Suhas Patil

BANK DATA ANALYSIS FOR MARKETING CAMPAIGN

1.1 BACKGROUND

A leading Portuguese Banking Institution wants to analyze its previous campaigns data in order to determine whether a client will subscribe to its term deposit scheme or not. These marketing campaigns were mainly based on phone calls.

1.2 OBJECTIVES

- The main objective of this project is to effectively group potential clients for future campaign targets
- The other aim is to reduce the cost of marketing campaigns and focus on the areas of improvements
- To apply and suggest a best model for classification and prediction.

1.3 DATASET OVERVIEW

Source - http://archive.ics.uci.edu/ml/datasets/Bank+Marketing
Total Records – 41188
Variables- 21
The variables can be further divided into following different categories:

Bank Client Data –

1. **Age** (numeric) - Age of the client
2. **job** (categorical) - Type of occupation generating income for the client
3. **marital** (categorical) - Marital status of the client
4. **education** (categorical) – Level of highest education
5. **default** (categorical) – Whether the client has credit in default?
6. **Housing** (categorical) - Whether the client has any housing loan?
7. **Loan** (categorical) - Whether the client has any personal loan?
8. **Contact** (categorical) - Contact communication type
9. **Month** (categorical) - Last contact month of year
10. **day_of_week** (categorical) - Last contact day of the week
11. **duration** (numeric) - Last contact duration, in seconds
    (This attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model)

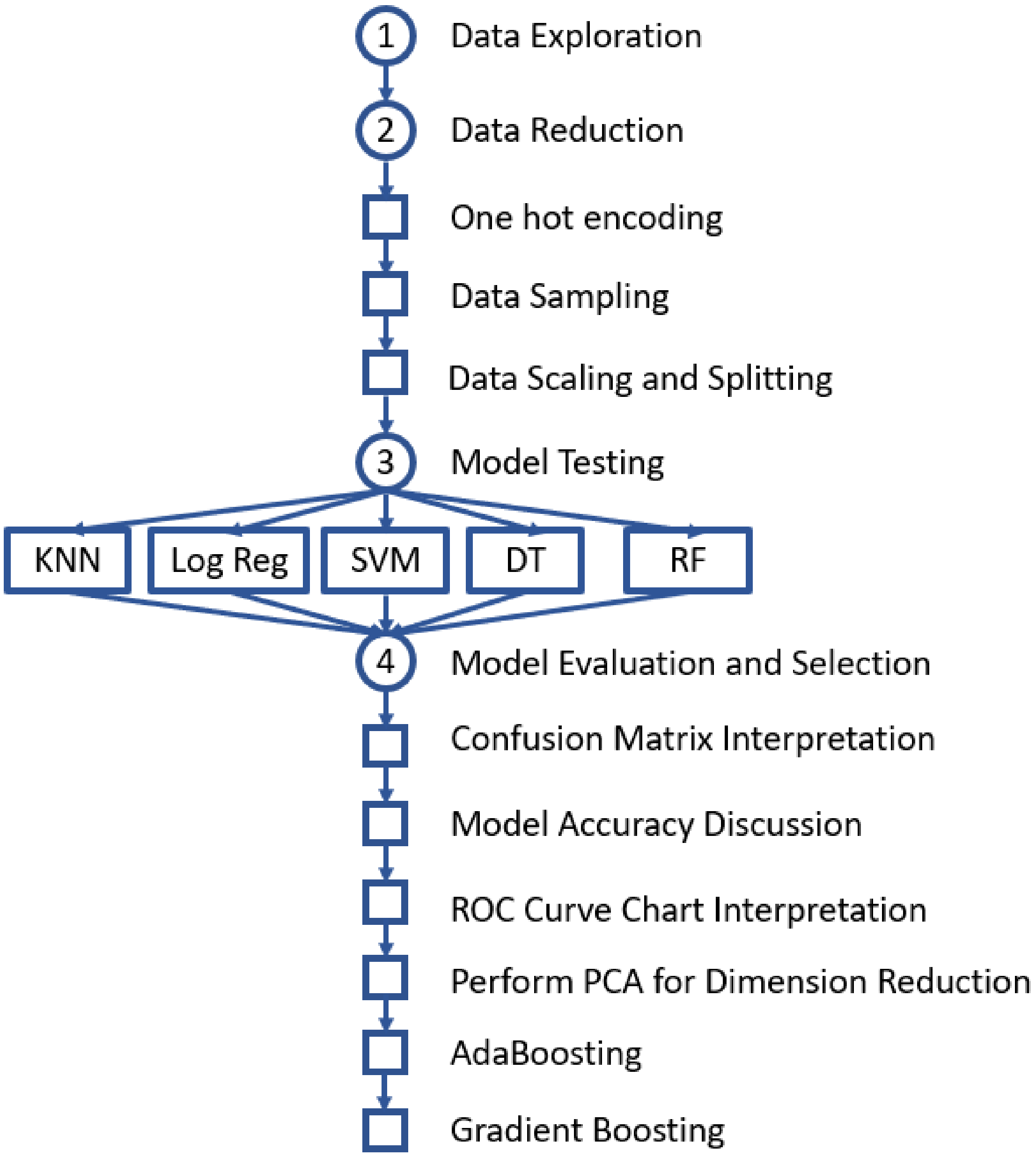
Social and economic context attributes –

1. **emp.var.rate** (numeric) - Employment variation rate - quarterly indicator
2. **cons.price.idx** (numeric) - Consumer price index - monthly indicator
3. **cons.conf.idx** (numeric) - Consumer confidence index - monthly indicator
4. **euribor3m** (numeric) - Euribor 3-month rate - daily indicator
5. **nr.employed** (numeric) - Number of employees - quarterly indicator

Other attributes –

1. **campaign** (numeric) - Number of contacts performed during this campaign for a client
2. **pdays** (numeric) - Number of days that passed by after the client was last contacted from a previous campaign (999 means client was not previously contacted)
3. **previous** (numeric) - Number of contacts performed before this campaign for this client
4. **poutcome** (categorical) - Outcome of the previous marketing campaign
5. **y** - Has the client subscribed a term deposit? (Output Variable)

1.4 PROCESS



1
Data Exploration
2
Data Reduction
One hot encoding
Data Sampling
Data Scaling and Splitting
3
Model Testing
KNN
Log Reg
SVM
DT
RF
4
Model Evaluation and Selection
Confusion Matrix Interpretation
Model Accuracy Discussion
ROC Curve Chart Interpretation
Perform PCA for Dimension Reduction
AdaBoosting
Gradient Boosting

