SMART_CALL!
APPLICATION OF DATAMINING FOR BANK DIRECT MARKETING

Project Report

Practical Application Assignment 17.1

Comparing Classifiers

For

Partial Requirements

Of

UC Berkeley Certificate in AI/ML Program

BH-PCMLAI-M01

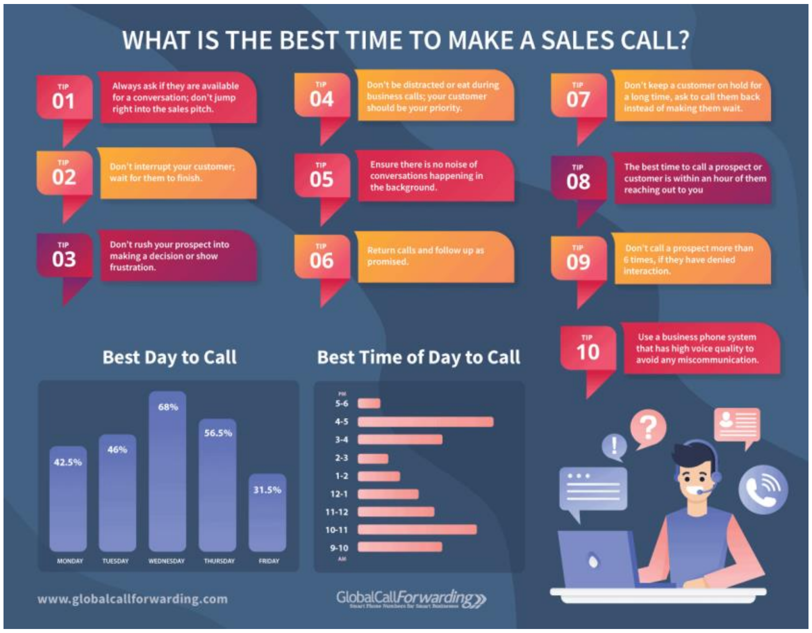
OCTOBER 2, 2022

SHAJI RAVINDRANATHAN
Section B

1 INTRODUCTION

Call center agents conduct outbound or inbound phone sales, and face several challenges on a day-to-day basis. Phone sales have been an effective way to earn new business over the years, but the sales environment is full of obstacles. These challenges impact the customer experience, agent satisfaction and, of course, business results. Increasingly they have to compete with the digital self service channels for business.

“68% of B2B sales[1] were found to involve some form of human interaction such as telemarketing. Customers have been found to respond to better to the human touch than to some other form such as digital marketing. Figure 1 shows a series of tips for telemarketers on how to make a sales call.



WHAT IS THE BEST TIME TO MAKE A SALES CALL?

TIP 01 Always ask if they are available for a conversation; don't jump right into the sales pitch.

TIP 02 Don't interrupt your customer; wait for them to finish.

TIP 03 Don't rush your prospect into making a decision or show frustration.

TIP 04 Don't be distracted or eat during business calls; your customer should be your priority.

TIP 05 Ensure there is no noise of conversations happening in the background.

TIP 06 Return calls and follow up as promised.

TIP 07 Don't keep a customer on hold for a long time, ask to call them back instead of making them wait.

TIP 08 The best time to call a prospect or customer is within an hour of them reaching out to you

TIP 09 Don't call a prospect more than 6 times, if they have denied interaction.

TIP 10 Use a business phone system that has high voice quality to avoid any miscommunication.

Figure 1 Global Call Forwarding Tips for Telemarketing

Baylor University’s Keller Center for Research and a team of Baylor MBA students partnered with the research and development department at Keller Williams Realty International (KWRI) to begin to quantify the importance and effectiveness of cold calling as a prospecting tool. One of the interesting statistics they mention is that it takes 8 cold calls to reach a prospect and 72% of all sales calls aren’t answered. The average salesperson generates roughly one appointment or referral from every 209 cold calls. (KELLER RESEARCH REPORT)[2].

According to recent study from a sales enablement platform company Gong inc and Rental Beast B2B real estate brokerage firm 4 to 6pm is the best time to call to contact a lead (by 114% over the worst time block). The odds of calling to contact a lead decrease by over 10x in the 1st hour. The odds of contacting a lead if called in 5 minutes versus 30 minutes drop 100x.[3].

[1] https://www.globalcallforwarding.com/

[2] https://www.baylor.edu/content/services/document.php/183060.pdf/

[3] https://rentalbeast.blog/2019/11/06/whats-the-best-time-to-contact-a-lead/

Business Objective: Given all the statistics and challenges can the good old CRM data be made more effective. Can a sales organization improve its market performance by analyzing customer data and come up with an efficient direct marketing campaign by improving customer experience during a sales call? The objective is to come up with an approach to predict the success of telemarketing calls for selling bank long-term deposits using machine learning classification techniques. The main task is to derive a machine learning model that can explain success of a contact, i.e., if the client subscribes the deposit. Such a model can increase campaign efficiency by identifying the main characteristics that affect success, helping in a better management of the available resources (e.g., human effort, phone calls, time) and selection of a high quality and affordable set of potential buying customers.

2 BUSINESS UNDERSTANDING

In this business case, we are presented with anonymized customer data that were collected from a Portuguese marketing campaign for bank deposit subscriptions. The business goal of the bank is to find a machine learning model that can explain success of a contact, i.e., if the client subscribes the deposit.

CRM data of direct marketing campaigns (phone calls) from the banking institution is provided. The classification goal is to predict if the client will subscribe a term deposit (target variable y). This case study is adapted from a research paper where the researchers Sergio Moro and Raul Laurceano have used a very similar dataset as the one provided to study the effectiveness of Laureano Bank Telemarketing[4]. As per the paper their model was able to achieve an AUC score of 0.8 and a ALIFT of 0.7. The objective of this exercise will be to come close to the effort.

What are the various factors involved in classification of a customer? What are the important customer attributes necessary for effective classification. A typical bank consumer profile will have the following attributes:

- Age and job profile
- Education Level
- Housing Data
- Credit Default History
- Outstanding Debt Profile
- Marital Status
- Frequency and Recency of direct contacts/campaigns etc.

2.1 Problem Definition

Given the various attributes of a typical Bank Customer, can a model be derived which can increase campaign efficiency by identifying the main characteristics that affect success, helping in a better management of the available resources (e.g., human effort, phone calls, time) and selection of a high quality and affordable set of potential buying customers. Can the customer classification be done based on the given customer attributes with reasonable accuracy?

[4] https://core.ac.uk/download/pdf/55631291.pdf

2.2 Solution

This report outlines a customer classification engine **(Smart_Call!),** that is designed from the ground up to induce frictionless transaction experience between bank telemarketer and potential bank customers.

This customer classification prediction system uses data mining techniques to build a model using a variety of features and attributes. **Smart_Call!** is a classification tool that mines the CRM data to identify potential customers who are highly likely to buy the advertised product. This allows the telemarketer to talk to potential customers in a frictionless manner.

Target Audience: The primary audience for this report is telemarketers, however it can be used by all of the categories of users who are involved with marketing campaigns (Inbound, Outbound Sales, Channel Marketing, Product Managers etc.)

3 DATA UNDERSTANDING AND EXPLORATORY ANALYSIS

The data used in this project was provided in the application 3 starter package. The original dataset is from Kaggle, that contained information collected from **17** campaigns that occurred between **May 2008 and November 2010**. A total of **79354** customer contacts were made to collect the data. The redacted dataset provided has 20 client attributes. It was conducted through two channels which were:

1.The telephone, with a human agent as the interlocutor, was the dominant marketing channel.

2. Occasional auxiliary use of the Internet online banking channel.

In the provided dataset the **Input variables** are as follows**:**

# bank client data:

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5 - default: has credit in default? (categorical: 'no','yes','unknown')

6 - housing: has housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown')

# related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

11 - duration: last contact duration, in seconds (numeric). Important note: 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.

# other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

# social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

17 - cons.price.idx: consumer price index - monthly indicator (numeric)

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

20 - nr.employed: number of employees - quarterly indicator (numeric)

The Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

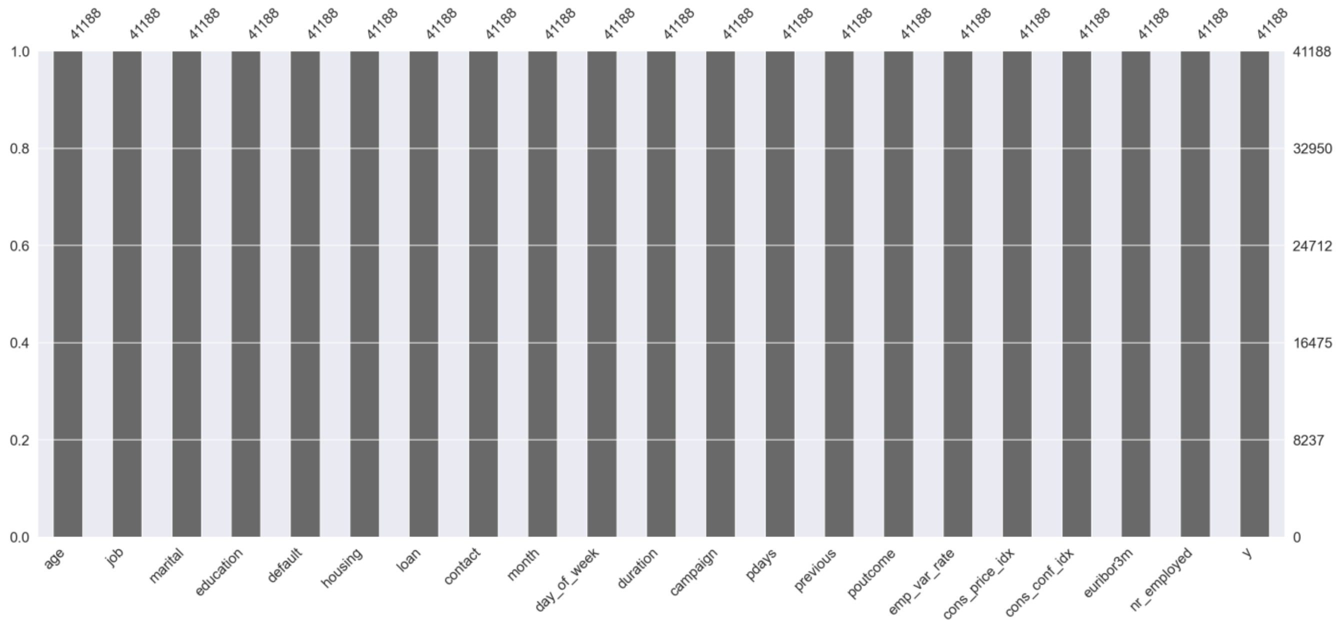
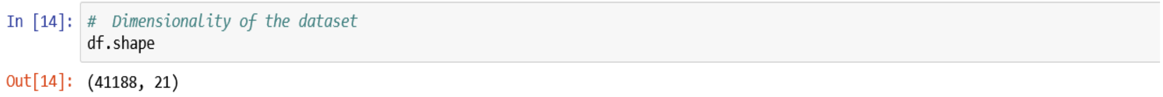


Figure 2 List of features in the bank crm dataset

We can see from figure 2 which is visual representation of the dataset, the last column features the classification outcome. This is a binary classification problem. Our two classes are “yes” denoting that the customer subscribed to a term deposit, and “no” denoting that the customer did not subscribe. This is the classification feature we want to predict based on the other 20 features in the dataset.



```
In [14]: #  Dimensionality of the dataset
         df.shape

Out[14]: (41188, 21)
```

There are no missing values in the 20 feature columns, hence missing data cleaning is not necessary.

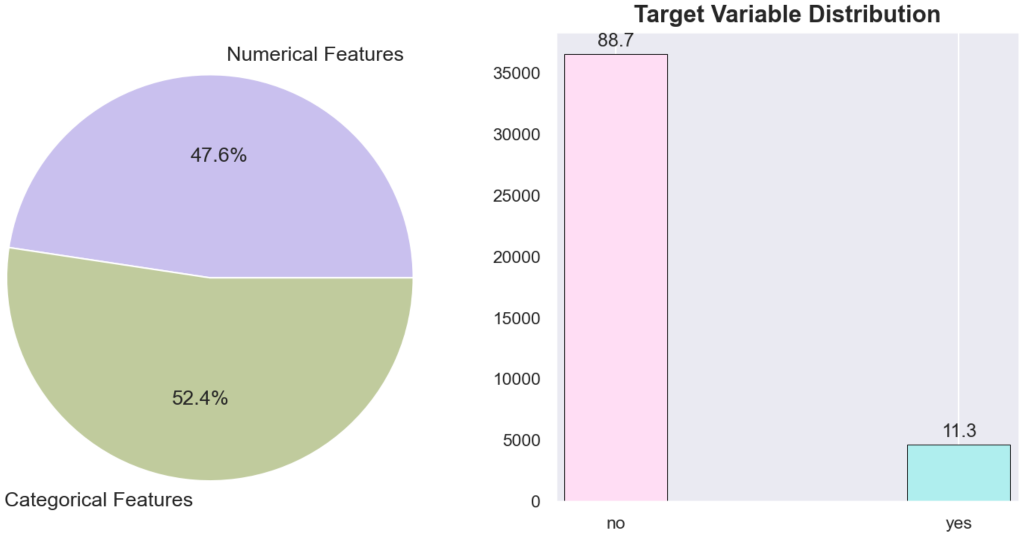
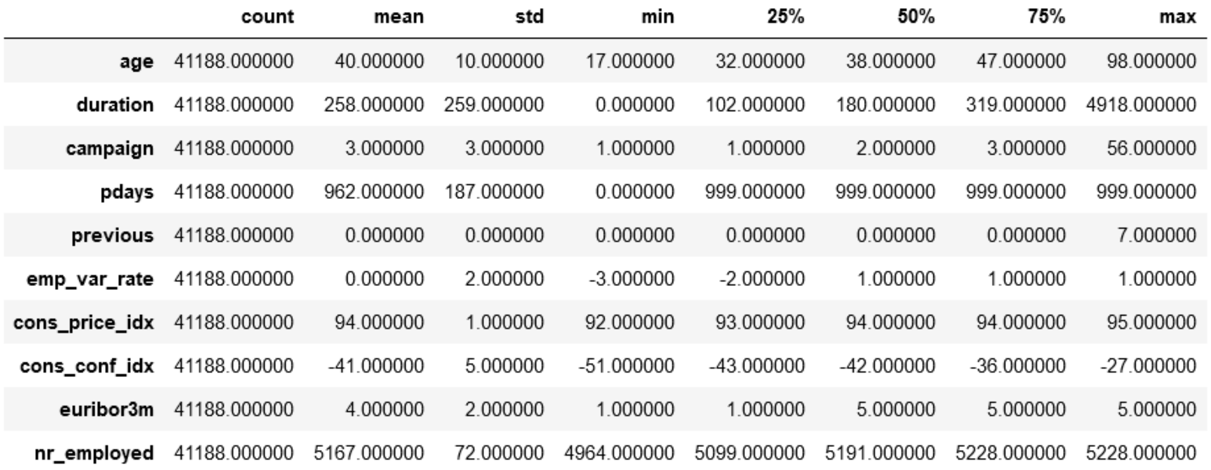


Figure 3 Dataset Attributes

The figure 3 below shows the distribution of categorical and numerical features in the given dataset. We can also infer from the plot above that there is a class imbalance in the output variable 'no' accounts for 88.7% of the whole and a 'yes' is about 11.3%. The number of negative class is roughly 8 times the number of positive class. Some univariate analysis will be necessary to check whether we can take each feature and see whether it can help us in delineating the classes better. We have to figure out whether there is a specific category of customers that enrolls for term deposit more than others.

