CLASSIFICATION PROJECT REPORT

Group No – 21

Group Members – Nikhil Gaikwad, Sourav Gajbhiye

Topic – Classification

Title – Determining whether a client has subscribed the term deposit or not at a banking institution in response to the marketing done by the bank.

Dataset Description

The data is related with direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Input variables:

Bank Client Data:

- age (numeric)
- job: type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
- marital: marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)
- education (categorical: ‘primary’, ‘secondary’, ‘tertiary’, 'unknown')
- default: has credit in default? (categorical: 'no', 'yes', 'unknown')
- housing: has housing loan? (categorical: 'no', 'yes', 'unknown')
- loan: has personal loan? (categorical: 'no', 'yes', 'unknown')
- related with the last contact of the current campaign:
- contact: contact communication type (categorical: 'cellular', 'telephone')
- month: last contact month of year (categorical: 'jan', 'feb', 'mar’..., 'nov', 'dec')
- day: last contact day of the week (categorical: 'mon', 'tue', 'wed', 'thu', 'fri')
- 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:

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

- previous: number of contacts performed before this campaign and for this client (numeric)
- poutcome: outcome of the previous marketing campaign
(categorical: 'failure', 'non-existent', 'success'.

Output variable (desired target):

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

Data Pre-processing

```
import pandas as  pd
import numpy as np
# model metrics
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score
# import models
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import AdaBoostClassifier
# import Visualization
import seaborn as sns
import matplotlib.pyplot as plt

```

Importing all the required libraries in one go. As, the above snippet shows all the required libraries like numpy, pandas for data manipulation, seaborn and Matplotlib for visualization purpose and sklearn library for all the machine learning models.

1. Removing Outliers:

```
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name]
                                                        < fence_high)]
    return df_out

remove_outlier(df,'age')
remove_outlier(df,'balance')
remove_outlier(df,'day')
remove_outlier(df,'duration')
remove_outlier(df,'campaign')
remove_outlier(df,'p_days')
remove_outlier(df,'previous')
```

Out[13]:

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	ca

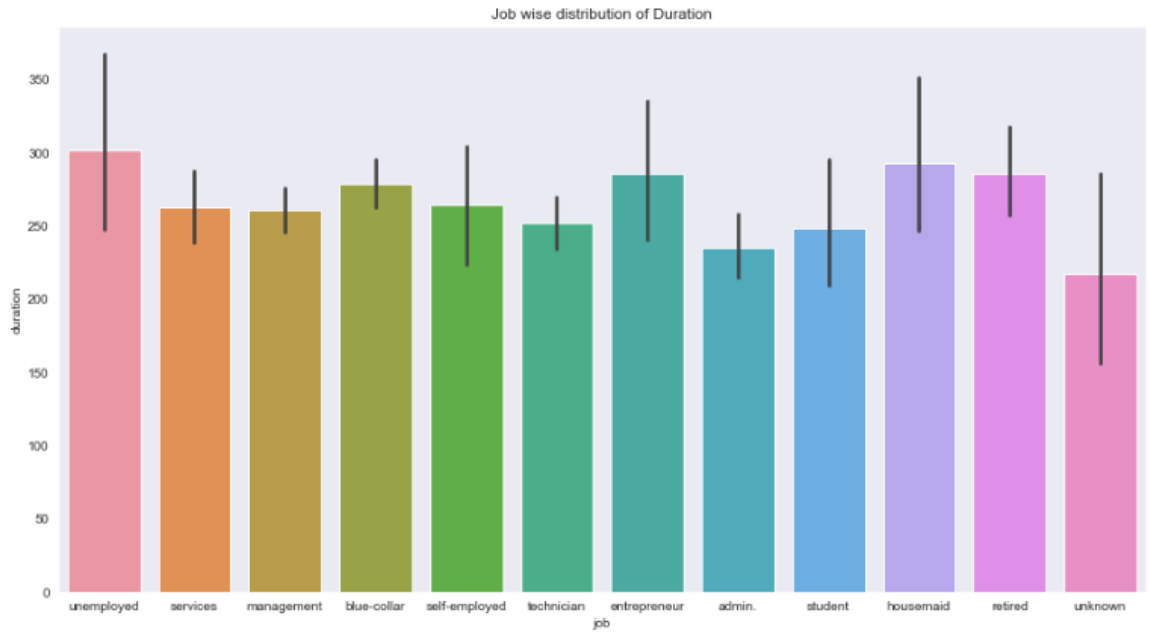
Trying to remove outliers using the Inter quartile range by creating a python function for it but it seems like there were no outliers in the dataset, still we tried to clear out all the outliers that may have been there in all the numeric columns.