1.5 DATA EXPLORATION

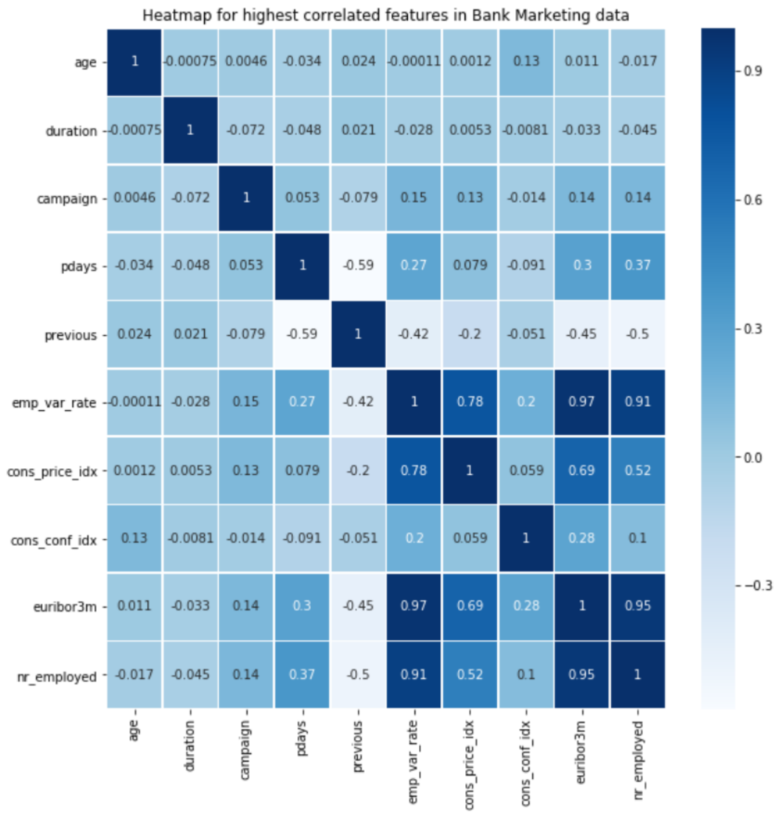
- The dataset has 41188 records with 21 variables
- Drop all the columns with null values
- Numeric variables can be described as

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

- Categorical variables can be described as

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	y
count	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188
unique	12	4	8	3	3	3	2	10	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	thu	nonexistent	no
freq	10422	24928	12168	32588	21576	33950	26144	13769	8623	35563	36548

- Rename social and economic context attributes to maintain the consistency across all column names
- There are 12 duplicate rows which we will drop for further smooth analysis
- To determine the correlation between variables we will generate correlation matrix



-

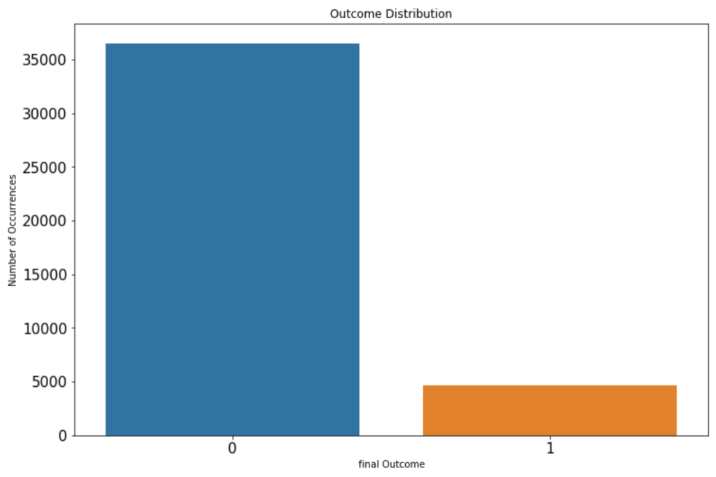
- From above correlation heatmap we can say that

1. **euribor3m** is highly correlated with **emp-var-rate** with ρ = 0.9722
2. **nr_employed** is highly correlated with **euribor3m** with ρ = 0.9451

- Hence, we will not include these 3 columns for further analysis

IMPORTANT VARIABLE ANALYSIS

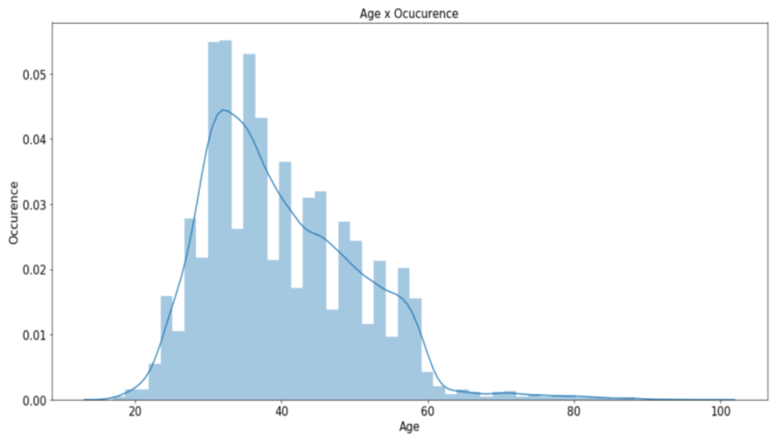
1. Y – Output Variable



This variable indicates whether a client has subscribed to term deposit or not.

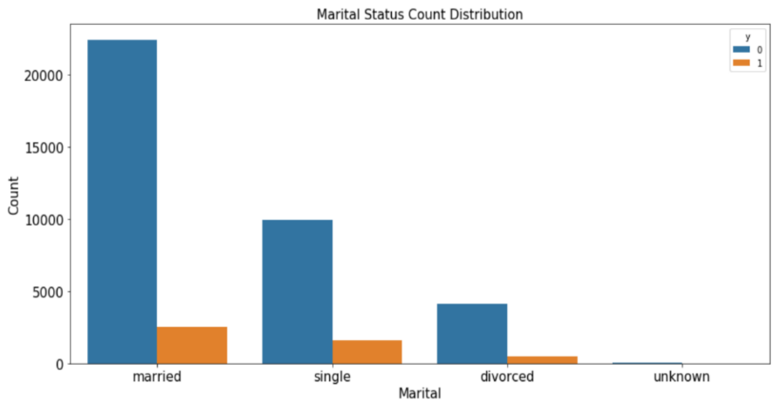
The data set is highly imbalanced. There are 6 times as many 'No' cases than 'Yes' cases. Therefore, accuracy alone as the evaluation metric won't cut it. Since False Positives and False Negatives both cost out marketing campaign to a certain degree, we will consider F1 – Score to be our primary evaluation metric.

2. Age



Adjacent distribution shows clients in the age range **30-50Yrs** forms a major portion and can be targeted clients for further analysis

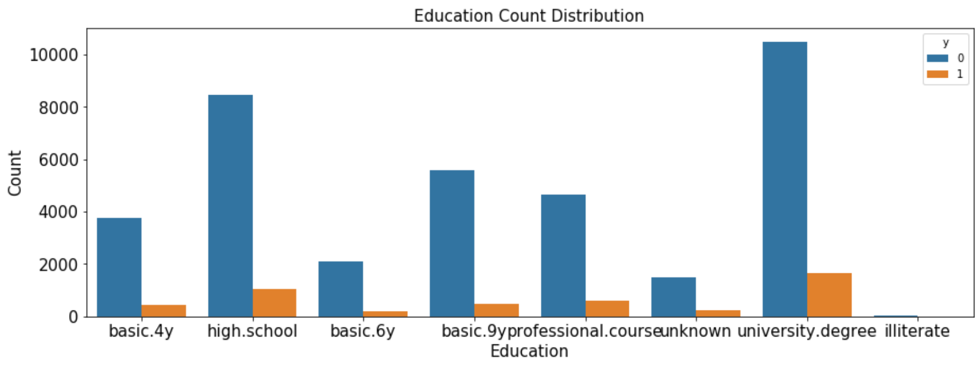
3. Marital Status



Most of the clients are **Married** and can be targeted for the next marketing campaign. We can also focus on the single ones as they account for second highest customers