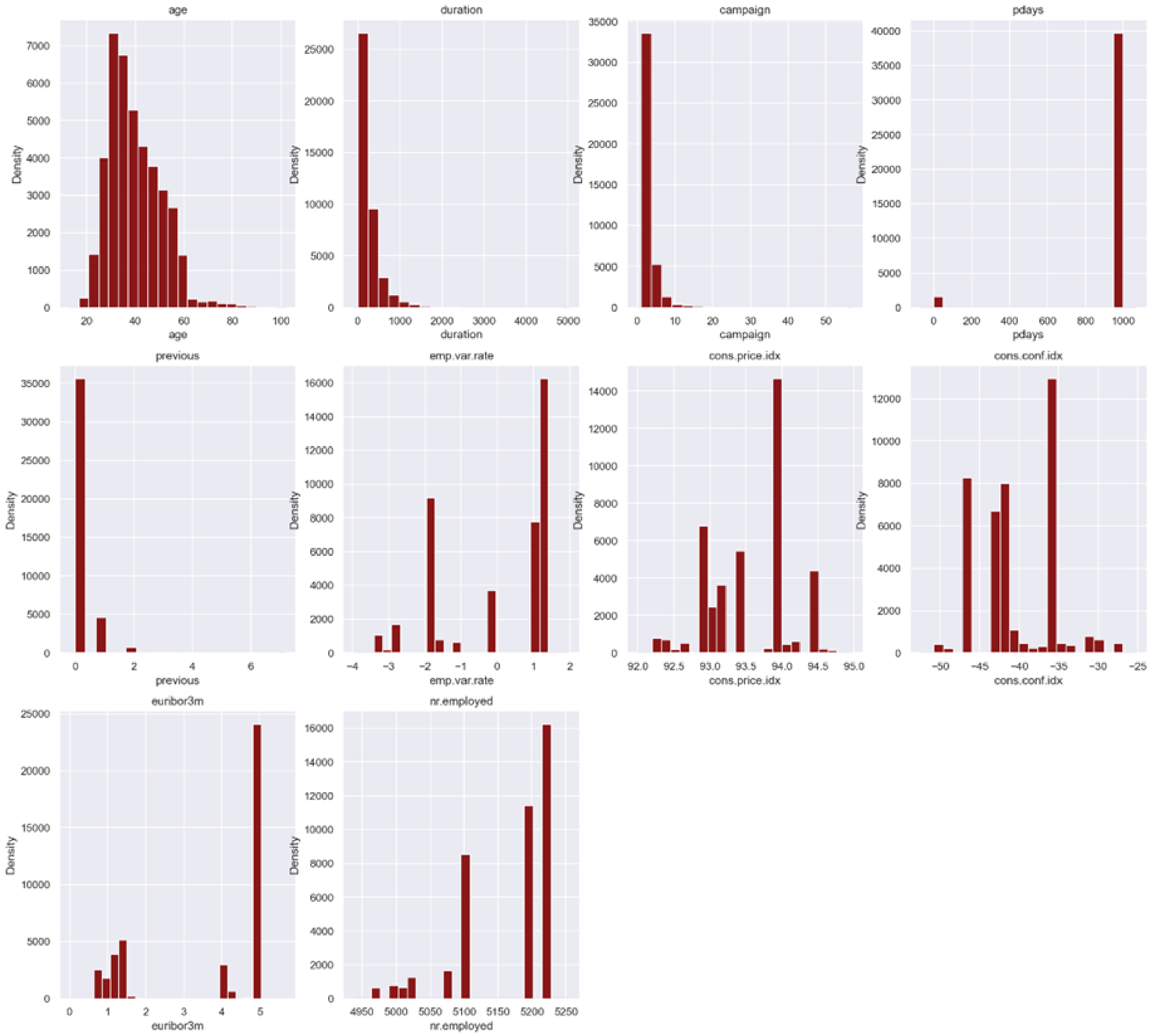


	count	mean	std	min	25%	50%	75%	max
age	41188.000000	40.000000	10.000000	17.000000	32.000000	38.000000	47.000000	98.000000
duration	41188.000000	258.000000	259.000000	0.000000	102.000000	180.000000	319.000000	4918.000000
campaign	41188.000000	3.000000	3.000000	1.000000	1.000000	2.000000	3.000000	56.000000
pdays	41188.000000	962.000000	187.000000	0.000000	999.000000	999.000000	999.000000	999.000000
previous	41188.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	7.000000
emp_var_rate	41188.000000	0.000000	2.000000	-3.000000	-2.000000	1.000000	1.000000	1.000000
cons_price_idx	41188.000000	94.000000	1.000000	92.000000	93.000000	94.000000	94.000000	95.000000
cons_conf_idx	41188.000000	-41.000000	5.000000	-51.000000	-43.000000	-42.000000	-36.000000	-27.000000
euribor3m	41188.000000	4.000000	2.000000	1.000000	1.000000	5.000000	5.000000	5.000000
nr_employed	41188.000000	5167.000000	72.000000	4964.000000	5099.000000	5191.000000	5228.000000	5228.000000

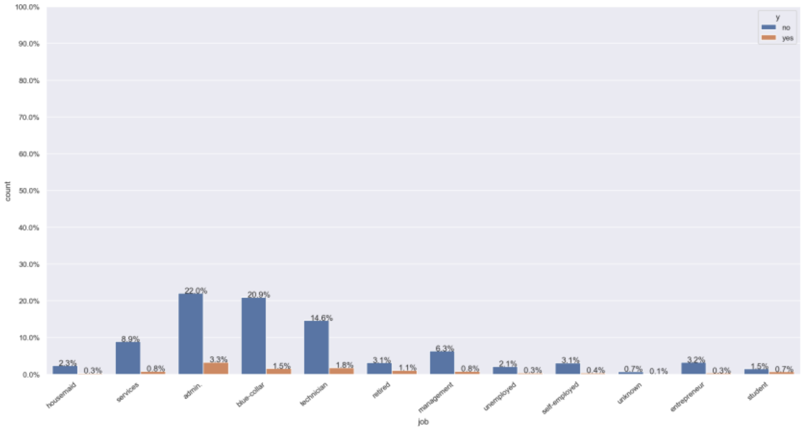
Figure 4 Summary Statistics

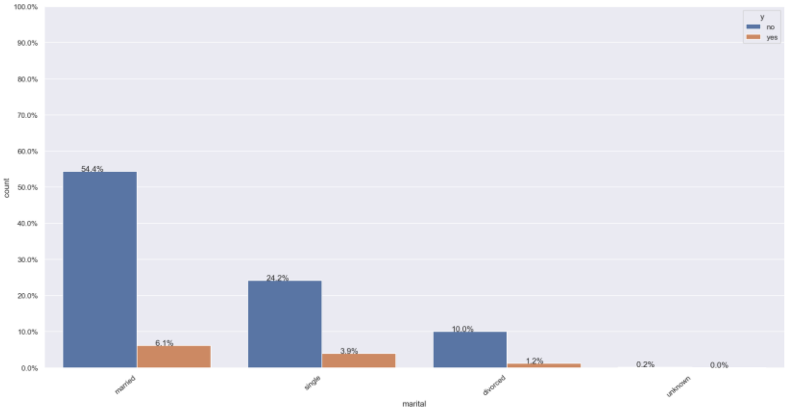
The dataset is diverse. We can see from the descriptive statistics that many of the features need to be normalized/scaled. Tree based model would be able to handle outliers in this dataset. There is a mix data of both int and floating type. The range of data is diverse especially the pdays feature that needs to be looked into detail. 999 needs replacing with 0 as dataset notes suggests. We should be able to drop duration as per dataset notes because of undue predictive influence.

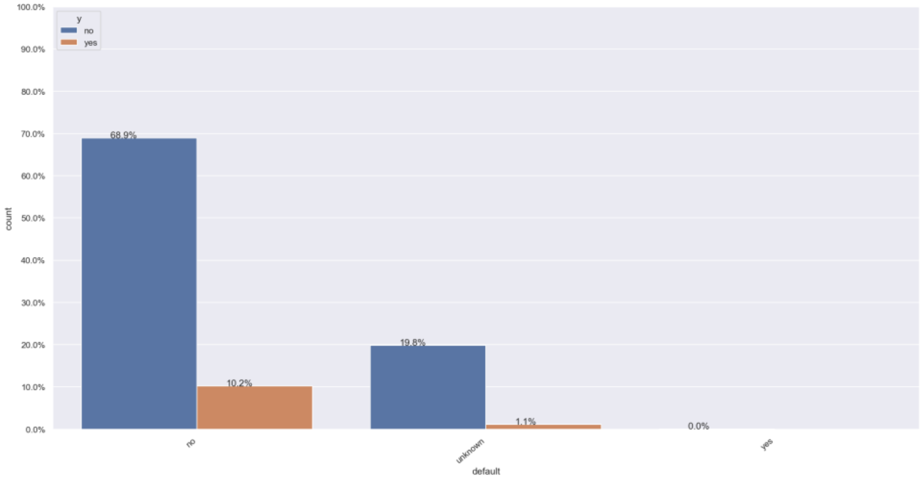
Next we will look at the count plots of some of the 20 features to find out the class distribution for each feature. Since there are a large number of features to look at, a separate function was defined to visualize the count plots.



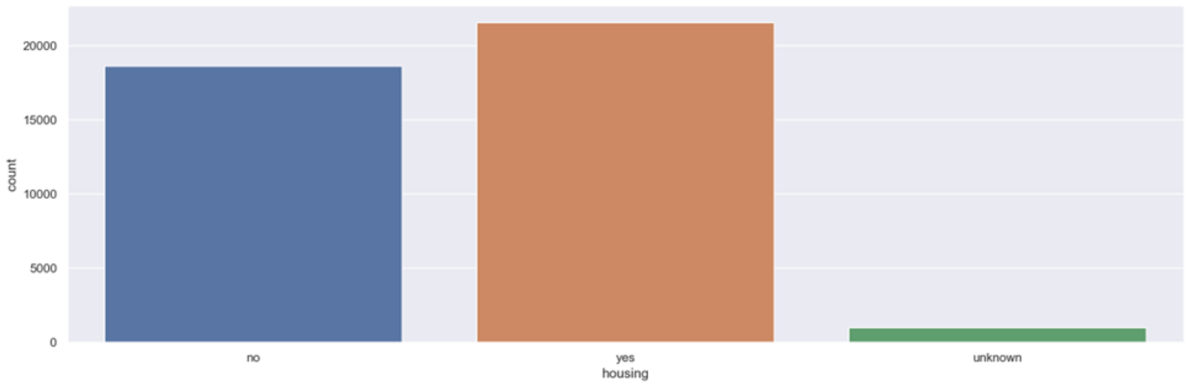
3.1.1 Observations:



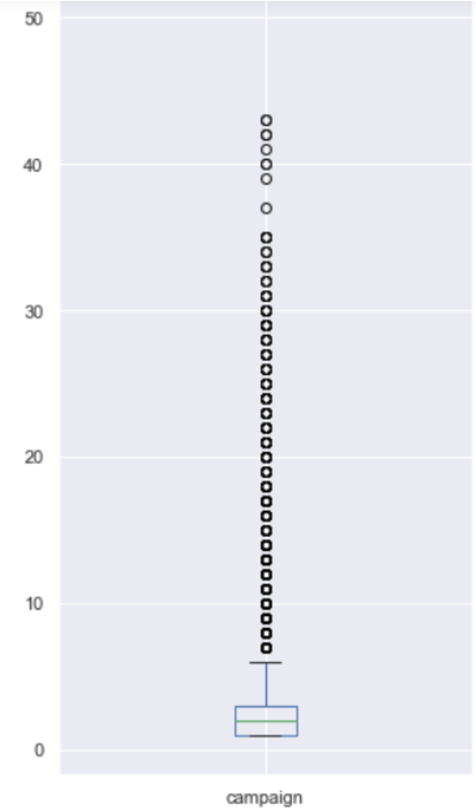




- The top three job profiles for customers are "admin", "blue-collar" or "technician". Customers who with ‘admin’ job profiles have the highest rate of term deposits as well as highest rates for non-subscription. This may be simply because there are a greater number of admin job profiles. Very hard to figure out a correlation between job and target variable.
- Married, single and divorced categories make up the top three categories of customers.
- We can also see that majority of the customers do not have credit in default.



- A large number of the customers have active home loans



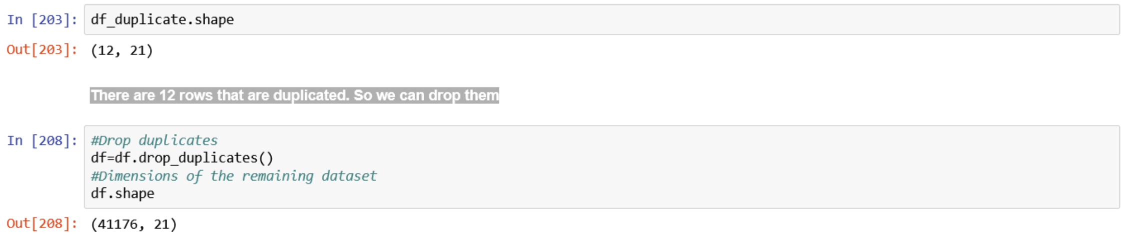
- Majority of the Bank's customers have been contacted by the campaign around 1-3 times. There are some clients who seem to have been contacted as many as 48 times. This outlier could be an anomalous record.
- From the existing customers, the previous marketing campaign outcome records are not present. This would imply that most of the customers are new customers who have not been contacted earlier. Among customers who had a successful outcome from the previous campaign, a majority of them did not opt in to open a term deposit. The class distribution is 2.2% for positive class, and 1.2% for negative class. So, we can assume that this feature may be important in predicting the target variable.

4 DATA PREPARATION:

4.1 Data Cleaning:

The following steps were undertaken as a part of the data cleaning procedure:

1. Some features are marked with "." and others with"_". So, we should make it uniform for consistency.
2. Check for duplicate data and drop the corresponding rows.