2. Call duration wise distribution of Jobs and its Box-Plot

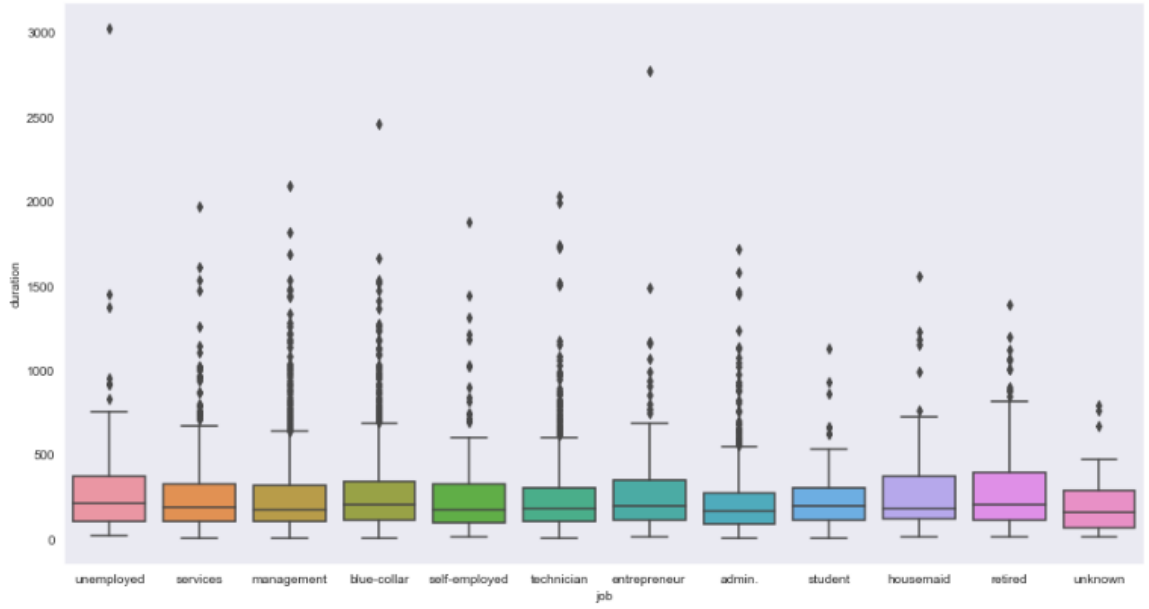
```
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')

# Bar plot for Job wise distribution of Duration

sns.barplot(x='job',y='duration',data=df[['job','duration']],ax=ax)
ax.set_title('Job wise distribution of Duration')
plt.show()
```



```
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
sns.boxplot(x="job", y="duration", data=df)
plt.show()
```



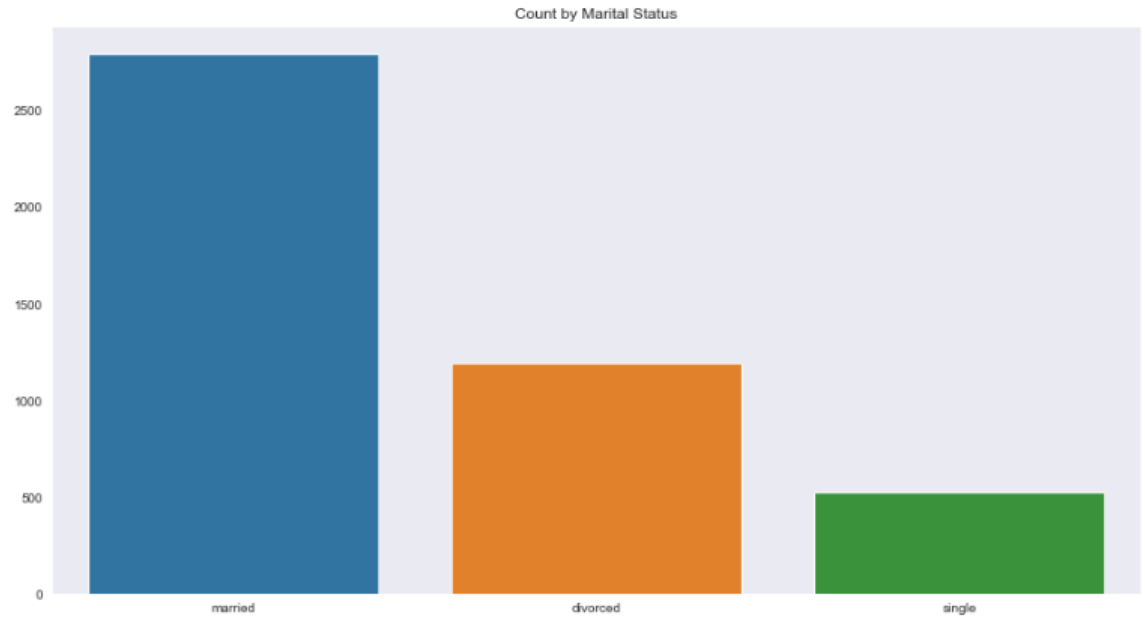
This plot gives the distribution of job types according to the duration of last call made to them. Type of job:

- *admin,*
- *blue-collar,*
- *entrepreneur,*
- *housemaid,*
- *management,*
- *retired,*
- *self-employed,*
- *services,*
- *student,*
- *technician,*
- *unemployed,*
- *unknown*

It seems that Unemployed and Housemaids were the two groups who took maximum of the duration of calls done by the bank officials.

3. Count of Marital Status in the dataset

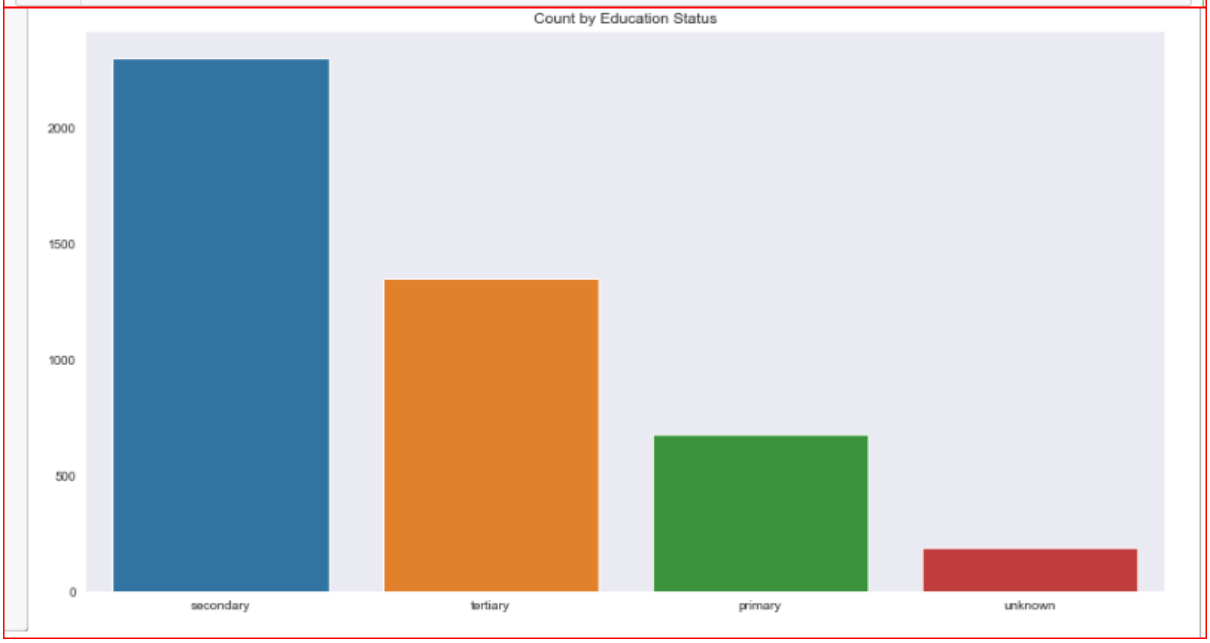
```
##Ploting the count of marital and education
values = df['marital'].value_counts().tolist()
labels = ['married', 'divorced', 'single']
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
sns.barplot(x=labels, y=values, data=df)
plt.title("Count by Marital Status")
plt.show()
```



This plot shows the number of count of people falling under the category of “Married”, “Divorced” and “Single”. Which gives us the idea that maximum number of people present in the bank database falls under the category of “Married”.