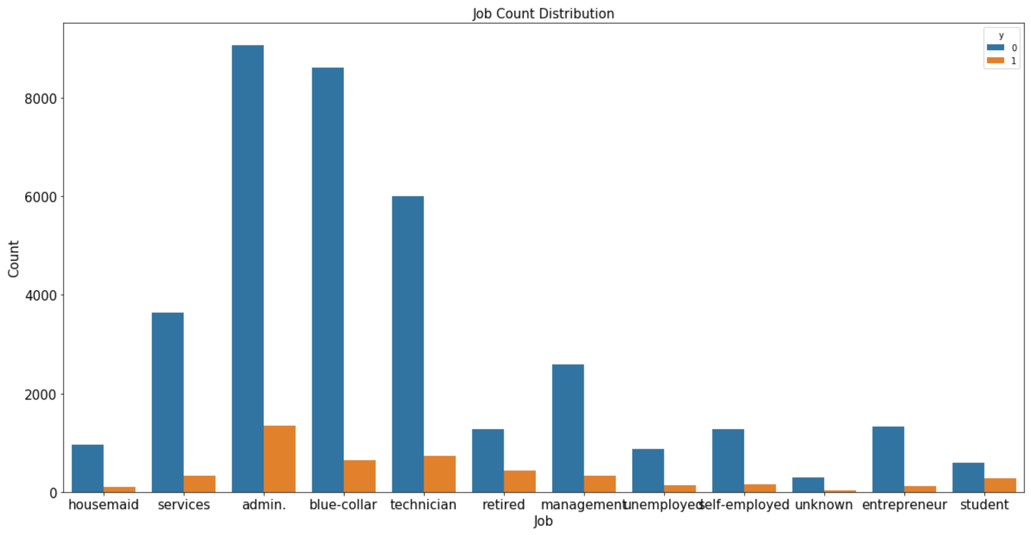
4. Education

Clients with **University Degree** and **High School** graduated comprises majority of clients if targeted by their education

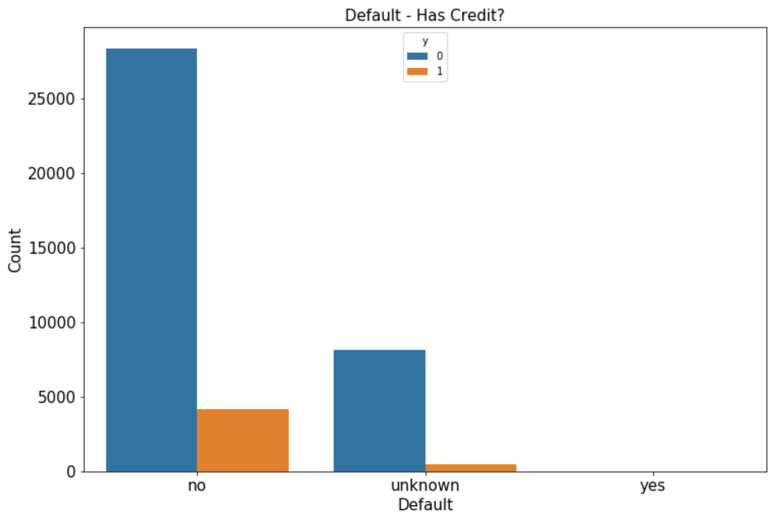


5. Occupation

People into **Administrative** jobs are highest clients among all other occupations. WE can also target Blue-collar people as they count almost equal to admin people



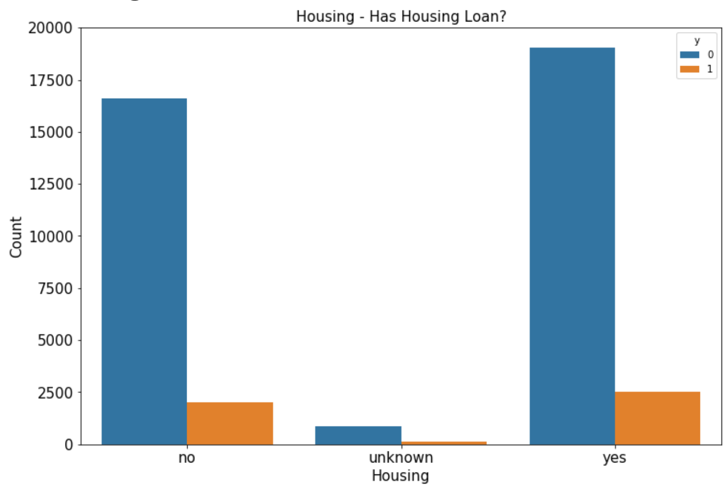
6. Default



Most of the clients are not defaulted to previous payments but still they did not agree to open term deposit in last campaign

About 69% of clients are not defaulted to previous payments

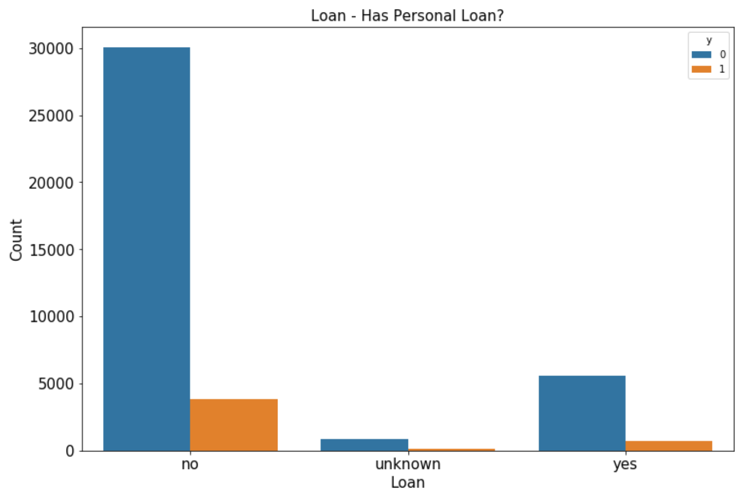
7. Housing



There is no much difference in the percentage of clients with or without any housing loan.

We have 45% of total clients without any housing loan. The status for few clients is not clear so we have included them in the remaining 55% of population with housing loan for further analysis

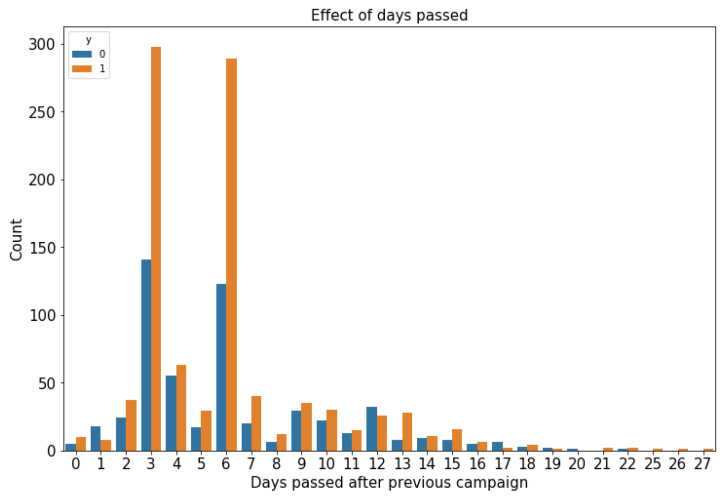
8. Loan



Majority of the clients do not have any Personal loan so they can be targeted as they maintain a good payment history.

Here we cannot neglect the individual with unknown personal loan status so we will keep them as it is and will focus more on 82% people without any personal loan.

9. pdays



Elapsed number of days last contacted	Count of Clients
Not contacted (>999 Days)	39649
0 to 10 Days	1311
11 to 20 Days	196
21 to 27 Days	8

For the above graph we have not included the customers for who were contacted before. This variable represents the number of days passed by after the client was last contacted from a previous campaign.