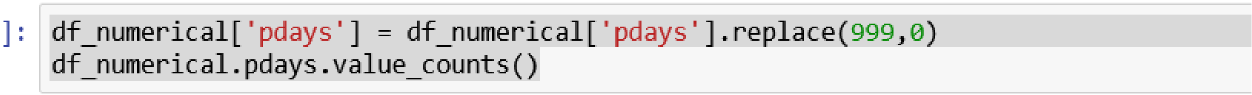


```
In [203]: df_duplicate.shape
Out[203]: (12, 21)
```

There are 12 rows that are duplicated. So we can drop them

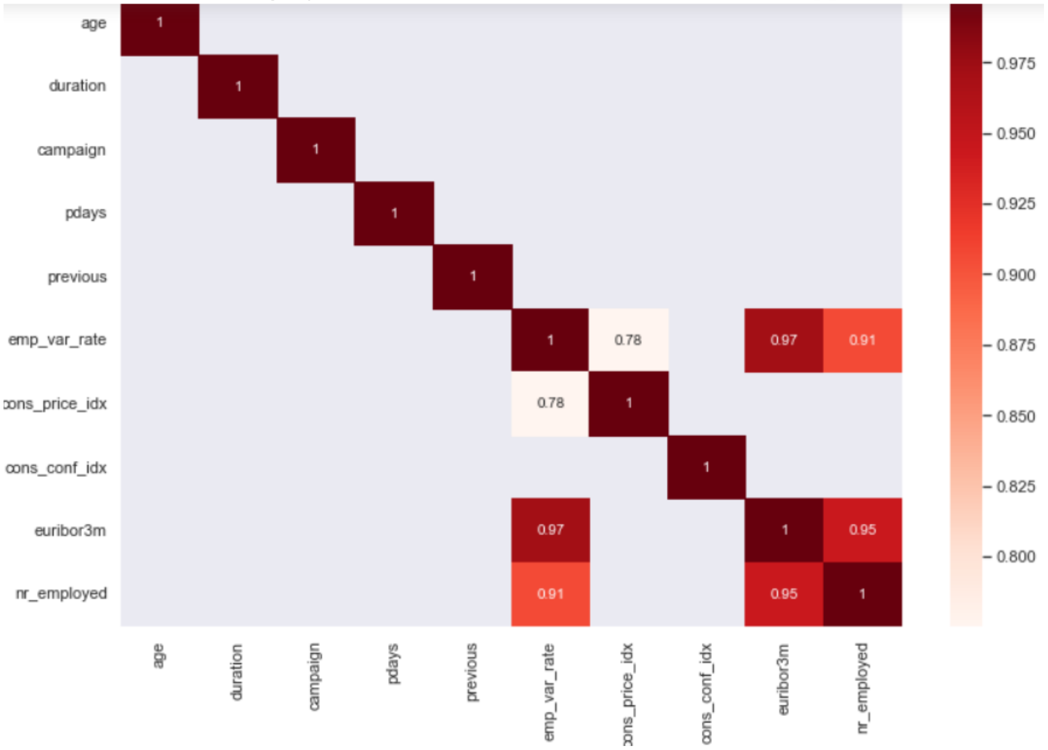
```
In [208]: #Drop duplicates
          df=df.drop_duplicates()
          #Dimensions of the remaining dataset
          df.shape
Out[208]: (41176, 21)
```

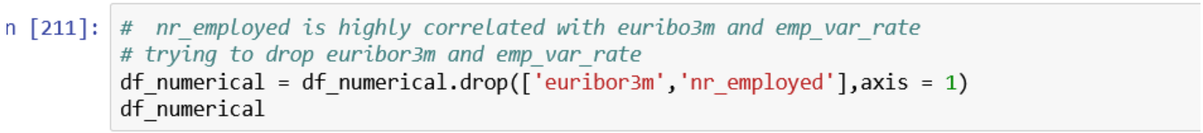
3. As mentioned above the range of data is diverse especially the pdays feature that needs to be looked into detail. 999 needs replacing with 0 as dataset notes suggests.



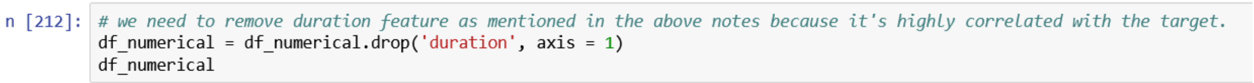
```
]: df_numerical['pdays'] = df_numerical['pdays'].replace(999,0)
   df_numerical.pdays.value_counts()
```

4. Some features are highly correlated So those columns were dropped.



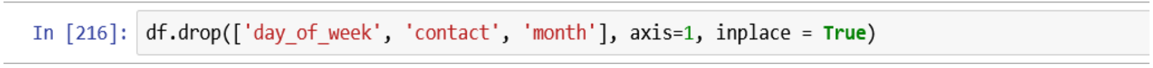


```
n [211]: #  nr_employed is highly correlated with euribo3m and emp_var_rate
         # trying to drop euribor3m and emp_var_rate
         df_numerical = df_numerical.drop(['euribor3m','nr_employed'],axis = 1)
         df_numerical
```



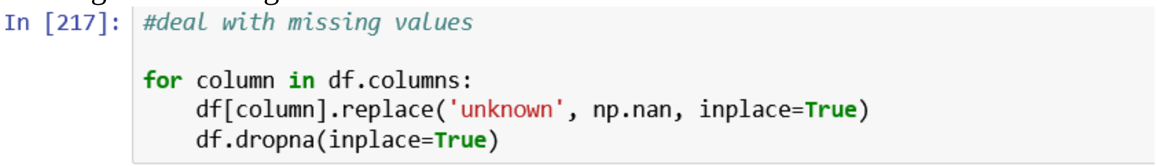
```
n [212]: # we need to remove duration feature as mentioned in the above notes because it's highly correlated with the target.
         df_numerical = df_numerical.drop('duration', axis = 1)
         df_numerical
```

5. Regardless of the day customers are contacted, there does not seem to be any bearing on the willingness to sign on for a deposit account, so removing those

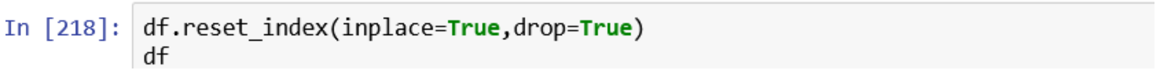


```
In [216]: df.drop(['day_of_week', 'contact', 'month'], axis=1, inplace = True)
```

6. Dealing with missing values or NAN.

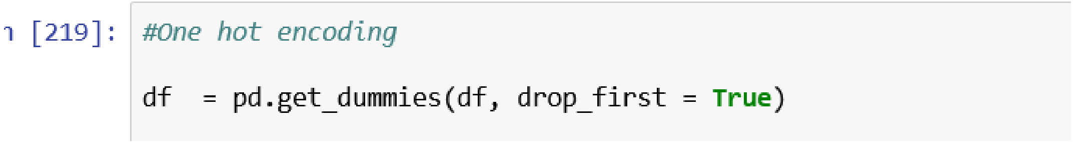


```
In [217]: #deal with missing values

          for column in df.columns:
              df[column].replace('unknown', np.nan, inplace=True)
              df.dropna(inplace=True)
```



```
In [218]: df.reset_index(inplace=True,drop=True)
          df
```

7. Encode categorical integer features using a one-hot encoding

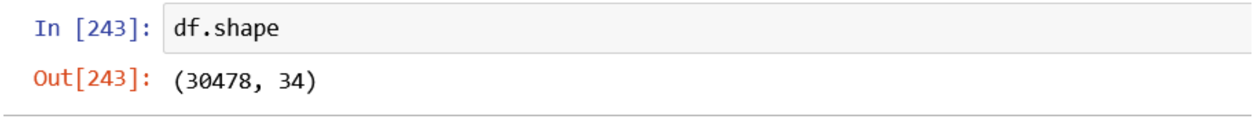


```
ı [219]: #One hot encoding

         df  = pd.get_dummies(df, drop_first = True)
```

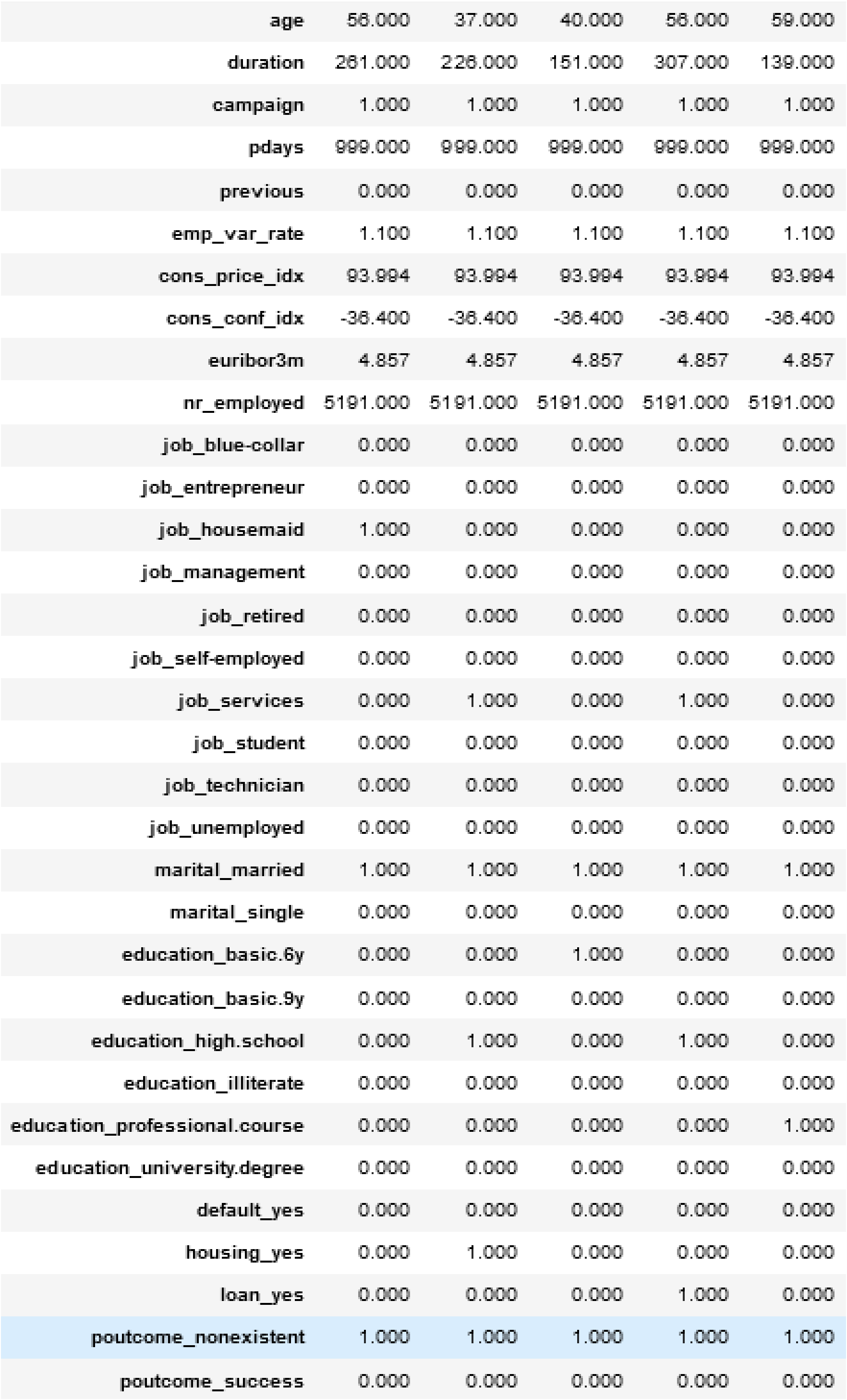
Figure 5 One Hot Encoding of categorical features

8. Figure 5 shows the cleaned dataset with all the missing data addressed.

We have 30478 rows with 34 features.



```
In [243]: df.shape
Out[243]: (30478, 34)
```



age	56.000	37.000	40.000	56.000	59.000
duration	261.000	226.000	151.000	307.000	139.000
campaign	1.000	1.000	1.000	1.000	1.000
pdays	999.000	999.000	999.000	999.000	999.000
previous	0.000	0.000	0.000	0.000	0.000
emp_var_rate	1.100	1.100	1.100	1.100	1.100
cons_price_idx	93.994	93.994	93.994	93.994	93.994
cons_conf_idx	-36.400	-36.400	-36.400	-36.400	-36.400
euribor3m	4.857	4.857	4.857	4.857	4.857
nr_employed	5191.000	5191.000	5191.000	5191.000	5191.000
job_blue-collar	0.000	0.000	0.000	0.000	0.000
job_entrepreneur	0.000	0.000	0.000	0.000	0.000
job_housemaid	1.000	0.000	0.000	0.000	0.000
job_management	0.000	0.000	0.000	0.000	0.000
job_retired	0.000	0.000	0.000	0.000	0.000
job_self-employed	0.000	0.000	0.000	0.000	0.000
job_services	0.000	1.000	0.000	1.000	0.000
job_student	0.000	0.000	0.000	0.000	0.000
job_technician	0.000	0.000	0.000	0.000	0.000
job_unemployed	0.000	0.000	0.000	0.000	0.000
marital_married	1.000	1.000	1.000	1.000	1.000
marital_single	0.000	0.000	0.000	0.000	0.000
education_basic.6y	0.000	0.000	1.000	0.000	0.000
education_basic.9y	0.000	0.000	0.000	0.000	0.000
education_high.school	0.000	1.000	0.000	1.000	0.000
education_illiterate	0.000	0.000	0.000	0.000	0.000
education_professional.course	0.000	0.000	0.000	0.000	1.000
education_university.degree	0.000	0.000	0.000	0.000	0.000
default_yes	0.000	0.000	0.000	0.000	0.000
housing_yes	0.000	1.000	0.000	0.000	0.000
loan_yes	0.000	0.000	0.000	1.000	0.000
poutcome_nonexistent	1.000	1.000	1.000	1.000	1.000
poutcome_success	0.000	0.000	0.000	0.000	0.000

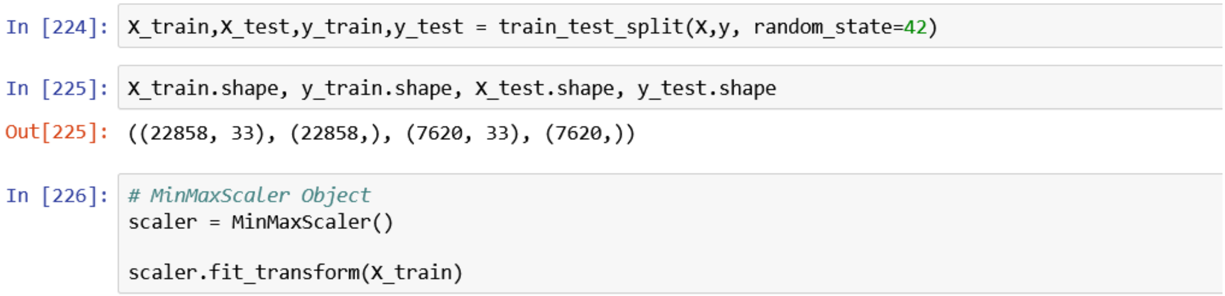
Figure 6 Cleaned Dataset for Machine Learning