4. Count of Education categories

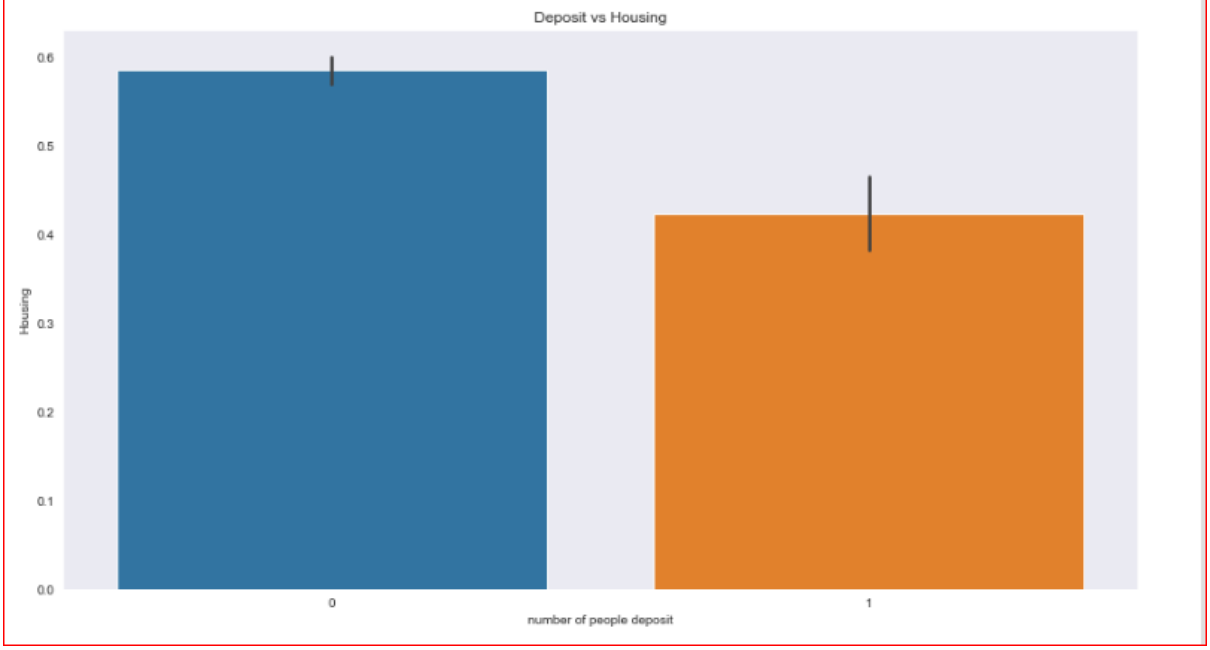
```
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
val = df['education'].value_counts().tolist()
lab = ['secondary', 'tertiary', 'primary', 'unknown']
sns.barplot(x=lab, y=val, data=df)
plt.title("Count by Education Status")
plt.show()
```



This plot shows the number of count of people falling under the category of “Primary”, “Secondary”, “Tertiary” and “Unknown” in the education column. Which gives us the idea that maximum number of people present in the bank database falls under the category of “Secondary” type of education.

5. Count of Housing Deposit

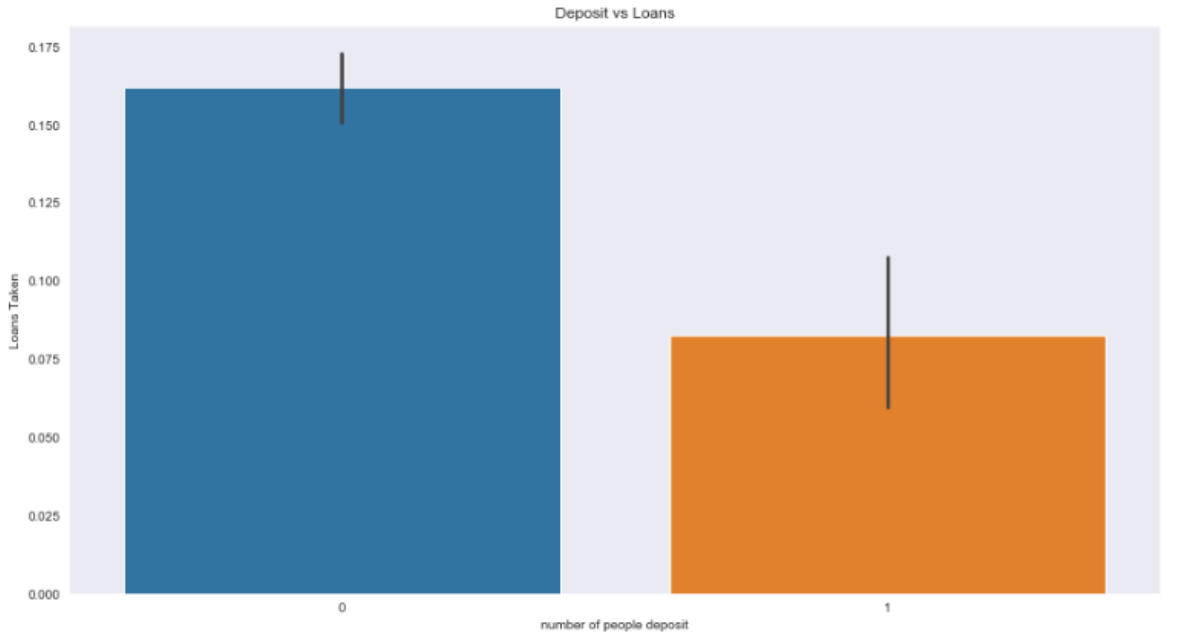
```
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
sns.barplot(x=deposit, y=housing, data= df)
plt.xlabel("number of people deposit")
plt.ylabel("Housing")
plt.title("Deposit vs Housing")
plt.show()
```



This plot between the housing loan and the Deposit show us clearly that how many people have deposited in terms of housing loan.