2. DATA REDUCTION & PRE-PROCESSING

2.1. DROPPING COLUMNS

'euribor3m', 'nr_employed' and 'duration' columns are dropped to handle multicollinearity. Multicollinearity is a situation where an independent variable exhibits high correlation with other independent variables. In such cases, the inferences made about the data are seldom sound and reliable. Therefore, we have decided to drop the three columns that are highly correlated.

2.2. ONE HOT ENCODING

We are performing one hot encoding on 'default', 'job', 'marital', 'housing', 'loan', 'contact', 'month', 'day_of_week', ’poutcome’ as we assume them to categorical values with levels that have no inherent internal superiority over the other levels of the same categorical variables. We are left with a dataset of 41164 rows and 53 columns after the one hot encoding.

```
bank_marketing.shape
```

```
(41164, 53)
```

2.3. PRE-PROCESSING

Sampling:

```
bank_marketing = pd.DataFrame.sample(bank_marketing,frac=0.3,random_state=0)
bank_marketing.shape
```

```
(12349, 53)
```

We then sampled data to just consider 30% of the whole data to speed up our model fitting process.

The next step is to determine the target variable from our data and separating it out from the independent variables.

Features and Target:

```
y = bank_marketing['y']
X = bank_marketing.drop(['y'], axis = 1)
```

The data is then split into train and test sets.

```
X_train_org, X_test_org, y_train, y_test = train_test_split(X, y, random_state = 0)
```

Scaling is an essential step before fitting models as it brings all our features onto the same scale. We used MinMax Scaler on our data.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_org)
X_test = scaler.transform(X_test_org)
```

2.4. FEATURE ENGINEERING

We performed Principal Component Analysis on our data to find the number of features required to cover 95% of explained variance. This reduced the count of our features from 53 to 25.

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
X_train_reduced_Bf = pca.fit_transform(X_train)
pca.n_components_
```

```
25
```

```
pca = PCA(n_components = pca.n_components_)
X_train_reduced_Bf = pca.fit_transform(X_train)
X_test_reduced_Bf = pca.transform(X_test)
print(X_train_reduced_Bf.shape)
print(X_test_reduced_Bf.shape)
```

```
(9261, 25)
(3088, 25)
```

Since the dataset is highly imbalanced with very few observations pertaining to the positive outcome, we have balanced the weights for equal representation of classes in our dataset used for building the models.

```
models = {
        "KNN-Classifier": KNeighborsClassifier(),
         "Logistic Reg": LogisticRegression(random_state=0, class_weight='balanced'),
        "LinearSVC": LinearSVC(random_state=0,class_weight='balanced'),
        "SVM Kernelized": SVC(random_state=0,probability = True,class_weight='balanced'),
        "Decision Tree": DecisionTreeClassifier(random_state=0,class_weight='balanced'),
         "Random Forest": RandomForestClassifier(random_state=0,class_weight='balanced'),
        }
```

3. MODEL BUILDING & EVALUATION

We first built a dummy classifier on our data to get an idea of the baseline performance of any classifier on our data.

Dummy Classifier

```
from sklearn.dummy import DummyClassifier
dc = DummyClassifier()
dc.strategy = 'most_frequent'
dc.fit(X_train,y_train)
dc.score(X_train,y_train)

y_pr = pd.DataFrame(y_test)
y_pr = y_pr.replace([1],[0])

skm.confusion_matrix(y_test, dc.predict(X_test),labels=[1,0])
```

```
array([[   0,  347],
       [   0, 2741]], dtype=int64)
```

From the above confusion matrix we can calculate the accuracy of our baseline model to be 88.45%. This is the accuracy we can expect if we are simply guessing the outcome based on our data. So, ideally we would want our models to achieve an accuracy greater than 88.45%. But, accuracy is not the only measure we will be concerned with. We will also be looking at Precision, Recall and their harmonic mean – F1 Score.

We have built Logistic Regression, KNN Classifier, Linear SVC, Kernelized SVM, Decision Tree and Random Forest on the unreduced dataset.

3.1. MODELS ON UNREDUCED DATASET

3.1.1. LOGISTIC REGRESSION

Logistic Regression is one of the most popular ways to fit categorical data into models. It is the most used among the generalized linear models and works extremely well on data with binary responses. It predicts the probability of a categorical dependent variable belonging to one of the 2 classes.

```
Model: Logistic Reg
Parameter(Optimum): {'C': 1, 'solver': 'lbfgs'}
Cross-validation score: 0.45
```

Confusion Matrix for Logistic Regression –

```
[[ 220  127]
 [ 449 2292]]
```

We decreased the predict_proba threshold from 50%, which is the default, to 25% . This essentially means the model only labels an instance as ‘1’ when the predicted probability of it being ‘1’ is more than 25%. After this post processing, our results are as follows –

Confusion Matrix –

```
[[ 313   34]
 [1917  824]]
```

Classification Report –

```
              precision    recall  f1-score   support

           0       0.96      0.30      0.46      2741
           1       0.14      0.90      0.24       347

    accuracy                           0.37      3088
   macro avg       0.55      0.60      0.35      3088
weighted avg       0.87      0.37      0.43      3088
```

Logistic Regression on PCA reduced dataset –

```
Model: Logistic Reg
Parameter(Optimum): {'C': 25, 'solver': 'lbfgs'}
Cross-validation score: 0.42
```

Confusion Matrix –

```
[[ 222  125]
 [ 488 2253]]
```

Confusion Matrix on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
[[ 320   27]
 [2028  713]]
```

Classification Report on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
              precision    recall  f1-score   support

           0       0.96      0.26      0.41      2741
           1       0.14      0.92      0.24       347

    accuracy                           0.33      3088
   macro avg       0.55      0.59      0.32      3088
weighted avg       0.87      0.33      0.39      3088
```

3.1.2. KNN CLASSIFIER

K- Nearest Neighbors or also known as K-NN belong to the family of supervised machine learning algorithms which means we use labeled (Target Variable) dataset to predict the class of new data point. The K-NN algorithm is a robust classifier which is often used as a benchmark for more complex classifiers such as Artificial Neural Network (ANN) or Support vector machine (SVM).

K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.

```
Model: KNN-Classifier
Parameter(Optimum): {'n_neighbors': 3}
Cross-validation score: 0.30
```

Confusion Matrix for KNN Classifier –

```
[[  77  270]
 [ 111 2630]]
```

We again decreased the predict_proba threshold from 50% to 25%, the results after post processing are as follows.

