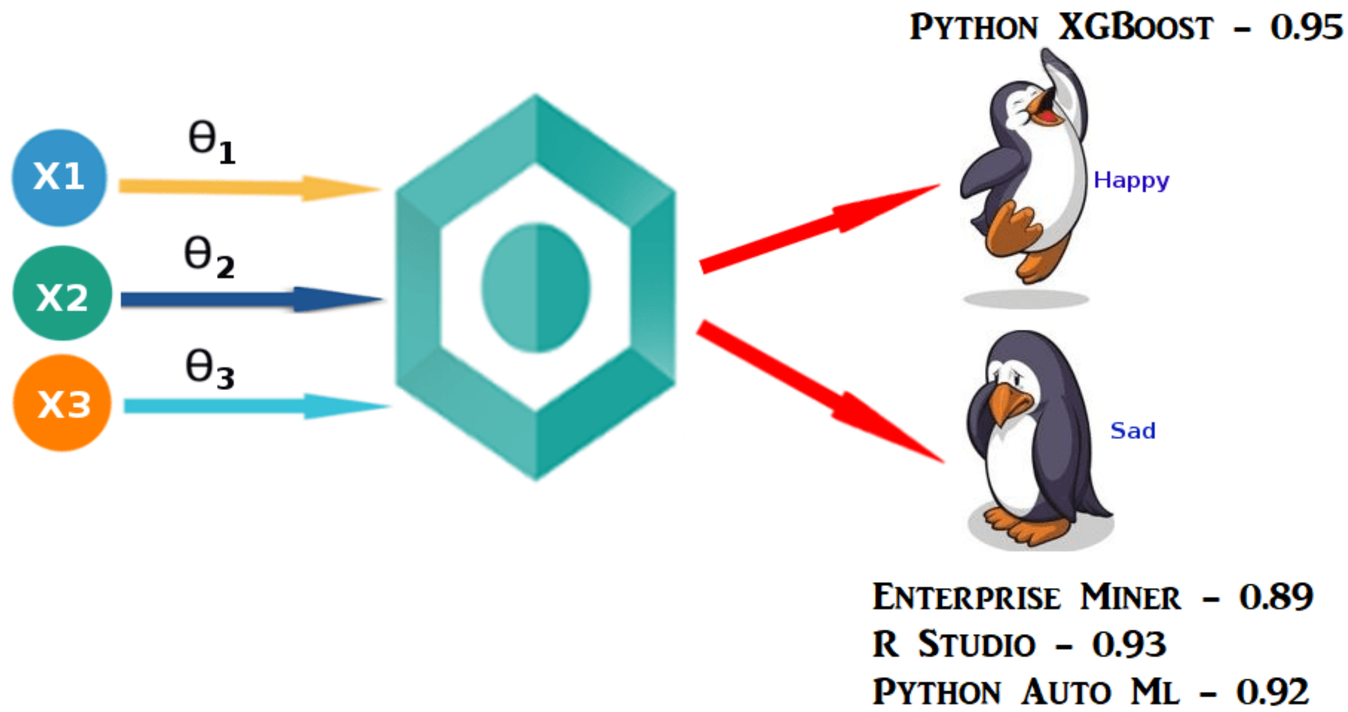
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
from sklearn.decomposition import PCA

pca = PCA(n_components = 30)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

```python
column_descriptions = {'class': 'output'}

ml_predictor = Predictor(type_of_estimator='classifier', column_descriptions=column_descriptions)

ml_predictor.train(df_train)

ml_predictor.score(df_test, df_test.Output)
```

Python: AutoML + PCA

Result Summary:



MODELS USED:

01 Enterprise Miner

- Neural Networks
- Decision Tree
- Gradient Boosting
- Logistic Regression
- LARS - Lasso
- Decision Tree
- LARS- Adaptive Lasso
- HP SVM
- HP Tree

02 R-Studio: H2O

- Deep Learning
- Random Forest
- Gradient Boosting
- H20- Auto ML

03 Python: AutoML & PCA

- Auto ML
- PCA

USED Python

- XGBoost
- SMOTE
- Hyperparameter Tuning

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
from sklearn.model_selection import KFold
from imblearn.over_sampling import SMOTE
K = 5
kf = KFold(n_splits = K, random_state = 3228, shuffle = True)

smote = SMOTE(ratio='minority')
X_train, y_train = smote.fit_sample(X_train, y_train)

X_train = pd.DataFrame(X_train)
y_train = pd.DataFrame(y_train)
y_train.rename(columns={0:"class"}, inplace=True)
```

```python
def modelfit(alg, dtrain, predictors,useTrainCV=True, cv_folds=5, early_stopping_rounds=50):

    if useTrainCV:
        xgb_param = alg.get_xgb_params()
        xgtrain = xgb.DMatrix(dtrain[predictors].values, label=dtrain[target].values)
        cvresult = xgb.cv(xgb_param, xgtrain, num_boost_round=alg.get_params()['n_estimators'], nfold=cv_folds,
            metrics='auc', early_stopping_rounds=early_stopping_rounds)
        alg.set_params(n_estimators=cvresult.shape[0])

    #Fit the algorithm on the data
    alg.fit(dtrain[predictors], dtrain['class'],eval_metric='auc')

    #Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors])
    dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]

    #Print model report:
    print ("\nModel Report")
    print ("Accuracy : %.4g" % metrics.accuracy_score(dtrain['class'].values, dtrain_predictions))
    print ("AUC Score (Train): %f" % metrics.roc_auc_score(dtrain['class'], dtrain_predprob))
```

Python: XGBoost + SMOTE + Hyperparameter

```
predictors = [x for x in train.columns if x not in [target, IDcol]]
```

```
xgb2 = XGBClassifier(
     learning_rate =0.07,
     n_estimators=4000,
     max_depth=6,
     min_child_weight=1,
     gamma=0,
     subsample=0.8,
     colsample_bytree=0.9,
     objective= 'binary:logistic',
     nthread=4,
     scale_pos_weight=1,
     seed=27)
print(modelfit(xgb2, train, predictors))

predictions = xgb2.predict(X_test)
#print(i)
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))

print("Classification Report")
print(classification_report(y_test, predictions))

prob_y_5 = xgb2.predict_proba(X_test)
prob_y_5 = [p[1] for p in prob_y_5]
print( roc_auc_score(y_test, prob_y_5))
```

Python: XGBoost + SMOTE + Hyperparameter

Output of Final Model

```
Model Report
Accuracy : 1
AUC Score (Train): 1.000000
None
Confusion Matrix:
[[1959    9]
 [  18   14]]
Classification Report
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1968
           1       0.61      0.44      0.51        32

    accuracy                           0.99      2000
   macro avg       0.80      0.72      0.75      2000
weighted avg       0.98      0.99      0.99      2000

0.9582063008130082
```

ahmed152@purdue.edu

THANK YOU!

Do you have any feedback or suggestions?

Appendix:

References:

- https://www.analyticsvidhya.com
- https://towardsdatascience.com
- https://elitedatascience.com