6. Plot between Deposit and Loan

```
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
sns.barplot(x=deposit, y=loan, data= df)
plt.xlabel("number of people deposit")
plt.ylabel("Loans Taken")
plt.title("Deposit vs Loans")
plt.show()
```



This plot gives us the relationship between people who have loans to pay with the deposit. Above plot has clearly shown that 83% has not deposited and 17% remaining have deposits which gives a weight to output.

Model Building

```
X = df.iloc[:, 0:16]
Y = df.iloc[:, 16]
```

```
#Splitting the catagorical variable to 0s and 1s using LabelEncoder
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

LabelEncoder().fit_transform(['no','yes'])
housing = LabelEncoder().fit_transform(X['housing'])
loan = LabelEncoder().fit_transform(X['loan'])
deposit = LabelEncoder().fit_transform(Y)
job = LabelEncoder().fit_transform(X['job'])
marital = LabelEncoder().fit_transform(X['marital'])
education = LabelEncoder().fit_transform(X['education'])
poutcome = LabelEncoder().fit_transform(X['p_outcome'])
```

Initially partitioning the data into Target and response variables which will help us to get into the format our model needs the data to be.
Listing out all the categorical variables and encoding it to Zeroes and Ones so that computer can understand its values, with the help of sklearn library's built in module called `LabelEncoder` we use `fit_transform` method to label the categorical values.

```
1
2  from sklearn.compose import ColumnTransformer
3  columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(),
4                                          [1,2,3,11])],
5                                        remainder='passthrough')
6  X=np.array(columnTransformer.fit_transform(X))
```

From `sklearn.compose` we import `ColumnTransformer`.

ColumnTransformer applies transformers to columns of an array or data-frames in pandas. This estimator allows different columns or column subsets of the input to be transformed separately and the features generated by each transformer will be concatenated to form a single feature space.

Using `ColumnTransformer` with `OneHotEncoder` we got all our categorical data converted into numeric data.

Splitting the Dataset

```
1  ## spliting the data into train and test
2  from sklearn.model_selection import train_test_split
3  X_train, X_test, y_train, y_test = train_test_split(X,Y,
4                                                      test_size = 0.2,
5                                                      random_state = 123)
6  ## Feature scaling
7  from sklearn.preprocessing import StandardScaler
8  sc = StandardScaler()
9  X_train = sc.fit_transform(X_train)
10 X_test = sc.fit_transform(X_test)
```

Splitting the dataset into training and test set using `train_test_split` method provided by sklearn model, and splitting 80% as training data and 20% as testing data.

Also, scaling the features of dataset using Standard Scaler, which standardizes all the features by removing the mean and scaling to unit variance.

```python
# train the model and use it to predict the label for unseen data
def fit_ml_algo(algo, X_train, y_train, X_test, y_test):

    model = algo.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = round(accuracy_score(y_pred, y_test) * 100, 2)
    cf_matrix = confusion_matrix(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = precision_score(y_pred, y_test)
    f1 = f1_score(y_test, y_pred)

    return acc, cf_matrix, precision, recall, f1, model
```

Trying to simplify the code by making a python function which will take model name, training part and testing part of data as input parameters and returns Model, Accuracy of the model, Confusion matrix, Precision, Recall and F1-Score. It's an approach for generalizing the model training and testing part and getting results from it.

Results

Logistic Regression:

Logistic Regression is used when the dependent variable(target) is categorical.
For example,