Confusion Matrix –

```
[[ 177  170]
 [ 582 2159]]
```

Classification Result –

```
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      2741
           1       0.23      0.51      0.32       347

    accuracy                           0.76      3088
   macro avg       0.58      0.65      0.59      3088
weighted avg       0.85      0.76      0.79      3088
```

KNN Classifier on PCA reduced dataset –

```
Model: KNN-Classifier
Parameter(Optimum): {'n_neighbors': 3}
Cross-validation score: 0.31
```

Confusion Matrix –

```
[[  88  259]
 [ 119 2622]]
```

Confusion Matrix on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
[[ 186  161]
 [ 619 2122]]
```

Classification Report on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2741
           1       0.23      0.54      0.32       347

    accuracy                           0.75      3088
   macro avg       0.58      0.66      0.58      3088
weighted avg       0.85      0.75      0.79      3088
```

3.1.3. LINEAR SVC

The purpose of Linear SVC is to return a best fit hyperplane that divides our data. In the Support Vector machine implementation, it finds the points closest to the separating line/hyper plane which are the support vectors. The goal then is to maximize the margin which is the distance between the line and support vectors.

```
Model: LinearSVC
Parameter(Optimum): {'C': 10}
Cross-validation score: 0.49
```

Confusion Matrix –

```
[[ 204  143]
 [ 347 2394]]
```

Post Processing –

```
y_train_prob = best_model.decision_function(X_train)

y_train_prob_new = best_model.decision_function(X_test)
y_pred_new = np.where(y_train_prob_new > 0.7, 1, 0)
print(skm.confusion_matrix(y_test, y_pred_new,labels=[1,0]))
print(skm.classification_report(y_test, y_pred_new))
```

Confusion Matrix after Post Processing –

```
[[  93  254]
 [  63 2678]]
```

Classification Report after Post Processing –

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      2741
           1       0.60      0.27      0.37       347

    accuracy                           0.90      3088
   macro avg       0.75      0.62      0.66      3088
weighted avg       0.88      0.90      0.88      3088
```

Linear SVC on PCA reduced dataset –

```
Model: LinearSVC
Parameter(Optimum): {'C': 10}
Cross-validation score: 0.42
```

Confusion Matrix –

```
[[ 219  128]
 [ 466 2275]]
```

Confusion Matrix on PCA reduced data after post processing -

```
[[  84  263]
 [  82 2659]]
```

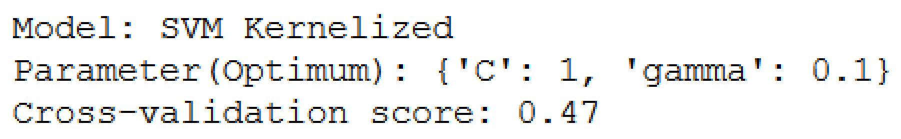
Classification Report on PCA reduced data after post processing -

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      2741
           1       0.51      0.24      0.33       347

    accuracy                           0.89      3088
   macro avg       0.71      0.61      0.63      3088
weighted avg       0.86      0.89      0.87      3088
```

3.1.4. SVM KERNELIZED

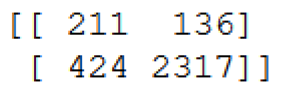
A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

The SVM model creates the hyperplane to gain the best separation with maximizing margin between the labelled classes and their associated PCs features in the dataset and the hyperplane cuts through the 50 orthogonal dimensions. This model is very efficient on the type of dataset and the accurate multiple classifications which we are expecting gain from it. Linear SVC can be used for linearly separable data and when the data assumes more complex shapes, Kernelized SVM can be used to classify the data.



```
Model: SVM Kernelized
Parameter(Optimum): {'C': 1, 'gamma': 0.1}
Cross-validation score: 0.47
```

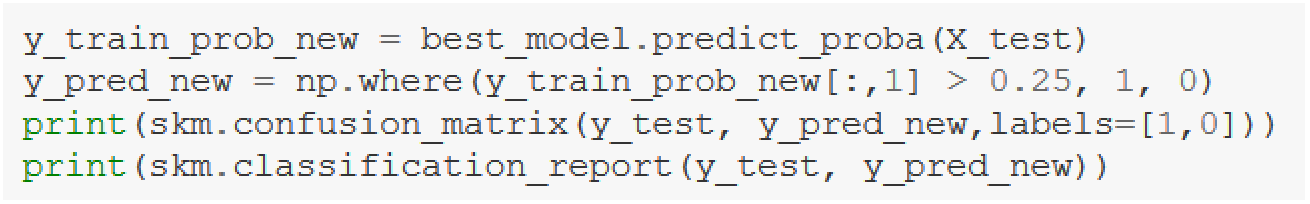
Confusion Matrix –



```
[[ 211  136]
 [ 424 2317]]
```

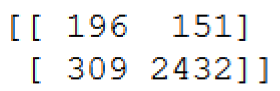
Post Processing –

We are decreasing predict_proba threshold from 50% to 25%.



```
y_train_prob_new = best_model.predict_proba(X_test)
y_pred_new = np.where(y_train_prob_new[:,1] > 0.25, 1, 0)
print(skm.confusion_matrix(y_test, y_pred_new,labels=[1,0]))
print(skm.classification_report(y_test, y_pred_new))
```

Confusion Matrix after post processing –



```
[[ 196  151]
 [ 309 2432]]
```

Classification Report after post processing –

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91      2741
           1       0.39      0.56      0.46       347

    accuracy                           0.85      3088
   macro avg       0.66      0.73      0.69      3088
weighted avg       0.88      0.85      0.86      3088
```

Kernelized SVM on PCA reduced dataset –

```
Model: SVM Kernelized
Parameter(Optimum): {'C': 1, 'gamma': 0.1}
Cross-validation score: 0.44
```

Confusion Matrix –

```
[[ 213  134]
 [ 476 2265]]
```

Confusion Matrix on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
[[ 186  161]
 [ 310 2431]]
```

Classification Report on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91      2741
           1       0.38      0.54      0.44       347

    accuracy                           0.85      3088
   macro avg       0.66      0.71      0.68      3088
weighted avg       0.87      0.85      0.86      3088
```

3.1.5. Decision Tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Tree based learning algorithms are considered as one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation.

```
Model: Decision Tree
Parameter(Optimum): {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto'}
Cross-validation score: 0.40
```

Confusion Matrix –

```
[[ 195  152]
 [ 471 2270]]
```

Post Processing –

Decreasing predict_proba threshold from 50% to 25%.

```
y_train_prob_new = best_model.predict_proba(X_test)
y_pred_new = np.where(y_train_prob_new[:,1] > 0.25, 1, 0)
print(skm.confusion_matrix(y_test, y_pred_new,labels=[1,0]))
print(skm.classification_report(y_test, y_pred_new))
```

Confusion Matrix after post processing –

```
[[ 282   65]
 [1762  979]]
```

Classification Report after post processing –

```
              precision    recall  f1-score   support

           0       0.94      0.36      0.52      2741
           1       0.14      0.81      0.24       347

    accuracy                           0.41      3088
   macro avg       0.54      0.58      0.38      3088
weighted avg       0.85      0.41      0.49      3088
```

Decision Tree on PCA reduced dataset –

```
Model: Decision Tree
Parameter(Optimum): {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'log2'}
Cross-validation score: 0.37
```

Confusion Matrix –

```
[[ 186  161]
 [ 509 2232]]
```

Confusion Matrix on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
[[ 347    0]
 [2741    0]]
```

Classification Report on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2741
           1       0.11      1.00      0.20       347

    accuracy                           0.11      3088
   macro avg       0.06      0.50      0.10      3088
weighted avg       0.01      0.11      0.02      3088
```

3.1.6. RANDOM FOREST

Random Forests is a type of decision tree model which creates multiple trees to reach the node through the most efficient path. The creation of multiple trees using random samples gives it the name Random Forest. It trains each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data.

```
Model: Random Forest
Parameter(Optimum): {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto'}
Cross-validation score: 0.46
```

Confusion Matrix –

```
[[ 202  145]
 [ 359 2382]]
```

Post Processing –

Decreasing the predict_proba threshold from 50% to 25%.