5 DATA MODELING

5.1 Methodology used in solving the problem

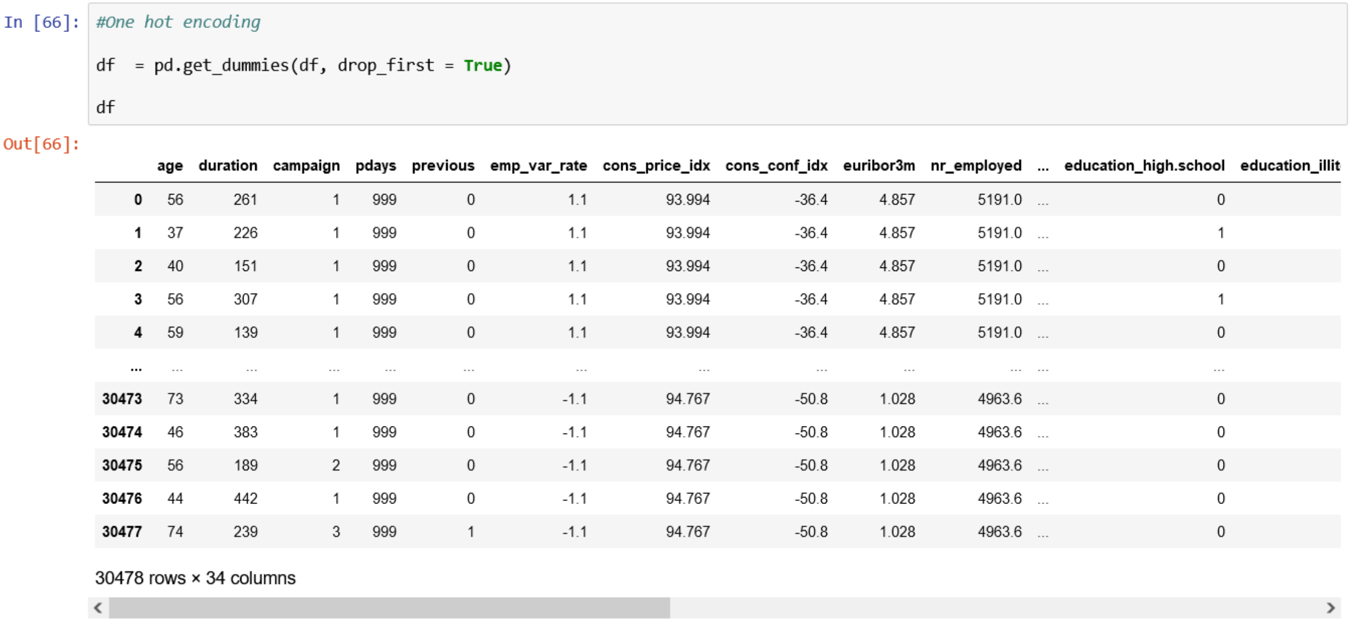
5.1.1 **Training the data**. In this process, 20% of the data was split for the test data and 80% of the data was taken as train data.

5.1.2 **Scaling the Data**. The data is not normally distributed. To avoid machine learning model disregarding coefficients of features with low values the minmax scaler was applied.



```
In [224]: X_train,X_test,y_train,y_test = train_test_split(X,y, random_state=42)

In [225]: X_train.shape, y_train.shape, X_test.shape, y_test.shape
Out[225]: ((22858, 33), (22858,), (7620, 33), (7620,))

In [226]: # MinMaxScaler Object
          scaler = MinMaxScaler()

          scaler.fit_transform(X_train)
```

5.1.3 Categorical Data Pre-processing the data



```
In [66]: #One hot encoding

         df  = pd.get_dummies(df, drop_first = True)

         df
```

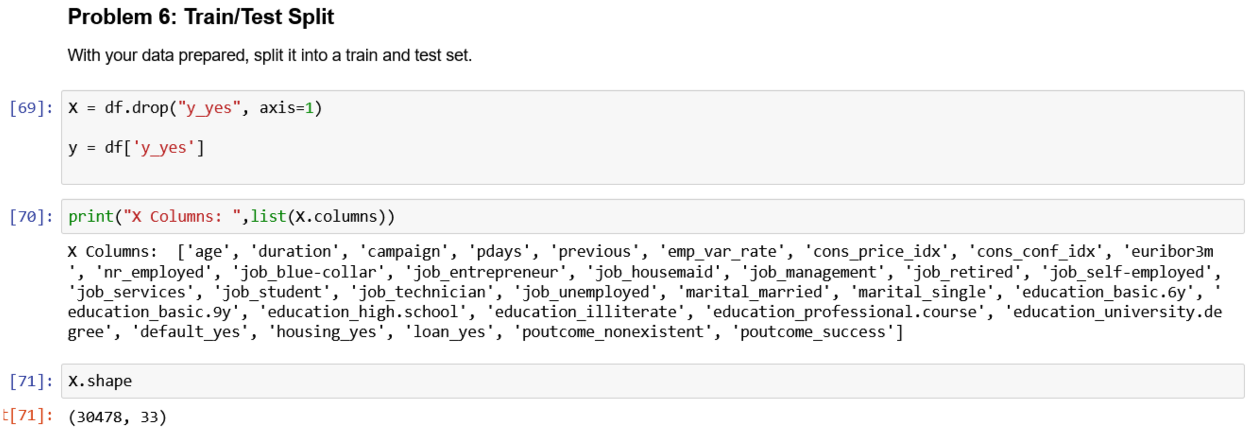
Out[66]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	...	education_high.school	education_illit
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	
1	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
2	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	
3	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
4	59	139	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
30473	73	334	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	...	0	
30474	46	383	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	...	0	
30475	56	189	2	999	0	-1.1	94.767	-50.8	1.028	4963.6	...	0	
30476	44	442	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	...	0	
30477	74	239	3	999	1	-1.1	94.767	-50.8	1.028	4963.6	...	0	

30478 rows × 34 columns

Figure 7 One Hot Encoding for Categorial Features

5.2 Test-Train Split and building a base model



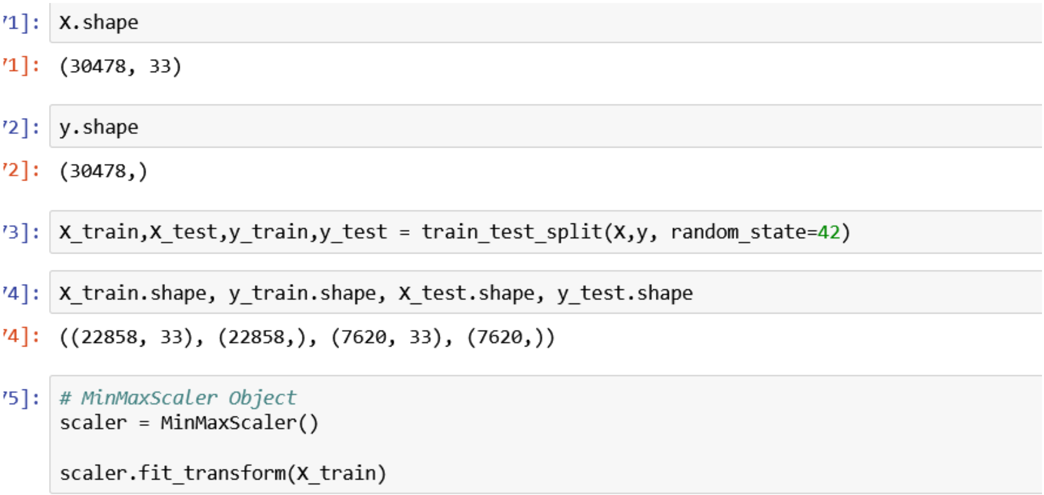
Problem 6: Train/Test Split

With your data prepared, split it into a train and test set.

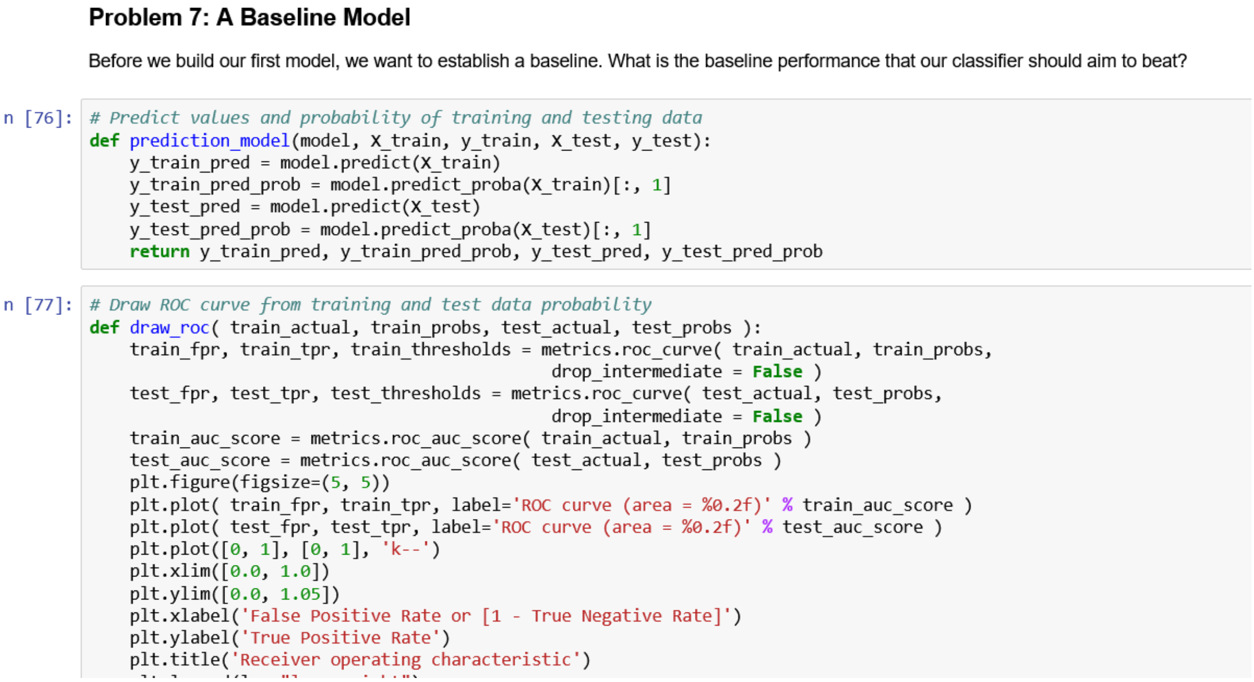
```
[69]: X = df.drop("y_yes", axis=1)

      y = df['y_yes']
```

```
[70]: print("X Columns: ",list(X.columns))

      X Columns:  ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euribor3m
      ', 'nr_employed', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',
      'job_services', 'job_student', 'job_technician', 'job_unemployed', 'marital_married', 'marital_single', 'education_basic.6y', '
      education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.course', 'education_university.de
      gree', 'default_yes', 'housing_yes', 'loan_yes', 'poutcome_nonexistent', 'poutcome_success']
```

```
[71]: X.shape

t[71]: (30478, 33)
```



```
’1]: X.shape
’1]: (30478, 33)

’2]: y.shape
’2]: (30478,)

’3]: X_train,X_test,y_train,y_test = train_test_split(X,y, random_state=42)

’4]: X_train.shape, y_train.shape, X_test.shape, y_test.shape
’4]: ((22858, 33), (22858,), (7620, 33), (7620,))

’5]: # MinMaxScaler Object
     scaler = MinMaxScaler()

     scaler.fit_transform(X_train)
```

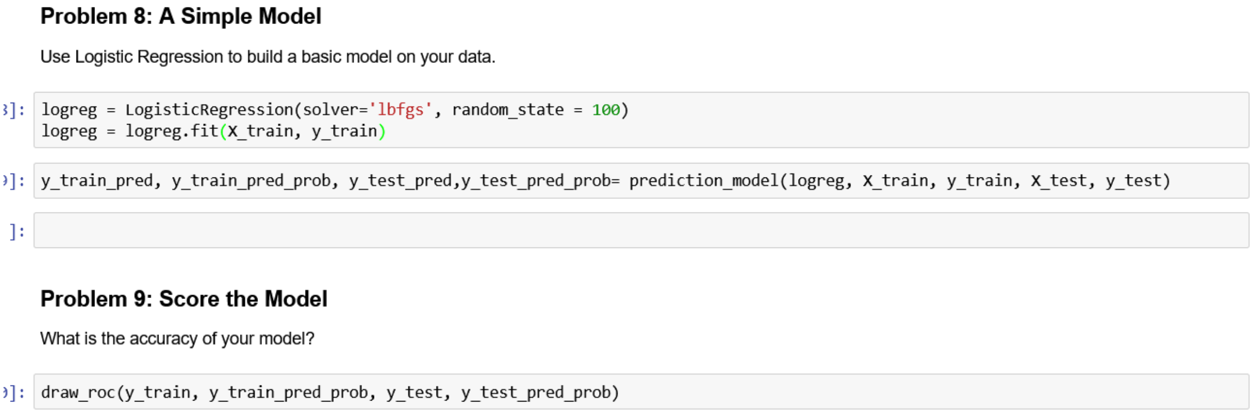


Problem 7: A Baseline Model

Before we build our first model, we want to establish a baseline. What is the baseline performance that our classifier should aim to beat?

```
n [76]: # Predict values and probability of training and testing data
        def prediction_model(model, X_train, y_train, X_test, y_test):
            y_train_pred = model.predict(X_train)
            y_train_pred_prob = model.predict_proba(X_train)[:, 1]
            y_test_pred = model.predict(X_test)
            y_test_pred_prob = model.predict_proba(X_test)[:, 1]
            return y_train_pred, y_train_pred_prob, y_test_pred, y_test_pred_prob
```

```
n [77]: # Draw ROC curve from training and test data probability
        def draw_roc( train_actual, train_probs, test_actual, test_probs ):
            train_fpr, train_tpr, train_thresholds = metrics.roc_curve( train_actual, train_probs,
                                                      drop_intermediate = False )
            test_fpr, test_tpr, test_thresholds = metrics.roc_curve( test_actual, test_probs,
                                                      drop_intermediate = False )
            train_auc_score = metrics.roc_auc_score( train_actual, train_probs )
            test_auc_score = metrics.roc_auc_score( test_actual, test_probs )
            plt.figure(figsize=(5, 5))
            plt.plot( train_fpr, train_tpr, label='ROC curve (area = %0.2f)' % train_auc_score )
            plt.plot( test_fpr, test_tpr, label='ROC curve (area = %0.2f)' % test_auc_score )
            plt.plot([0, 1], [0, 1], 'k--')
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.05])
            plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
            plt.ylabel('True Positive Rate')
            plt.title('Receiver operating characteristic')
```