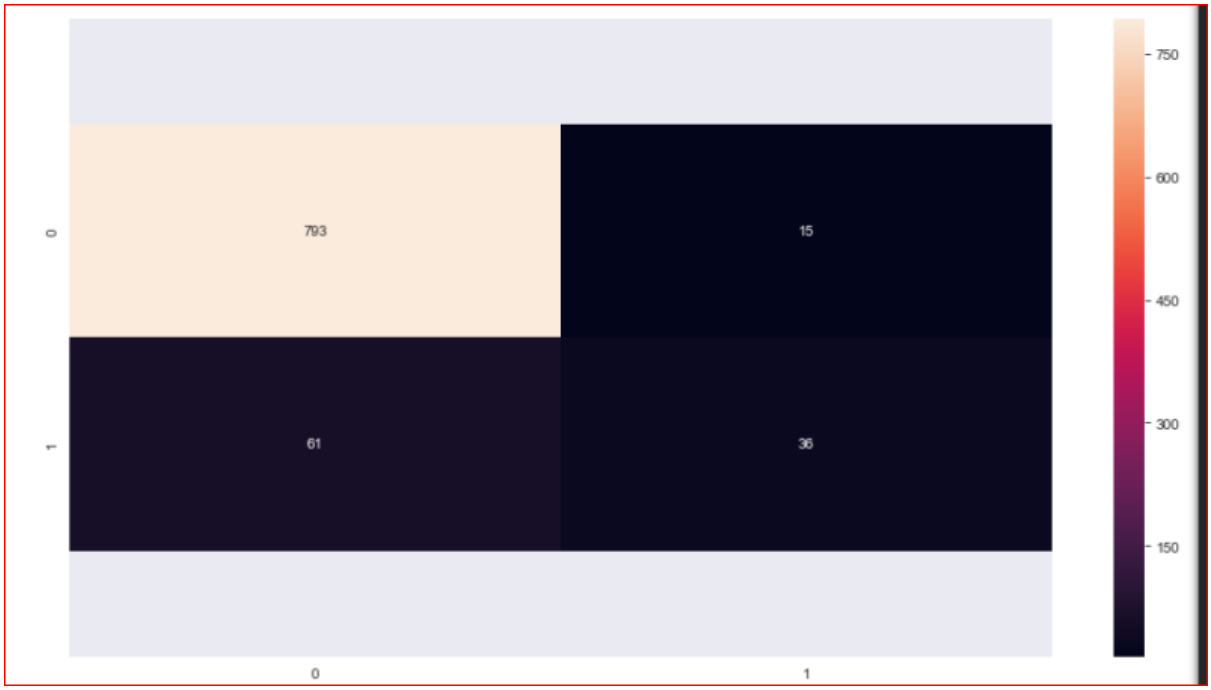
- To predict whether an email is spam (1) or (0)
- Whether the tumour is malignant (1) or not (0)

```python
# LOGISTIC REGRESSION

acc, cf_matrix, precision, recall, f1, model = fit_ml_algo(
    LogisticRegression(), X_train, y_train, X_test, y_test)
y_pred_train = model.predict(X_train)
acc_train = round(accuracy_score(y_pred_train, y_train) * 100, 2)
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
print("Testing Accuracy: ", acc,'%')
print("Training Accuracy :",acc_train,'%')
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)

```

```
Testing Accuracy:  91.6 %
Training Accuracy : 89.85 %
Precision:  0.7058823529411765
Recall:  0.3711340206185567
F1 Score:  0.48648648648648646
```



After applying Logistic Regression model on our dataset, we can definitely see that the model has given pretty nice results 91.6% testing accuracy with almost 71% times precision seems to be worthy of all our efforts put into it. And, the confusion matrix plot gives us more confidence about how good our model has been fitted over the data.

Random Forest Classifier

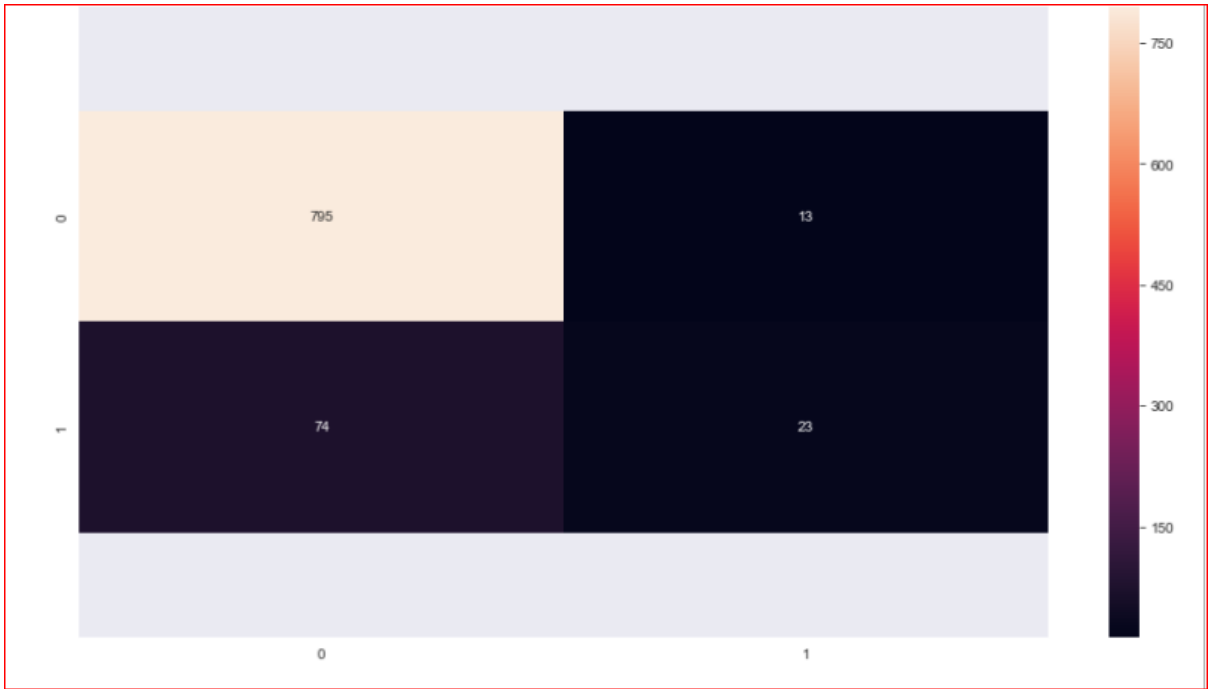
```
:
# RANDOM FOREST CLASSIFIER

acc, cf_matrix, precision, recall, f1, model = fit_ml_algo(
    RandomForestClassifier(), X_train, y_train, X_test, y_test)
y_pred_train = model.predict(X_train)
acc_train = round(accuracy_score(y_pred_train, y_train) * 100, 2)
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
print("Testing Accuracy: ", acc,'%')
print("Training Accuracy :",acc_train,'%')
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)
```

```
Testing Accuracy:  90.39 %
Training Accuracy : 100.0 %
Precision:  0.6388888888888888
Recall:  0.23711340206185566
F1 Score:  0.3458646616541353
```