```
y_train_prob_new = best_model.predict_proba(X_test)
y_pred_new = np.where(y_train_prob_new[:,1] > 0.25, 1, 0)
print(skm.confusion_matrix(y_test, y_pred_new,labels=[1,0]))
print(skm.classification_report(y_test, y_pred_new))
```

Confusion Matrix after post processing –

```
[[ 316   31]
 [1840  901]]
```

Classification Report after post processing –

```
              precision    recall  f1-score   support

           0       0.97      0.33      0.49      2741
           1       0.15      0.91      0.25       347

    accuracy                           0.39      3088
   macro avg       0.56      0.62      0.37      3088
weighted avg       0.87      0.39      0.46      3088
```

Random Forest on PCA reduced dataset –

```
Model: Random Forest
Parameter(Optimum): {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto'}
Cross-validation score: 0.43
```

Confusion Matrix –

```
[[ 203  144]
 [ 464 2277]]
```

Confusion Matrix on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

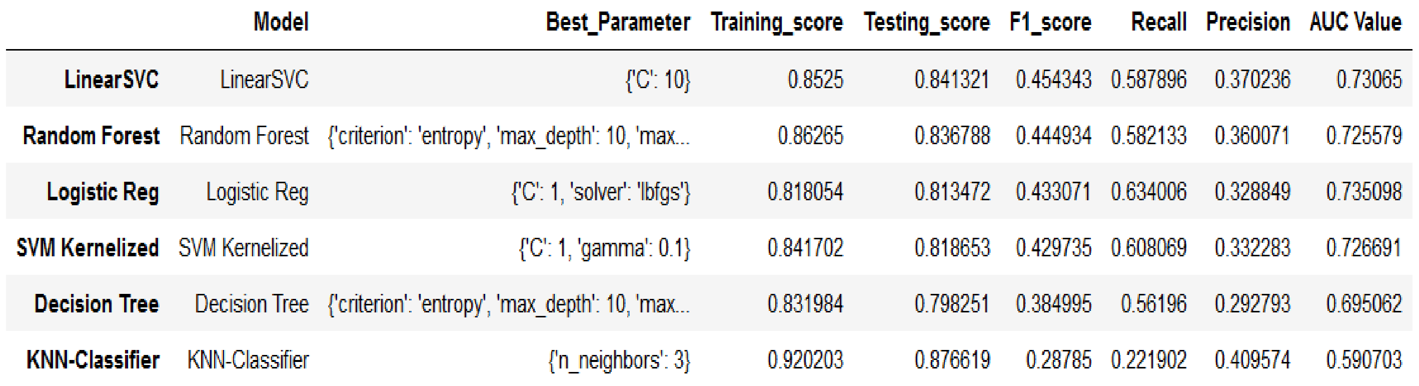
```
[[ 344    3]
 [2640  101]]
```

Classification Report on PCA reduced data after decreasing predict_proba threshold from 50% to 25% -

```
              precision    recall  f1-score   support

           0       0.97      0.04      0.07      2741
           1       0.12      0.99      0.21       347