Problem 8: A Simple Model

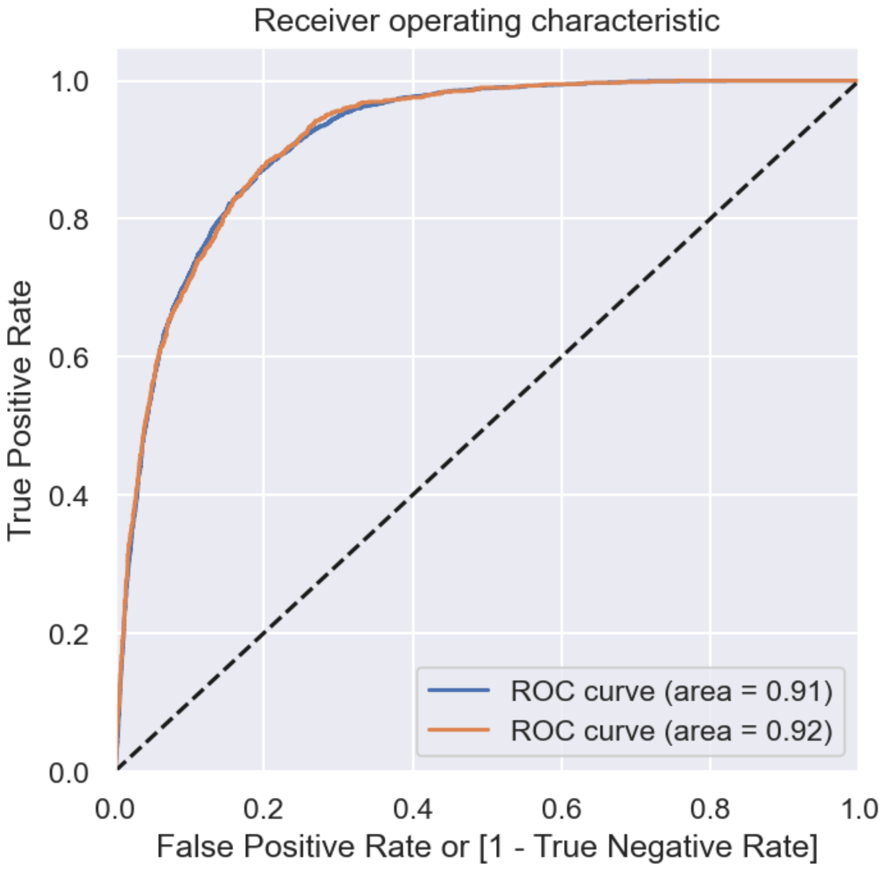
Use Logistic Regression to build a basic model on your data.

```
3]: logreg = LogisticRegression(solver='lbfgs', random_state = 100)
    logreg = logreg.fit(X_train, y_train)

9]: y_train_pred, y_train_pred_prob, y_test_pred,y_test_pred_prob= prediction_model(logreg, X_train, y_train, X_test, y_test)

 ]:
```

Problem 9: Score the Model

What is the accuracy of your model?

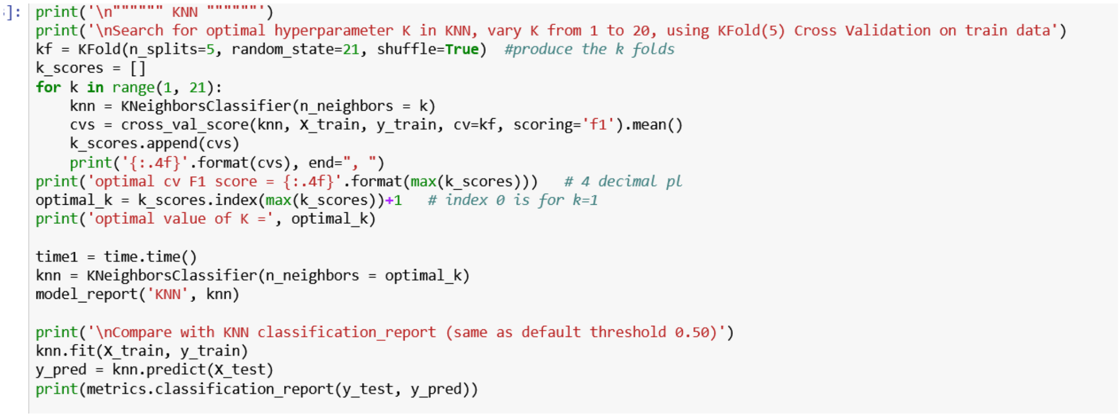
```
9]: draw_roc(y_train, y_train_pred_prob, y_test, y_test_pred_prob)
```

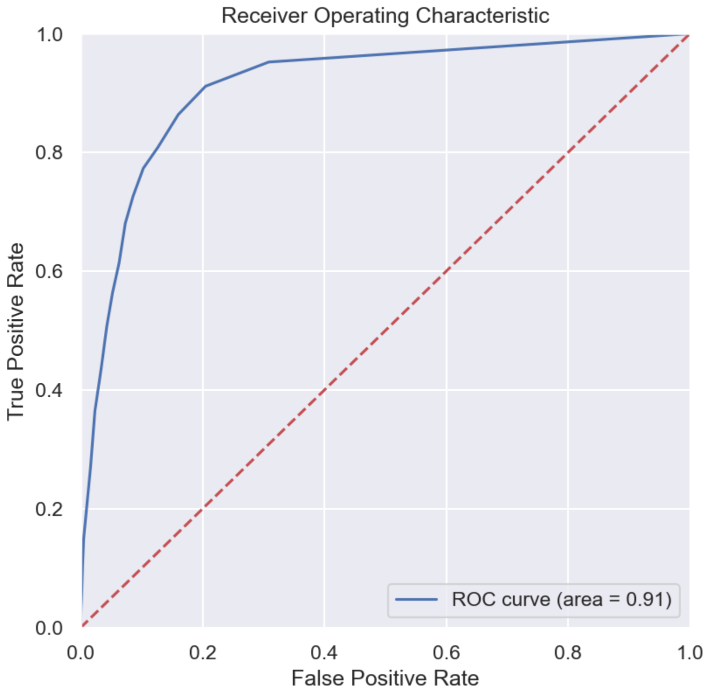


```
Accuracy train:  0.8981100708723423
Accuracy test:  0.89750656167979
```

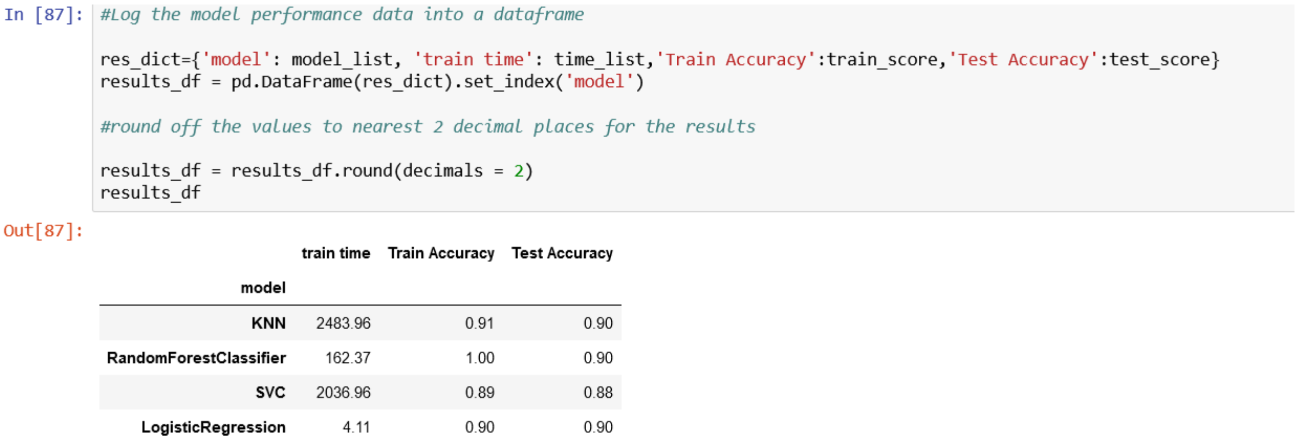
5.3 Models used to solve the problem

In this practical application, the goal is to compare the performance of K Nearest Neighbor, Logistic Regression, Decision Trees, and Support Vector Machine based classifiers. The models were built using sklearn and the data was transformed and scaled and the results were analyzed like so using a couple of methods: KNN example shown below:


```
]: print('\n"""""" KNN """""""')
   print('\nSearch for optimal hyperparameter K in KNN, vary K from 1 to 20, using KFold(5) Cross Validation on train data')
   kf = KFold(n_splits=5, random_state=21, shuffle=True)  #produce the k folds
   k_scores = []
   for k in range(1, 21):
       knn = KNeighborsClassifier(n_neighbors = k)
       cvs = cross_val_score(knn, X_train, y_train, cv=kf, scoring='f1').mean()
       k_scores.append(cvs)
       print('{:.4f}'.format(cvs), end=", ")
   print('optimal cv F1 score = {:.4f}'.format(max(k_scores)))   # 4 decimal pl
   optimal_k = k_scores.index(max(k_scores))+1   # index 0 is for k=1
   print('optimal value of K =', optimal_k)

   time1 = time.time()
   knn = KNeighborsClassifier(n_neighbors = optimal_k)
   model_report('KNN', knn)

   print('\nCompare with KNN classification_report (same as default threshold 0.50)')
   knn.fit(X_train, y_train)
   y_pred = knn.predict(X_test)
   print(metrics.classification_report(y_test, y_pred))
```



This step was repeated for each model and the results were stored in a dataframe as below:



```
In [87]: #Log the model performance data into a dataframe

         res_dict={'model': model_list, 'train time': time_list,'Train Accuracy':train_score,'Test Accuracy':test_score}
         results_df = pd.DataFrame(res_dict).set_index('model')

         #round off the values to nearest 2 decimal places for the results

         results_df = results_df.round(decimals = 2)
         results_df
```

Out[87]:

model	train time	Train Accuracy	Test Accuracy
KNN	2483.96	0.91	0.90
RandomForestClassifier	162.37	1.00	0.90
SVC	2036.96	0.89	0.88
LogisticRegression	4.11	0.90	0.90

5.4 Model Analysis

During the raw run of the model building exercise no parameters were tuned to get a rough idea on the best suited model for this problem domain. RandomForestClassifier and Logistic Regression performed well on the Test Dataset.

The ROC for each preliminary version of model was as follows:

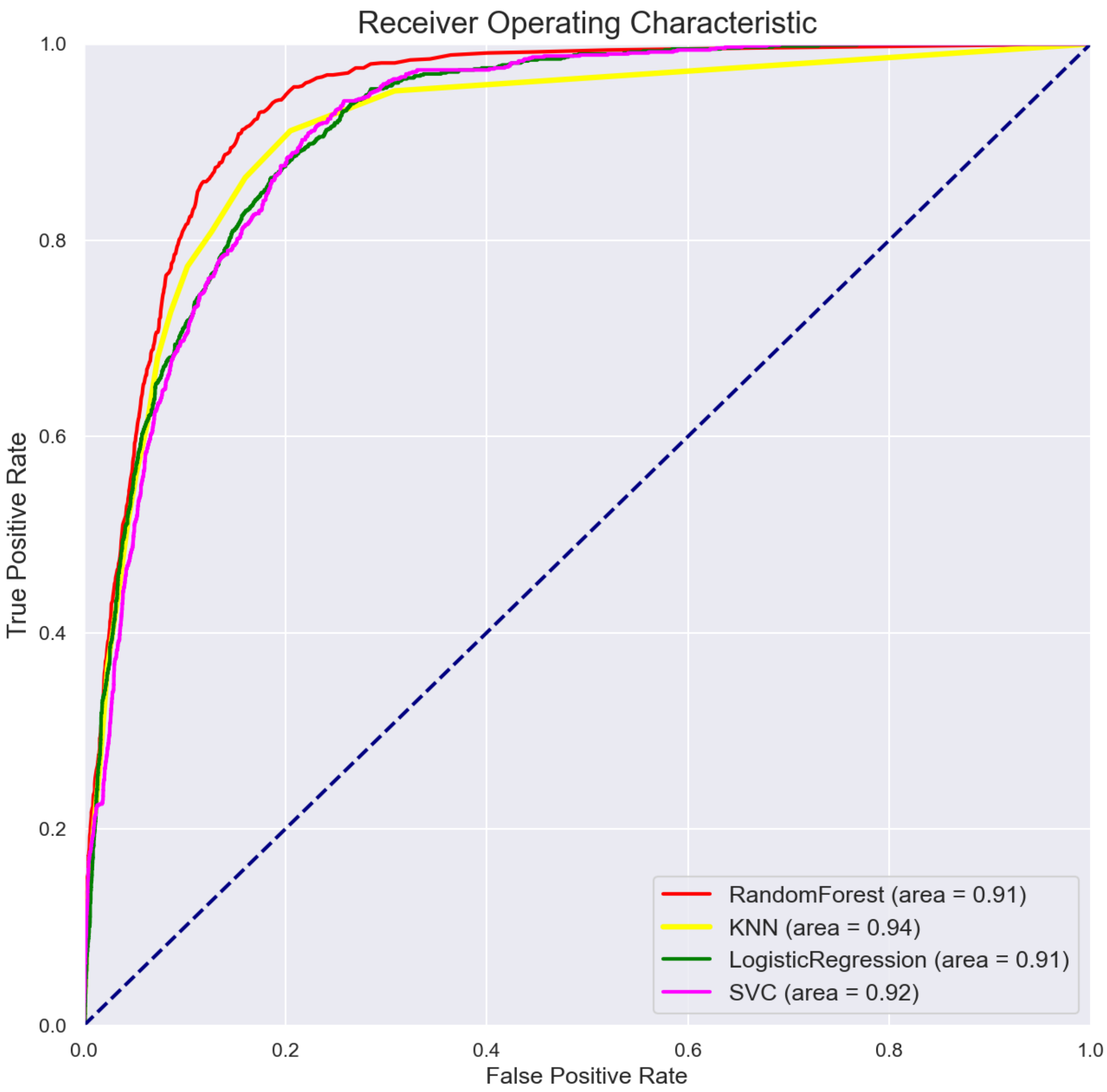
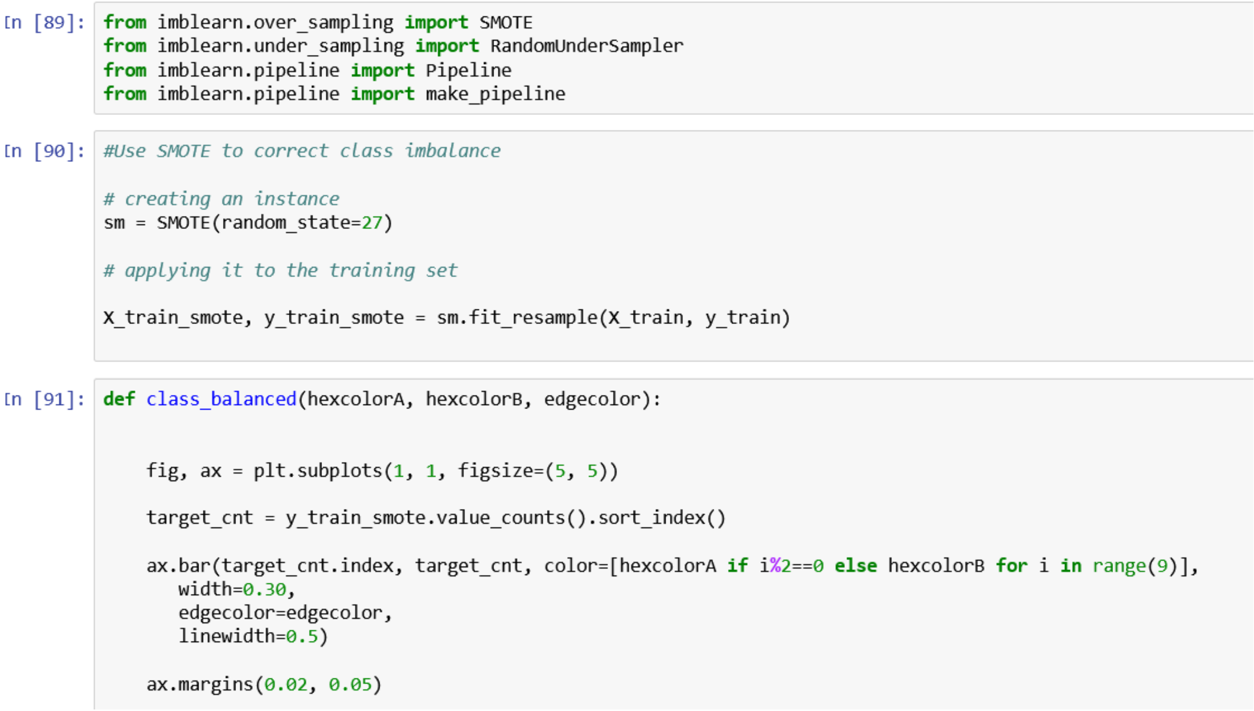