A random forest classifier is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and

control over-fitting. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.



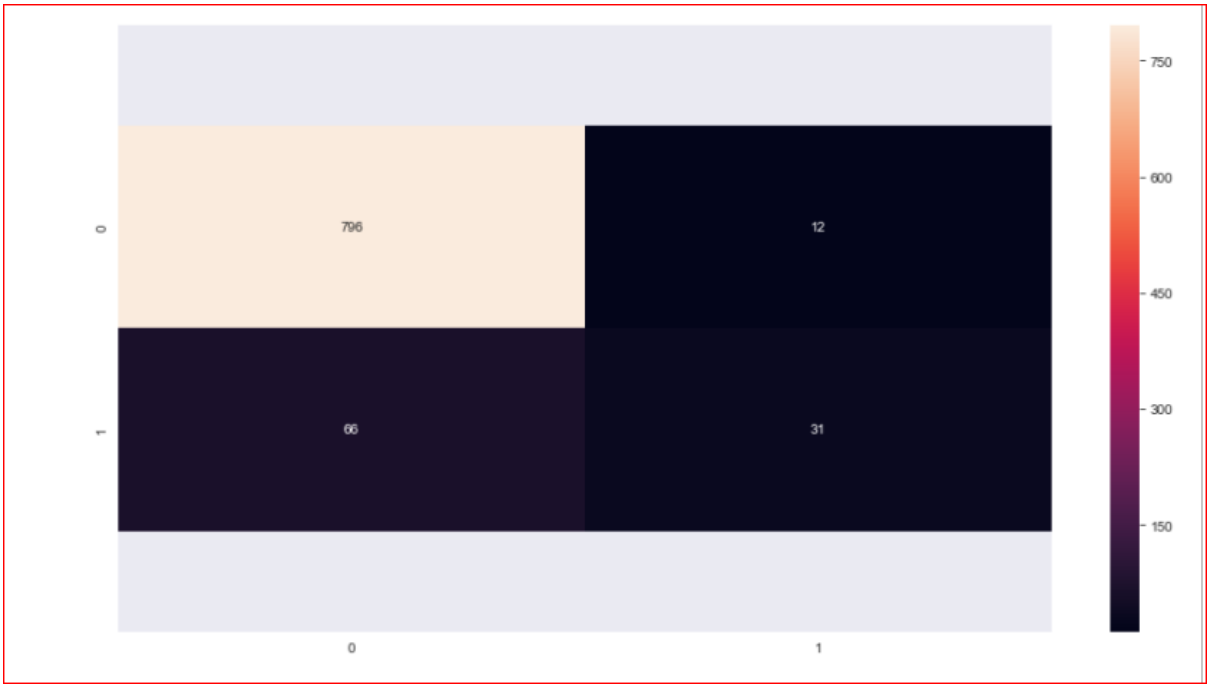
Random forest also gives us the accuracy of around 90% which is less than what was expected, also the key point to notice here is that the model is the classifying more of Negative samples as positive samples which in many cases may prove to be dangerous. So, even if the model is giving 90% accuracy it is still unreliable. It may or may not prove to be a case of overfitting.