    accuracy                           0.14      3088
   macro avg       0.54      0.51      0.14      3088
weighted avg       0.87      0.14      0.09      3088
```

4. MODEL EVALUATION & SELECTION

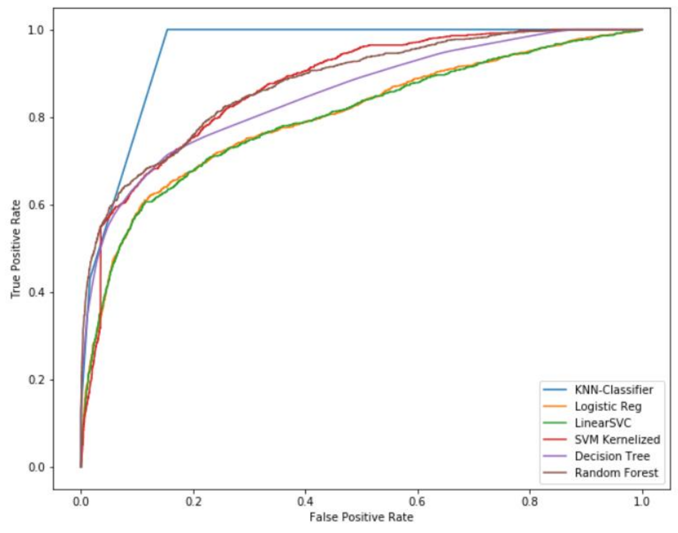
The results of the models on unreduced data sorted from the highest to lowest F1-Score are as follows –



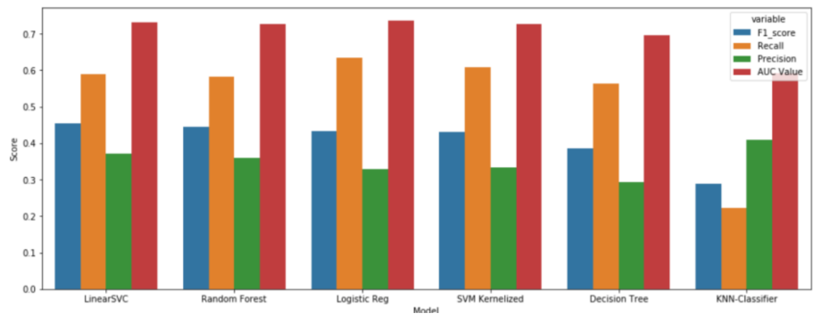
	Model	Best_Parameter	Training_score	Testing_score	F1_score	Recall	Precision	AUC Value
LinearSVC	LinearSVC	{'C': 10}	0.8525	0.841321	0.454343	0.587896	0.370236	0.73065
Random Forest	Random Forest	{'criterion': 'entropy', 'max_depth': 10, 'max...	0.86265	0.836788	0.444934	0.582133	0.360071	0.725579
Logistic Reg	Logistic Reg	{'C': 1, 'solver': 'lbfgs'}	0.818054	0.813472	0.433071	0.634006	0.328849	0.735098
SVM Kernelized	SVM Kernelized	{'C': 1, 'gamma': 0.1}	0.841702	0.818653	0.429735	0.608069	0.332283	0.726691
Decision Tree	Decision Tree	{'criterion': 'entropy', 'max_depth': 10, 'max...	0.831984	0.798251	0.384995	0.56196	0.292793	0.695062
KNN-Classifier	KNN-Classifier	{'n_neighbors': 3}	0.920203	0.876619	0.28785	0.221902	0.409574	0.590703

Looking at the F1 Score values – Linear SVC, Random Forest and Logistic Regression perform better than other models. Higher AUC tells us that the model is better at distinguishing between both classes, in terms of AUC too, these models perform well.

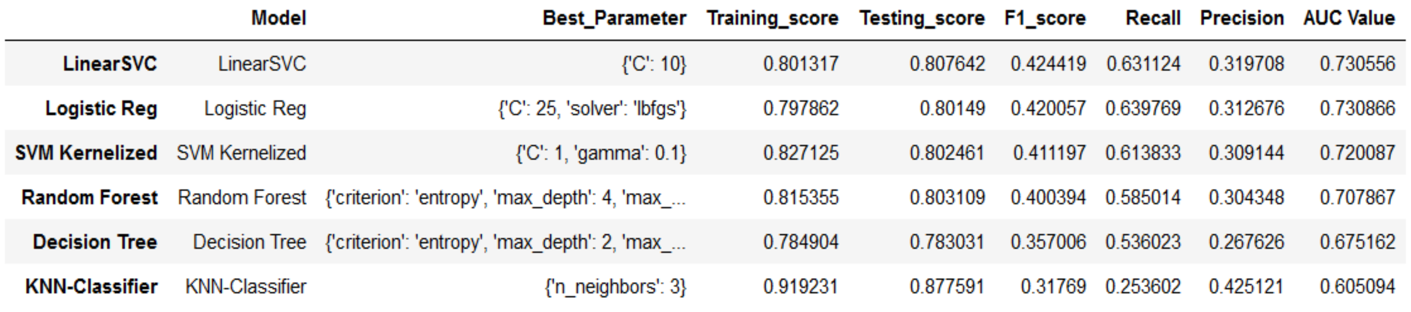
ROC Curves –



Model Performance Measures Comparison –



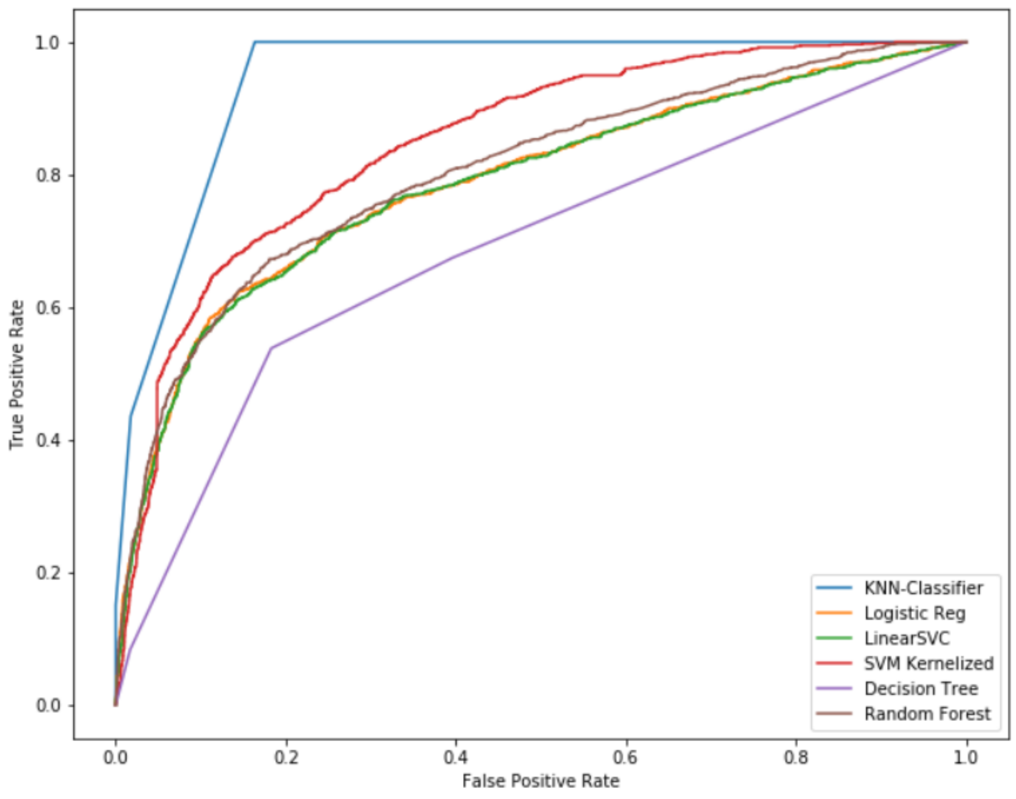
The models’ performance on PCA reduced dataset is as follows –



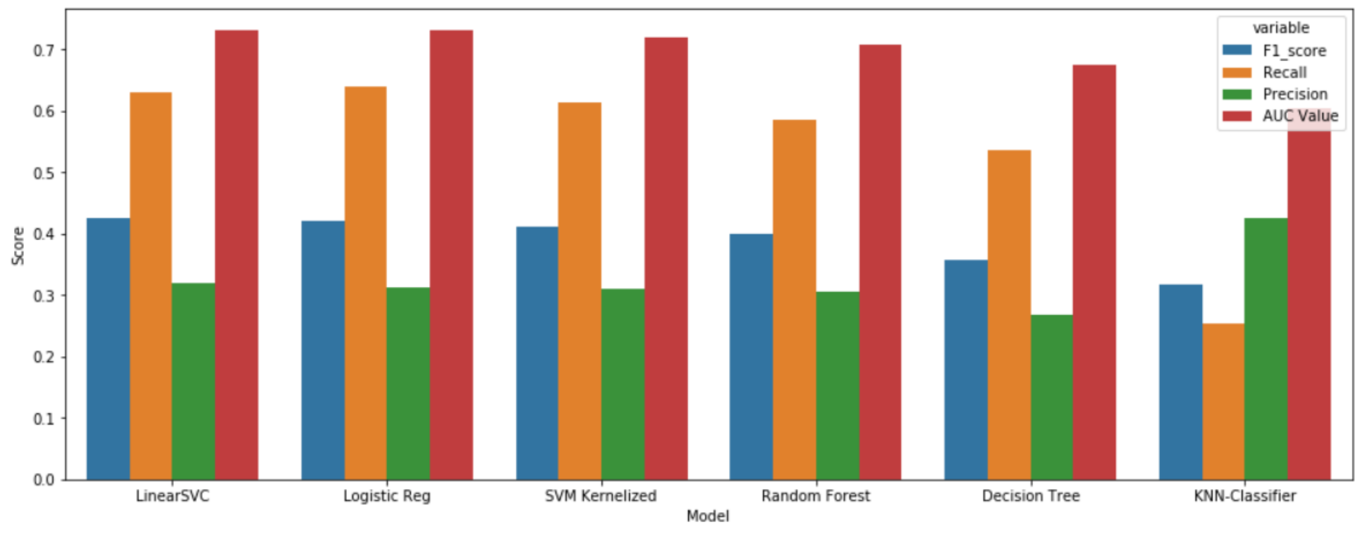
	Model	Best_Parameter	Training_score	Testing_score	F1_score	Recall	Precision	AUC Value
LinearSVC	LinearSVC	{'C': 10}	0.801317	0.807642	0.424419	0.631124	0.319708	0.730556
Logistic Reg	Logistic Reg	{'C': 25, 'solver': 'lbfgs'}	0.797862	0.80149	0.420057	0.639769	0.312676	0.730866
SVM Kernelized	SVM Kernelized	{'C': 1, 'gamma': 0.1}	0.827125	0.802461	0.411197	0.613833	0.309144	0.720087
Random Forest	Random Forest	{'criterion': 'entropy', 'max_depth': 4, 'max_...	0.815355	0.803109	0.400394	0.585014	0.304348	0.707867
Decision Tree	Decision Tree	{'criterion': 'entropy', 'max_depth': 2, 'max_...	0.784904	0.783031	0.357006	0.536023	0.267626	0.675162
KNN-Classifier	KNN-Classifier	{'n_neighbors': 3}	0.919231	0.877591	0.31769	0.253602	0.425121	0.605094

On the PCA reduced dataset - Linear SVC, Logistic Regression and Kernelized SVM perform better than other models looking at the F1 Score.

ROC Curves for models on PCA reduced dataset -

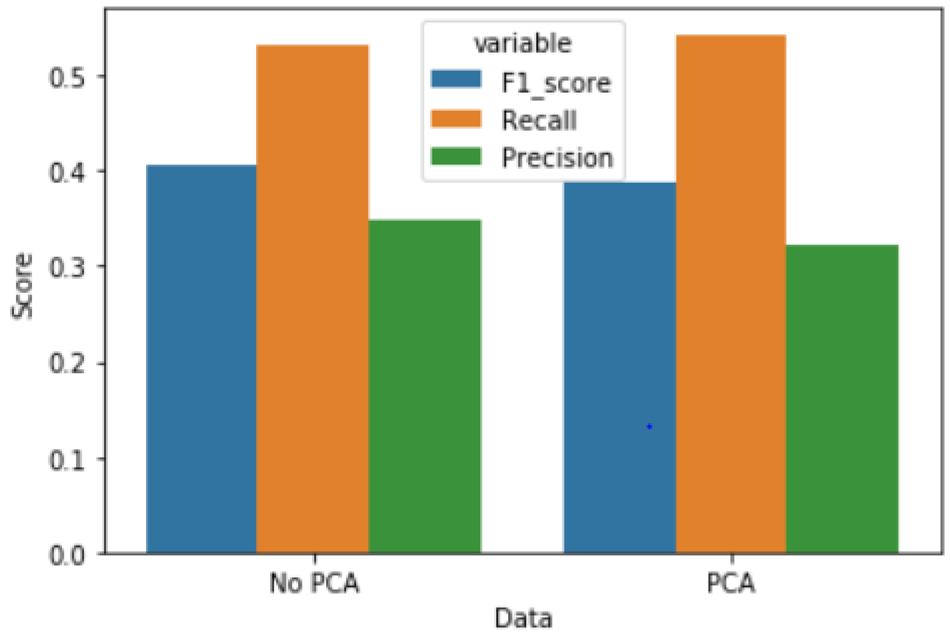


Model Performance Measures Comparison –

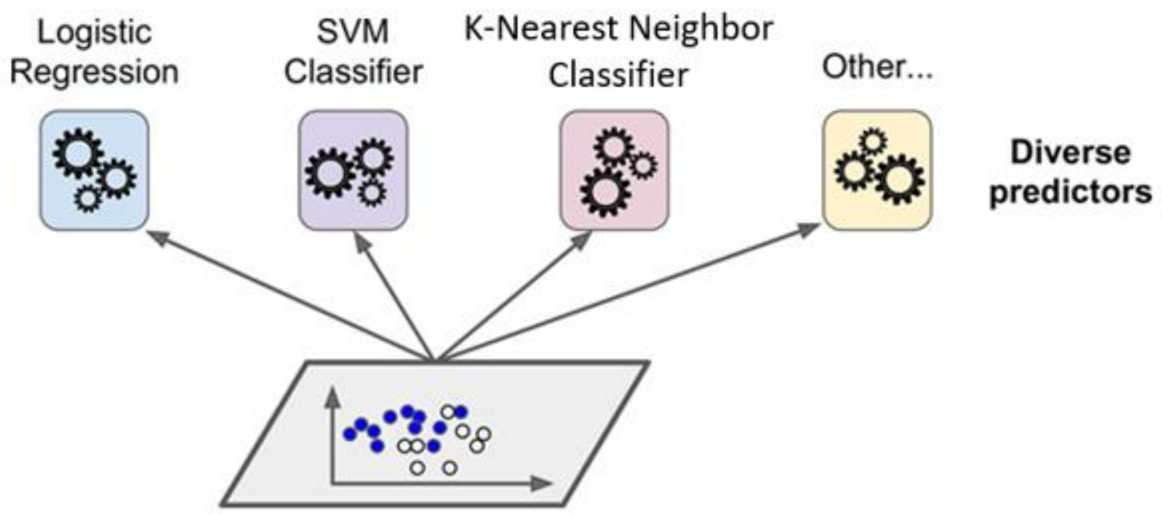


A comparison of models' performance with PCA and no PCA is as follows –

	Data	F1_score	Recall	Precision
No PCA	No PCA	0.405821	0.532661	0.348968
PCA	PCA	0.38846	0.543228	0.323104



We then built a voting classifier to leverage the predictive power of our best models. We used dataset without PCA for our ensemble model and employed the soft voting methodology.



The results from the Voting Classifier are as follows –

Confusion Matrix –