Figure 8 ROC for all the different Regression models

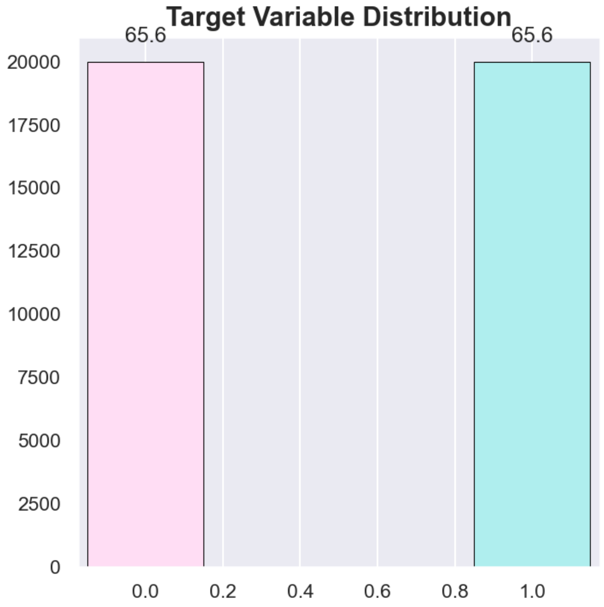
Since the number of features are very high and the parameter selection takes a lot of time even with all the cores allocated to GridSearch. After consulting the Scikit learn documentation I tried to improve the search performance by using an experimental feature called Searching for optimal parameters with successive halving[5].

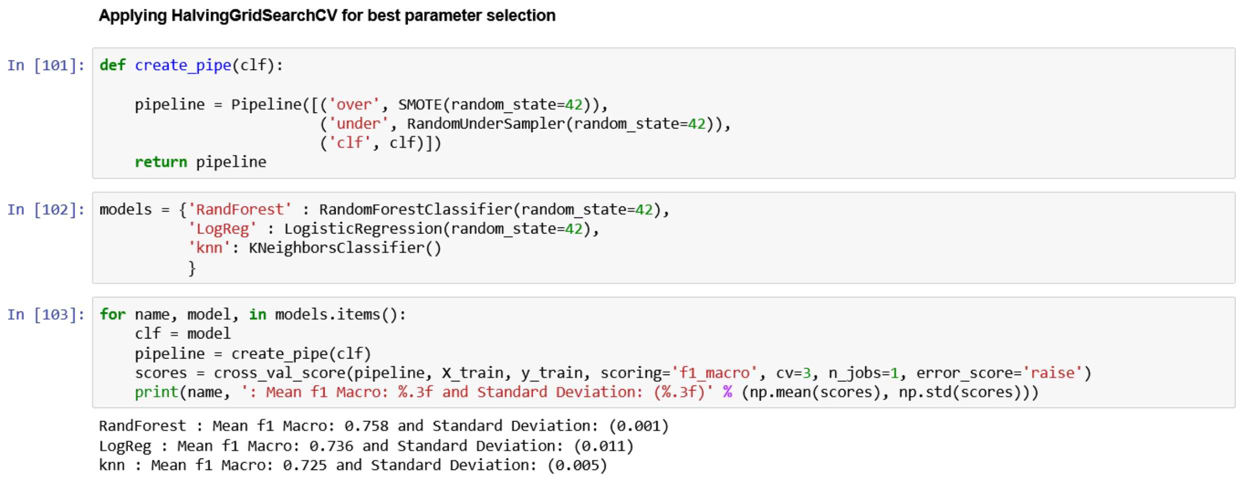
[5] Successive Halving GridSearch speed up: Reference: https://scikit-learn.org/stable/modules/grid_search.html

The Gridsearch saw a marked improvement in performance. Conventional Gridsearch took more than an hour, whereas the HalvingSearch crunched through under 5 minutes. Figure 9 shows the modified datapipeline along with balancing of the classes using SMOTE library.



```
In [89]: from imblearn.over_sampling import SMOTE
         from imblearn.under_sampling import RandomUnderSampler
         from imblearn.pipeline import Pipeline
         from imblearn.pipeline import make_pipeline

In [90]: #Use SMOTE to correct class imbalance

         # creating an instance
         sm = SMOTE(random_state=27)

         # applying it to the training set

         X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)

In [91]: def class_balanced(hexcolorA, hexcolorB, edgecolor):


             fig, ax = plt.subplots(1, 1, figsize=(5, 5))

             target_cnt = y_train_smote.value_counts().sort_index()

             ax.bar(target_cnt.index, target_cnt, color=[hexcolorA if i%2==0 else hexcolorB for i in range(9)],
                width=0.30,
                edgecolor=edgecolor,
                linewidth=0.5)

             ax.margins(0.02, 0.05)
```




Applying HalvingGridSearchCV for best parameter selection

```
In [101]: def create_pipe(clf):

              pipeline = Pipeline([('over', SMOTE(random_state=42)),
                                   ('under', RandomUnderSampler(random_state=42)),
                                   ('clf', clf)])
              return pipeline
```

```
In [102]: models = {'RandForest' : RandomForestClassifier(random_state=42),
                    'LogReg' : LogisticRegression(random_state=42),
                    'knn': KNeighborsClassifier()
                    }
```

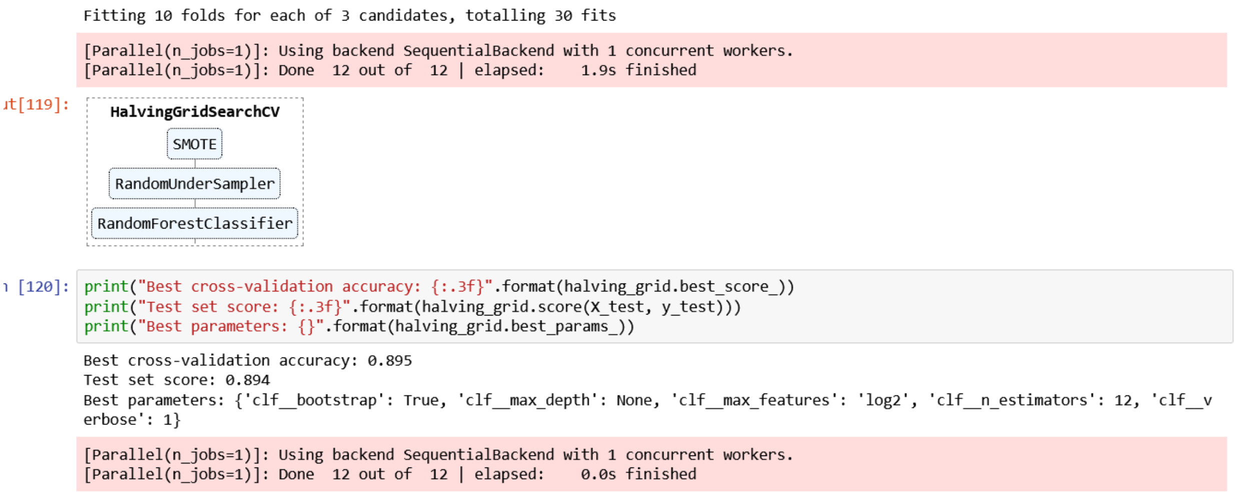
```
In [103]: for name, model, in models.items():
              clf = model
              pipeline = create_pipe(clf)
              scores = cross_val_score(pipeline, X_train, y_train, scoring='f1_macro', cv=3, n_jobs=1, error_score='raise')
              print(name, ': Mean f1 Macro: %.3f and Standard Deviation: (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
RandForest : Mean f1 Macro: 0.758 and Standard Deviation: (0.001)
LogReg : Mean f1 Macro: 0.736 and Standard Deviation: (0.011)
knn : Mean f1 Macro: 0.725 and Standard Deviation: (0.005)
```

Figure 9 Halving Grid Search for boosting model building performance

The RandomForest model was the best performing models. Its ROC performance was boosted from to which is in line with the results published in the original paper. The best ROC score for the RandomForest classifier from halving grid search was: 0.91. This is likely to improve with additional tweaks in the pipeline to the hyperparameters.

6 VALIDATION



```
Fitting 10 folds for each of 3 candidates, totalling 30 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    1.9s finished
```

ıt[119]:

HalvingGridSearchCV

SMOTE

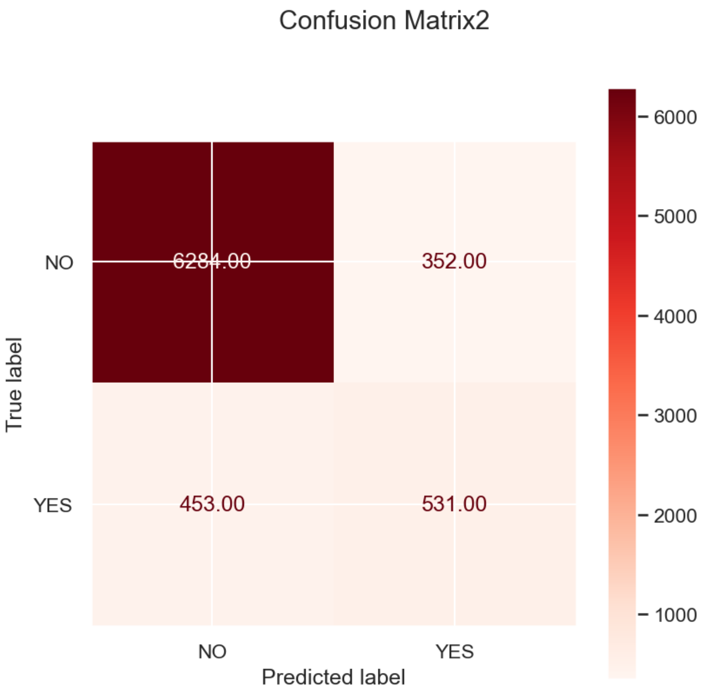
RandomUnderSampler

RandomForestClassifier

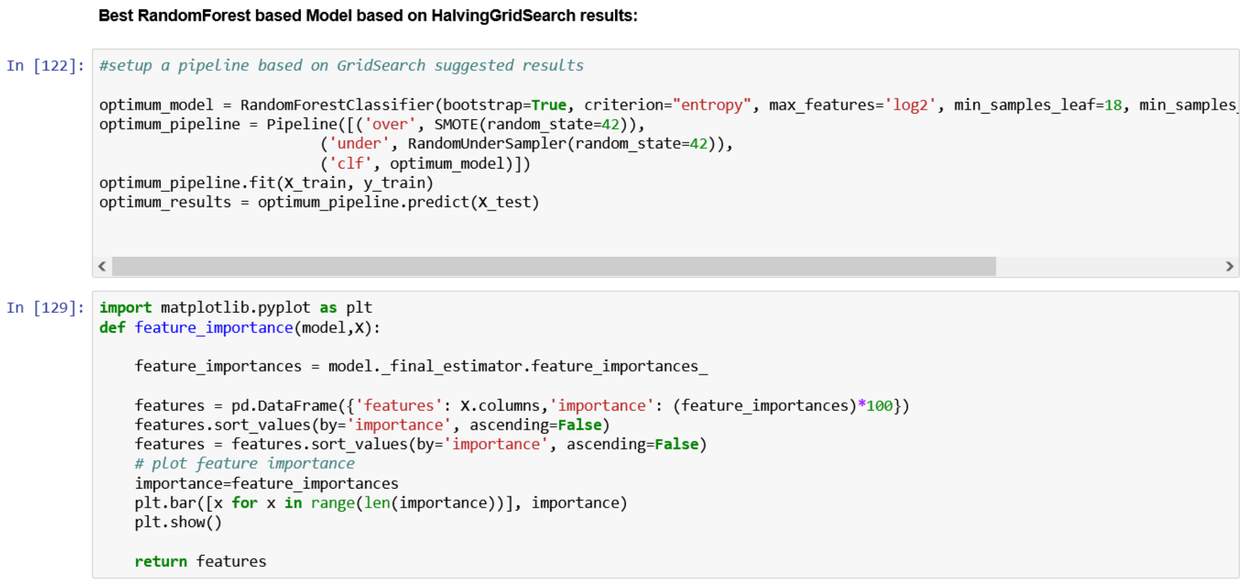
ı [120]:

```
print("Best cross-validation accuracy: {:.3f}".format(halving_grid.best_score_))
print("Test set score: {:.3f}".format(halving_grid.score(X_test, y_test)))
print("Best parameters: {}".format(halving_grid.best_params_))
```

```
Best cross-validation accuracy: 0.895
Test set score: 0.894
Best parameters: {'clf__bootstrap': True, 'clf__max_depth': None, 'clf__max_features': 'log2', 'clf__n_estimators': 12, 'clf__v
erbose': 1}
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.0s finished
```



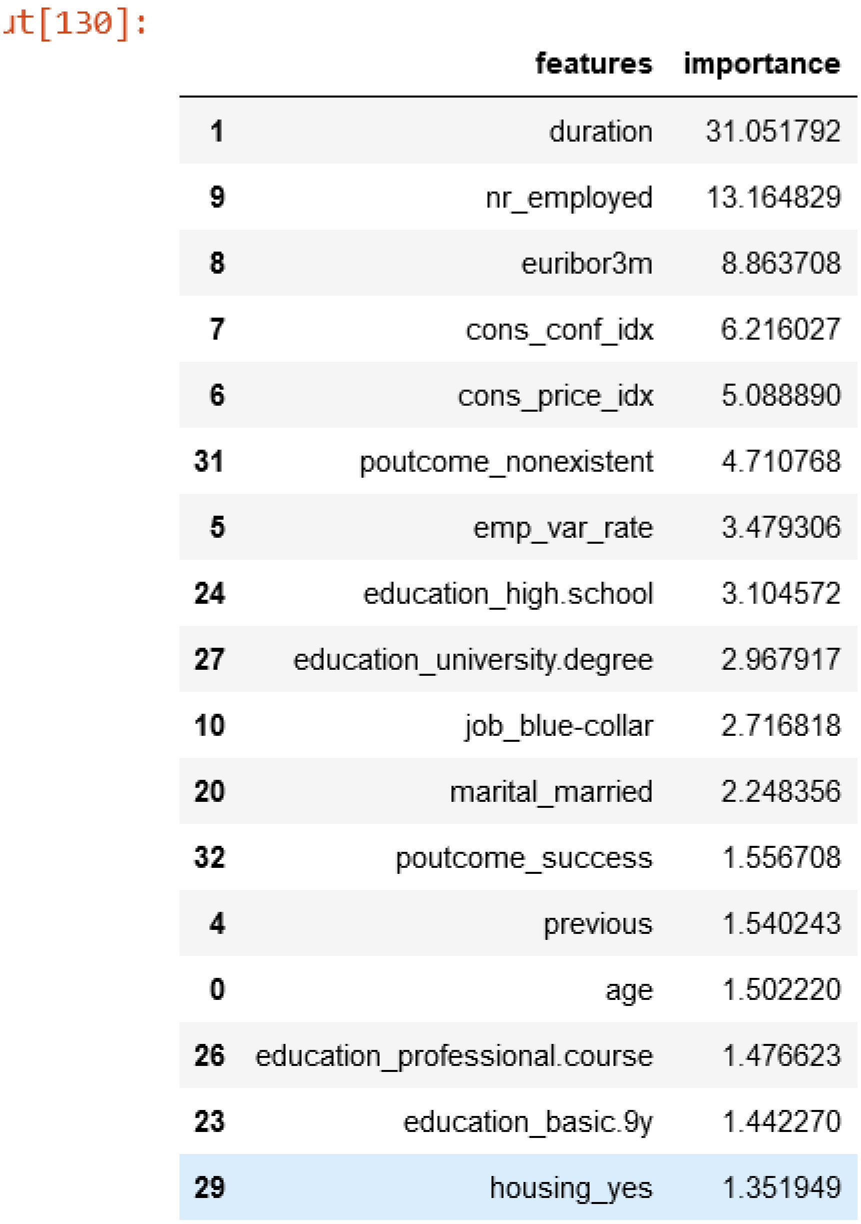
Confusion Matrix2



Best RandomForest based Model based on HalvingGridSearch results:

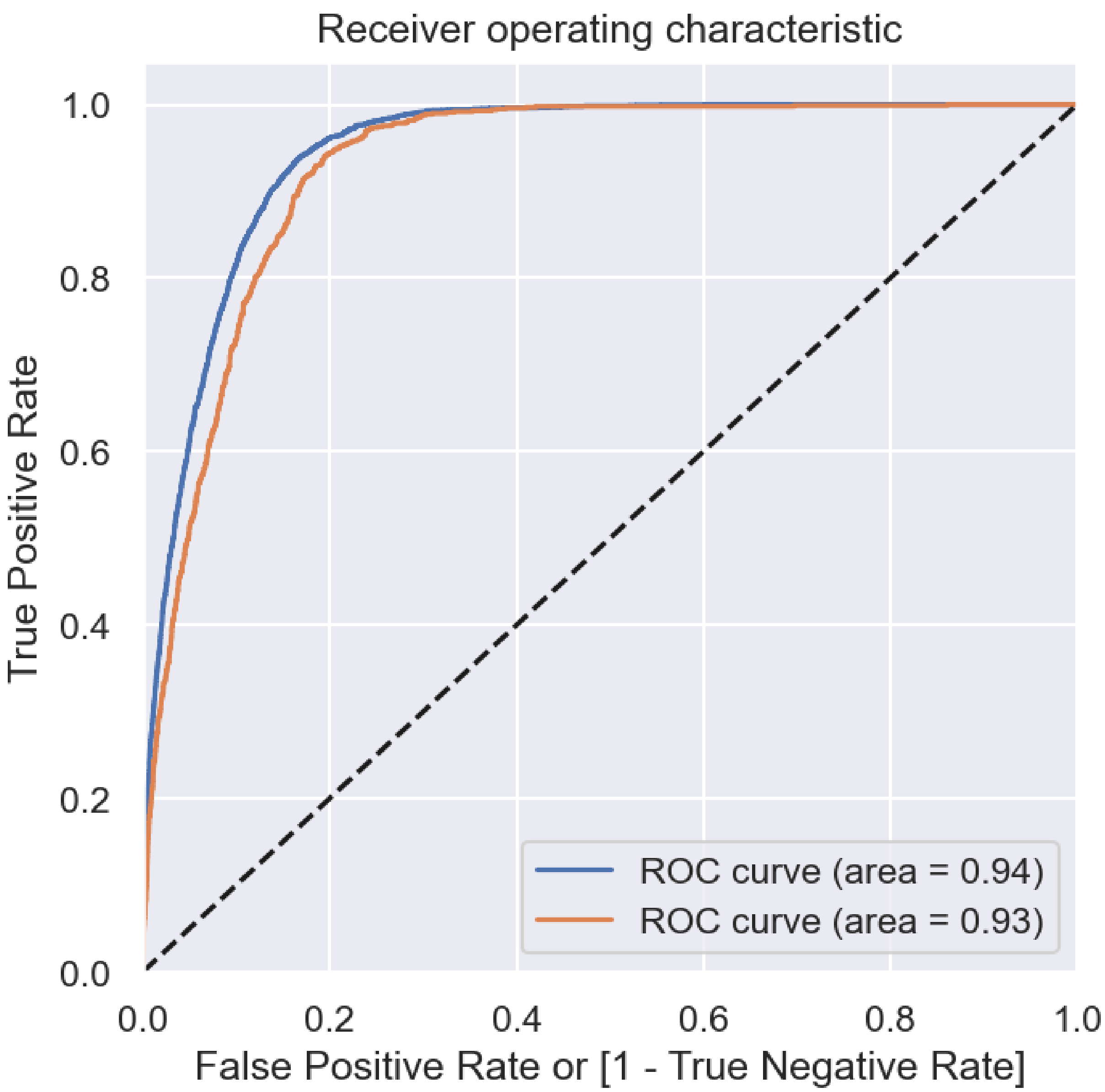
In [122]:
```
#setup a pipeline based on GridSearch suggested results

optimum_model = RandomForestClassifier(bootstrap=True, criterion="entropy", max_features='log2', min_samples_leaf=18, min_samples_
optimum_pipeline = Pipeline([('over', SMOTE(random_state=42)),
                         ('under', RandomUnderSampler(random_state=42)),
                         ('clf', optimum_model)])
optimum_pipeline.fit(X_train, y_train)
optimum_results = optimum_pipeline.predict(X_test)
```

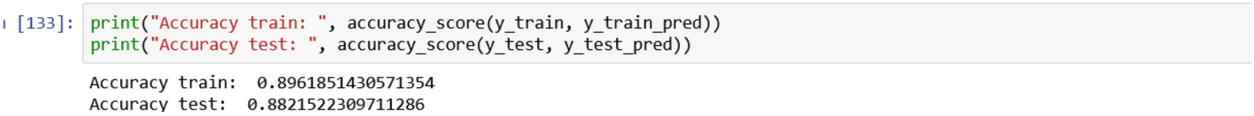
In [129]:
```
import matplotlib.pyplot as plt
def feature_importance(model,X):

    feature_importances = model._final_estimator.feature_importances_

    features = pd.DataFrame({'features': X.columns,'importance': (feature_importances)*100})
    features.sort_values(by='importance', ascending=False)
    features = features.sort_values(by='importance', ascending=False)
    # plot feature importance
    importance=feature_importances
    plt.bar([x for x in range(len(importance))], importance)
    plt.show()

    return features
```



ut[130]:

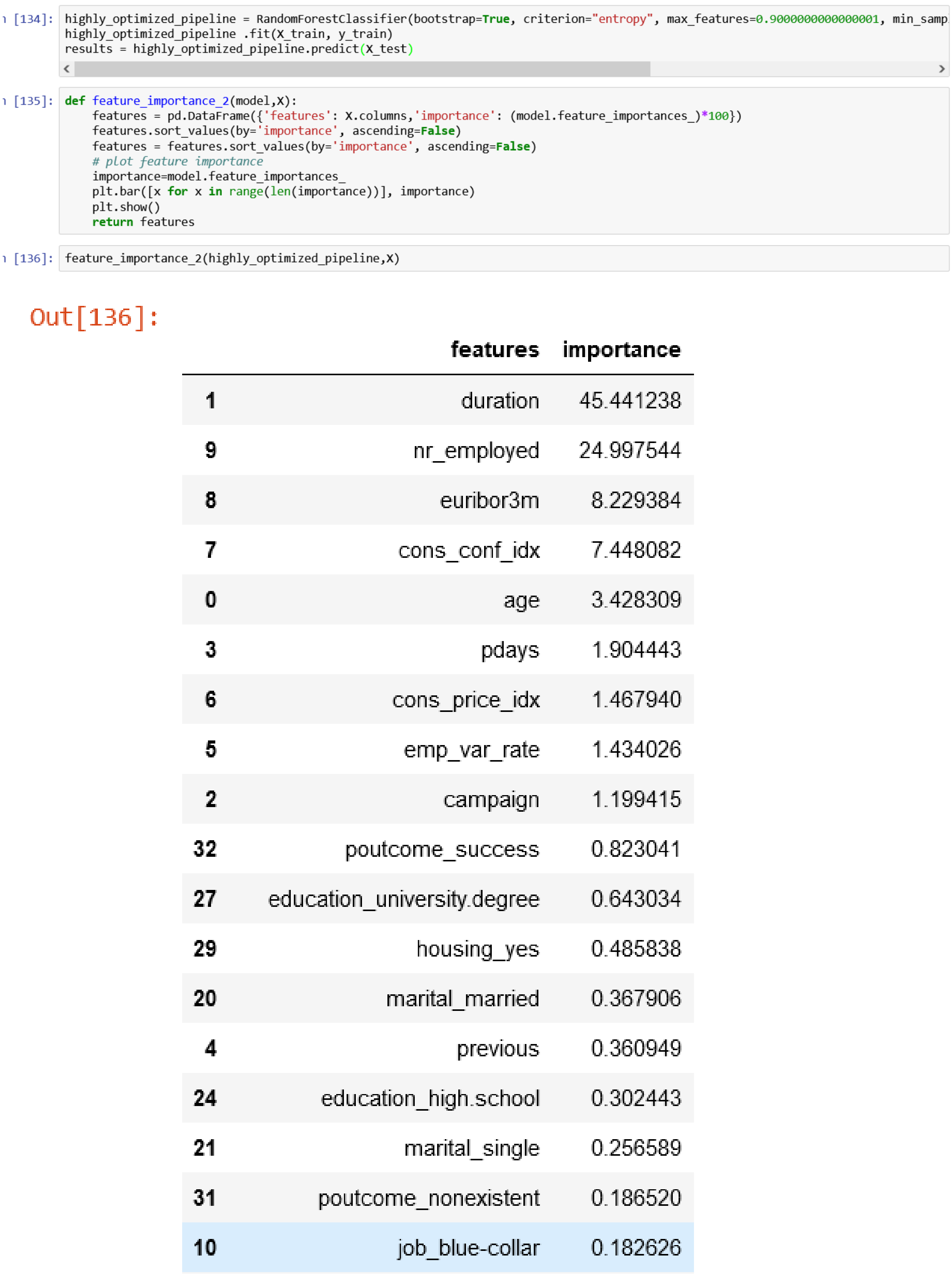
	features	importance
1	duration	31.051792
9	nr_employed	13.164829
8	euribor3m	8.863708
7	cons_conf_idx	6.216027
6	cons_price_idx	5.088890
31	poutcome_nonexistent	4.710768
5	emp_var_rate	3.479306
24	education_high.school	3.104572
27	education_university.degree	2.967917
10	job_blue-collar	2.716818
20	marital_married	2.248356
32	poutcome_success	1.556708
4	previous	1.540243
0	age	1.502220
26	education_professional.course	1.476623
23	education_basic.9y	1.442270
29	housing_yes	1.351949





```
[133]: print("Accuracy train: ", accuracy_score(y_train, y_train_pred))
       print("Accuracy test: ", accuracy_score(y_test, y_test_pred))

       Accuracy train:  0.8961851430571354
       Accuracy test:  0.8821522309711286
```

Finally using the Gridsearch recommended parameters as a base, the model was manually tuned to create a highly optimized version which has the following pipeline.



```
[134]: highly_optimized_pipeline = RandomForestClassifier(bootstrap=True, criterion="entropy", max_features=0.9000000000000001, min_samp
       highly_optimized_pipeline .fit(X_train, y_train)
       results = highly_optimized_pipeline.predict(X_test)
```