```
[[  75  272]
 [  65 2676]]
```

Classification Report –

```
              precision    recall  f1-score   support

           1       0.54      0.22      0.31       347
           0       0.91      0.98      0.94      2741

    accuracy                           0.89      3088
   macro avg       0.72      0.60      0.62      3088
weighted avg       0.87      0.89      0.87      3088
```

Post Processing –

```
y_train_prob_new = voting_clf.predict_proba(X_test)
y_pred_new = np.where(y_train_prob_new[:,1] > 0.25, 1, 0)
print(skm.confusion_matrix(y_test, y_pred_new,labels=[1,0]))
print(skm.classification_report(y_test, y_pred_new))
```

Confusion Matrix After Post Processing –

```
[[ 149  198]
 [ 261 2480]]
```

Classification Report after post processing –

```
              precision    recall  f1-score   support

           0       0.93      0.90      0.92      2741
           1       0.36      0.43      0.39       347

    accuracy                           0.85      3088
   macro avg       0.64      0.67      0.65      3088
weighted avg       0.86      0.85      0.86      3088
```

Our recall, which is the True Positive Rate, significantly increased. This signifies that our ensemble model correctly classifies the people who are likely to make a term deposit. In terms of F1-Score, Linear SVC on unreduced dataset is our best model.

ONGOING & FUTURE IMPLEMENTATION PLANS

We have performed adaptive boosting and gradient boosting on our models.

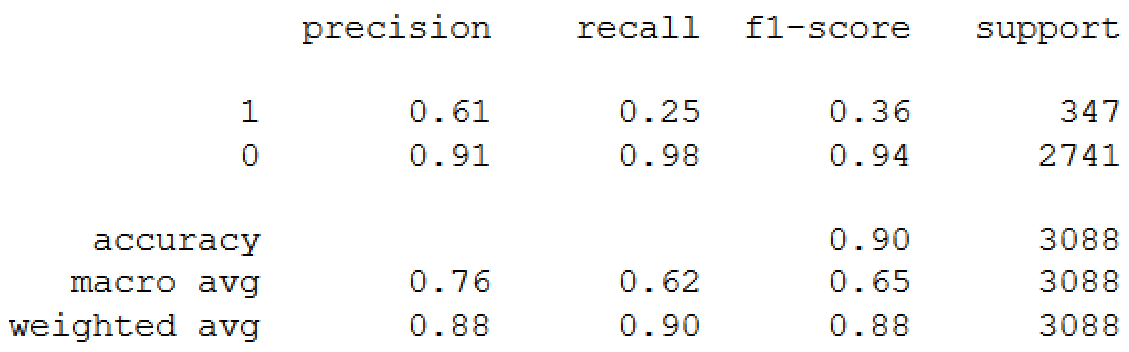
AdaBoost with Logistic Regression Results –

```
[[  72  275]
 [  50 2691]]
```

```
              precision    recall  f1-score   support

           1       0.59      0.21      0.31       347
           0       0.91      0.98      0.94      2741

    accuracy                           0.89      3088
   macro avg       0.75      0.59      0.63      3088
weighted avg       0.87      0.89      0.87      3088
```

Gradient Boosting Classifier Results –

```
[[  87  260]
 [  56 2685]]
```



```
              precision    recall  f1-score   support

           1       0.61      0.25      0.36       347
           0       0.91      0.98      0.94      2741

    accuracy                           0.90      3088
   macro avg       0.76      0.62      0.65      3088
weighted avg       0.88      0.90      0.88      3088
```

We plan to implement neural nets and bagging classifiers in the near future. We also plan to automate the model and parameter selections.