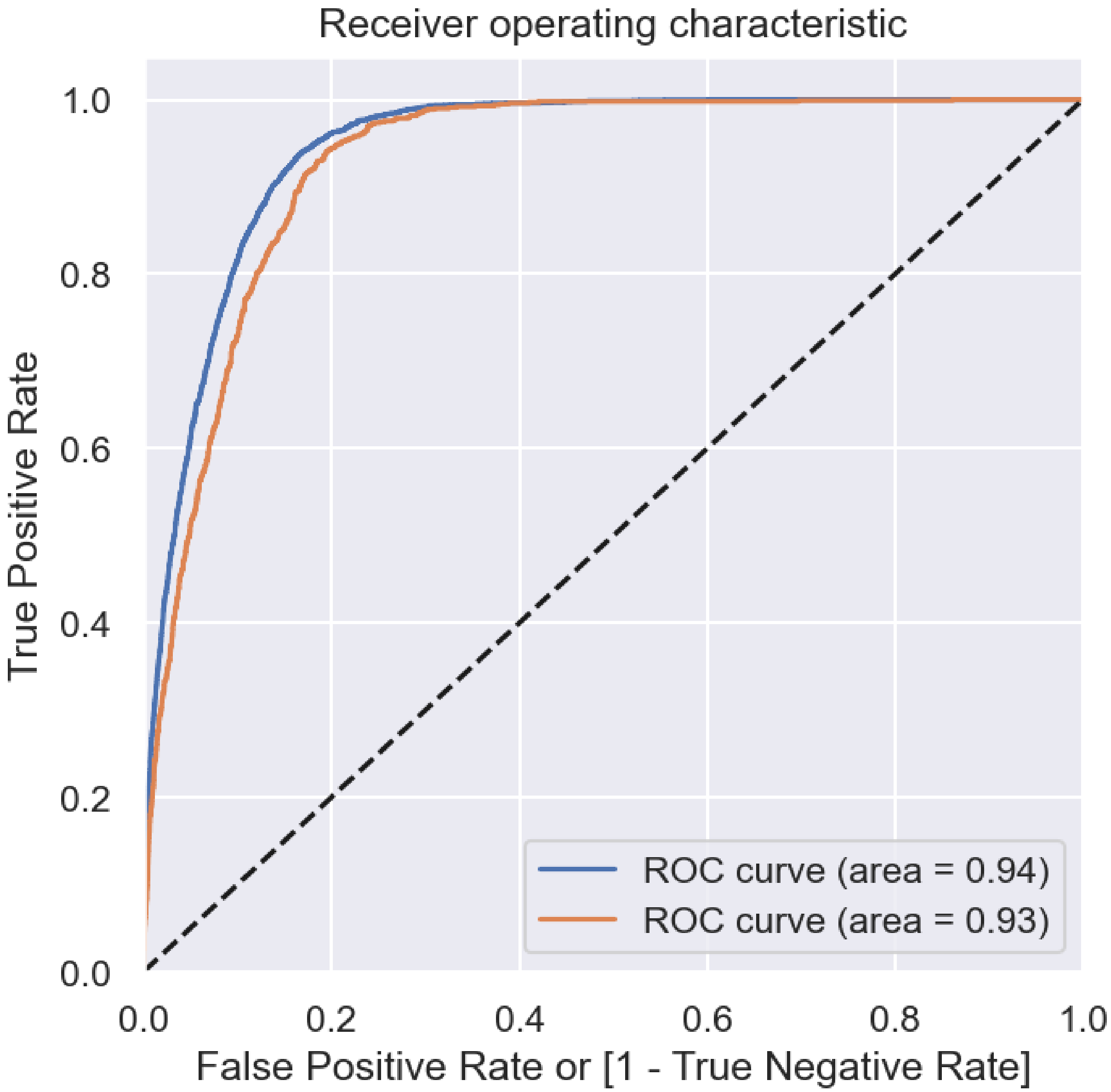
```
[135]: def feature_importance_2(model,X):
           features = pd.DataFrame({'features': X.columns,'importance': (model.feature_importances_)*100})
           features.sort_values(by='importance', ascending=False)
           features = features.sort_values(by='importance', ascending=False)
           # plot feature importance
           importance=model.feature_importances_
           plt.bar([x for x in range(len(importance))], importance)
           plt.show()
           return features
```

```
[136]: feature_importance_2(highly_optimized_pipeline,X)
```

Out[136]:

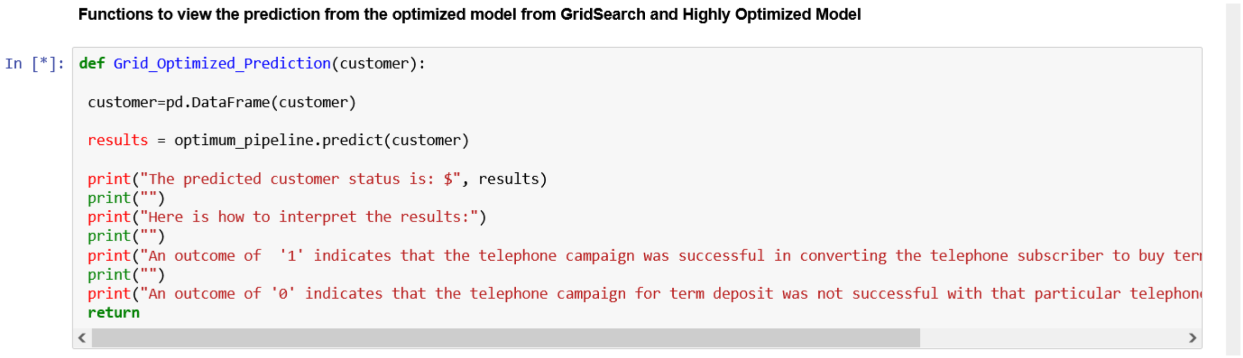
	features	importance
1	duration	45.441238
9	nr_employed	24.997544
8	euribor3m	8.229384
7	cons_conf_idx	7.448082
0	age	3.428309
3	pdays	1.904443
6	cons_price_idx	1.467940
5	emp_var_rate	1.434026
2	campaign	1.199415
32	poutcome_success	0.823041
27	education_university.degree	0.643034
29	housing_yes	0.485838
20	marital_married	0.367906
4	previous	0.360949
24	education_high.school	0.302443
21	marital_single	0.256589
31	poutcome_nonexistent	0.186520
10	job_blue-collar	0.182626

The ROC was further optimized with this last version of the model.



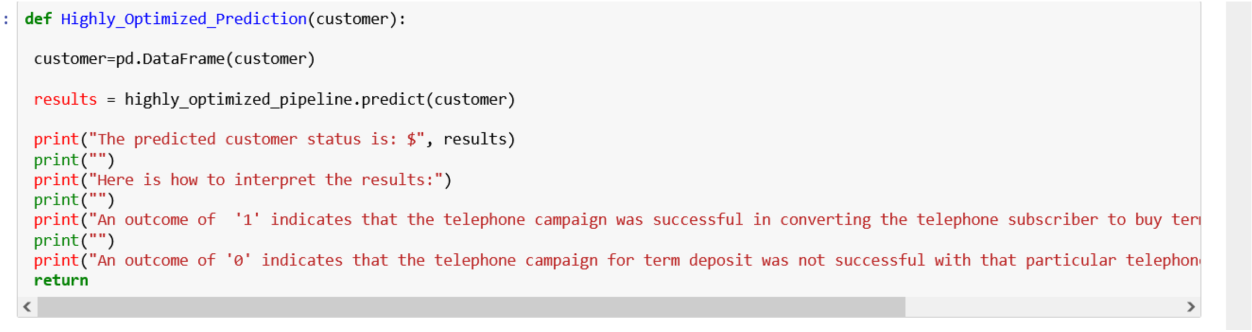
6.1 Predictions

A couple of back-end functions were developed to predict the results using the random forest classification model. This backend function can be called by a web app to retrieve and present the classification decision to the telemarketer.



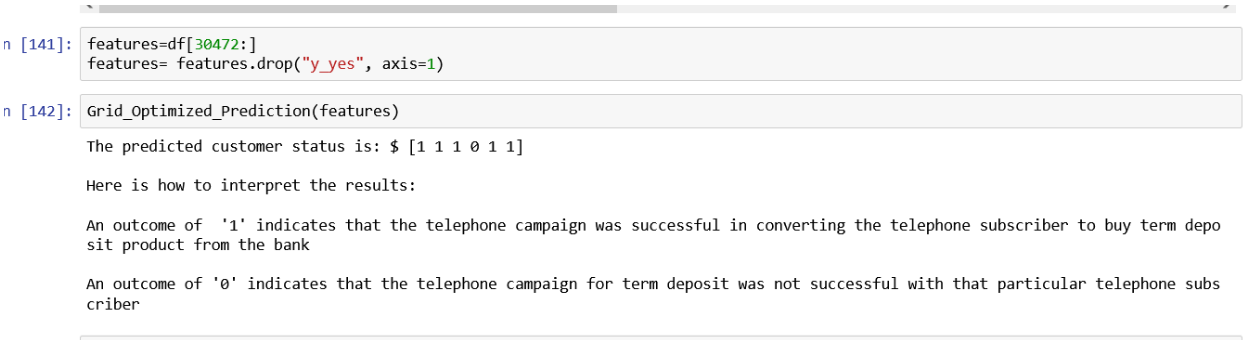
Functions to view the prediction from the optimized model from GridSearch and Highly Optimized Model

```
In [*]: def Grid_Optimized_Prediction(customer):

     customer=pd.DataFrame(customer)

     results = optimum_pipeline.predict(customer)

     print("The predicted customer status is: $", results)
     print("")
     print("Here is how to interpret the results:")
     print("")
     print("An outcome of  '1' indicates that the telephone campaign was successful in converting the telephone subscriber to buy tern
     print("")
     print("An outcome of '0' indicates that the telephone campaign for term deposit was not successful with that particular telephon
     return
```

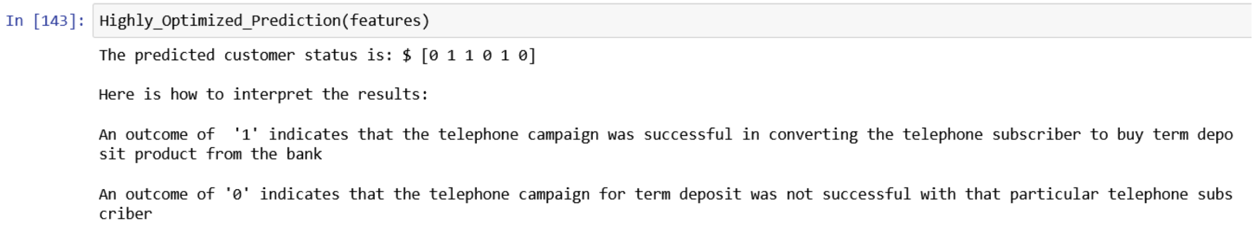


```
: def Highly_Optimized_Prediction(customer):

     customer=pd.DataFrame(customer)

     results = highly_optimized_pipeline.predict(customer)

     print("The predicted customer status is: $", results)
     print("")
     print("Here is how to interpret the results:")
     print("")
     print("An outcome of  '1' indicates that the telephone campaign was successful in converting the telephone subscriber to buy tern
     print("")
     print("An outcome of '0' indicates that the telephone campaign for term deposit was not successful with that particular telephon
     return
```

The following predictions are made by method when invoked using the jupyter notebook.



```
n [141]: features=df[30472:]
         features= features.drop("y_yes", axis=1)

n [142]: Grid_Optimized_Prediction(features)

         The predicted customer status is: $ [1 1 1 0 1 1]

         Here is how to interpret the results:

         An outcome of  '1' indicates that the telephone campaign was successful in converting the telephone subscriber to buy term depo
         sit product from the bank

         An outcome of '0' indicates that the telephone campaign for term deposit was not successful with that particular telephone subs
         criber
```



```
In [143]: Highly_Optimized_Prediction(features)

          The predicted customer status is: $ [0 1 1 0 1 0]

          Here is how to interpret the results:

          An outcome of  '1' indicates that the telephone campaign was successful in converting the telephone subscriber to buy term depo
          sit product from the bank

          An outcome of '0' indicates that the telephone campaign for term deposit was not successful with that particular telephone subs
          criber
```

Ideally this would be the model part of a model-view-controller webapp. The classification results using different customer criteria show that the model is working as per the current model performance.

7 Result Summary:

We can see that the last model is pretty accurate. It got one prediction wrong which is in line with the accuracy rate on test data and ROC characteristics

Findings and Actionable Insights:

1. The objective of this case study was to come up with an optimum classification model to predict whether a customer will subscribe to a term deposit from the bank or not given the customer data.
2. There were a lot of categorical variables and some numerical variables which capture the various attributed of a typical customer at this Portuguese bank.
3. Exploratory data analysis showed absence of null values in the dataset, and the data is imbalanced, where “no” is the majority class.
4. Univariate analysis revealed that Grid Search does not help very much when it comes to predicting the target variable. Some numerical features tend to predict the target variable much better (for example: nr_employed, euribor rate etc.)
5. Dataset preprocessing of Categorical data was done using 1 hot encoding method.
6. Basic models were built using K Nearest Neighbor, Logistic Regression, Decision Trees, and Support Vector Machines. The most important features in predicting whether any customer will open a term deposit based on the Random forest model was duration, nr.employed, emp.var.rate, poutcome_success, euribor3m
7. GridSearch was used to find the best parameters. Random Forest model gave the best performance with Halving GridSearchCV and the best test AUC was 0.94 which was similar to the results the researchers got in the relevant research paper.

8 CONCLUSION:

Optimized targeting for telemarketing is a key issue in most industries. CRM data is a valuable and can be made more effective. Sales organizations can improve their market performance by analyzing customer data to design an efficient direct marketing campaign by improving customer experience during a sales call. The objective can be met with an approach to predict the success of telemarketing calls using machine learning classification techniques. A good customer classification model can increase campaign efficiency while reducing capital and operational expenditure through better management of the available resources (e.g., human effort, phone calls, time) via intelligent targeting of potential customers. In this project the research paper results were closely replicated using a Random Forest model that gave the best performance with Halving GridSearchCV and the best test AUC of 0.94 which was similar to the results the researchers got in the relevant research paper. In a way I was able to validate the efficiency of data mining techniques in intelligent targeted telemarketing.

9 FINDINGS AND ACTIONABLE INSIGHTS AND SCOPE FOR FUTURE IMPROVEMENT:

Findings and Actionable insights:

3. Exploratory data analysis showed absence of null values in the dataset, and the data is imbalanced, where “no” is the majority class.

4. Univariate analysis revealed that day_of_week does not help very much when it comes to predicting the target variable. Some numerical features tend to predict the target variable much better (for example: nr_employed, euribor rate etc.)

5. Dataset preprocessing of Categorical data was done using 1 hot encoding method.

6. Basic models were built using K Nearest Neighbor, Logistic Regression, Decision Trees, and Support Vector Machines.

The most important features in predicting whether any customer will open a term deposit based on the Random forest model was duration, nr.employed, emp.var.rate, poutcome_success, euribor3m

7. GridSearch was used to find the best parameters. Random Forest model gave the best performance with Halving GridSearchCV and the best test AUC was 0.94 which was similar to the results the researchers got in the relevent research paper.

Recommendations/NextSteps:

1. We could reduce the dimensions/features further to tune the model.

2. Random forest training even with HalvingGridSearch was very slow.

3. There are some new libraries like T-POT https://epistasislab.github.io/tpot/using/ that use genetic algorithms for hyperparameter tuning to derive the best pipeline for this classification problem. For best feature selection we could use a library like YellowBrick https://www.scikit-yb.org/en/latest/api/model_selection/importances.html

4. Ensemble models, XGBoost, Neural Networks/Autoencoders are other models that could be tried for this class of problems

10 REFERENCES:

1. http://www.impacttargetmarketing.com/telemarketing-good-or-bad-its-still-effective/

2. http://www.bls.gov/oes/current/oes419041.htm

3. http://www.dianamey.com/telemarketing-statistics/

4. http://acma.gov.au/theACMA/ACMAi/Investigation-reports/Statistics/telemarketing-statistics