Linear Support Vector Machine

```
# SUPPORT VECTOR MACHINE

acc, cf_matrix, precision, recall, f1, model = fit_ml_algo(LinearSVC(),
                                        X_train, y_train, X_test, y_test)
y_pred_train = model.predict(X_train)
acc_train = round(accuracy_score(y_pred_train, y_train) * 100, 2)
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
print("Testing Accuracy: ", acc,'%')
print("Training Accuracy :",acc_train,'%')
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)
```

```
Testing Accuracy:  91.38 %
Training Accuracy : 89.57 %
Precision:  0.7209302325581395
Recall:  0.31958762886597936
F1 Score:  0.4428571428571428
```



A support vector Classifier (SVC) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVC model sets of labelled training data for each category, they're able to categorize new data.

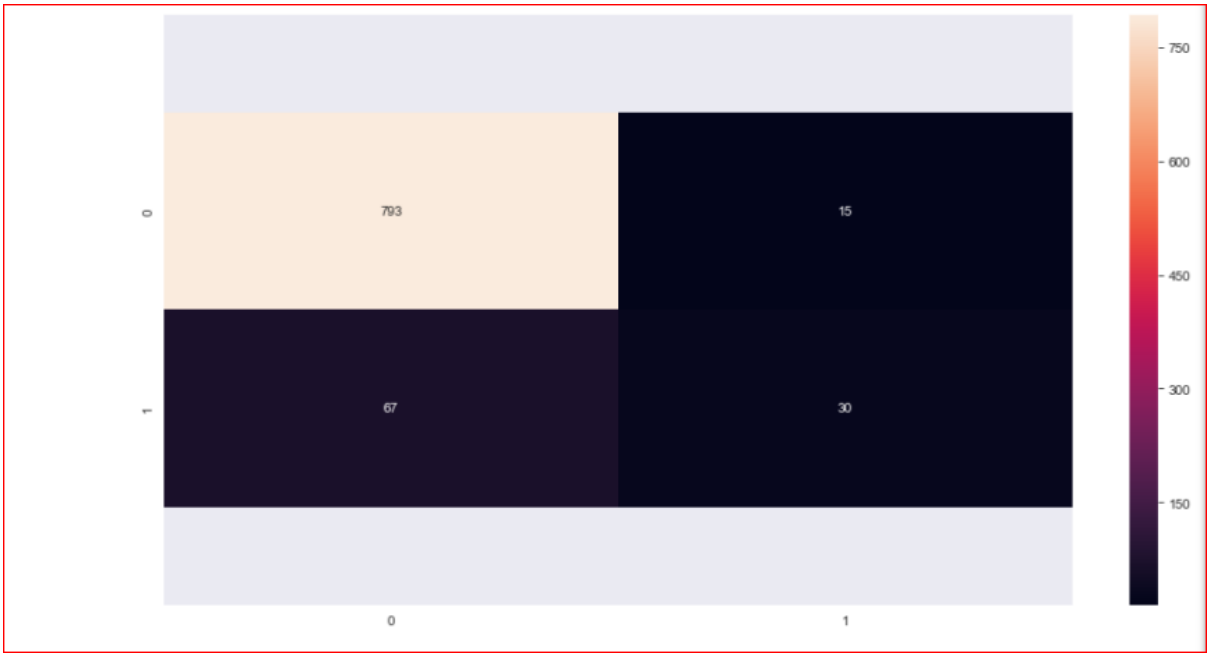
Linear SVC seems to be doing a pretty good job with our dataset, unlike Random forest model. As, it has decent testing accuracy with less misclassification rate than random forest. If, required we can definitely use this model for further analysis.

Gradient Boosting Classifier

```
# GRADIENT BOOSTING CLASSIFIER

acc, cf_matrix, precision, recall, f1, model = fit_ml_algo(
    GradientBoostingClassifier(), X_train, y_train, X_test, y_test)
y_pred_train = model.predict(X_train)
acc_train = round(accuracy_score(y_pred_train, y_train) * 100, 2)
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
print("Testing Accuracy: ", acc,'%')
print("Training Accuracy :",acc_train,'%')
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)
```

```
Testing Accuracy:  90.94 %
Training Accuracy : 92.89 %
Precision:  0.6666666666666666
Recall:  0.30927835051546393
F1 Score:  0.4225352112676056
```



Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

Gradient boosting algorithm most of the times performs a pretty good job but here I can say it performs decent. As, we can see the accuracy is approx. 91% but still confusion matrix shows us that misclassification rate is slightly higher than others (better than Random Forest).

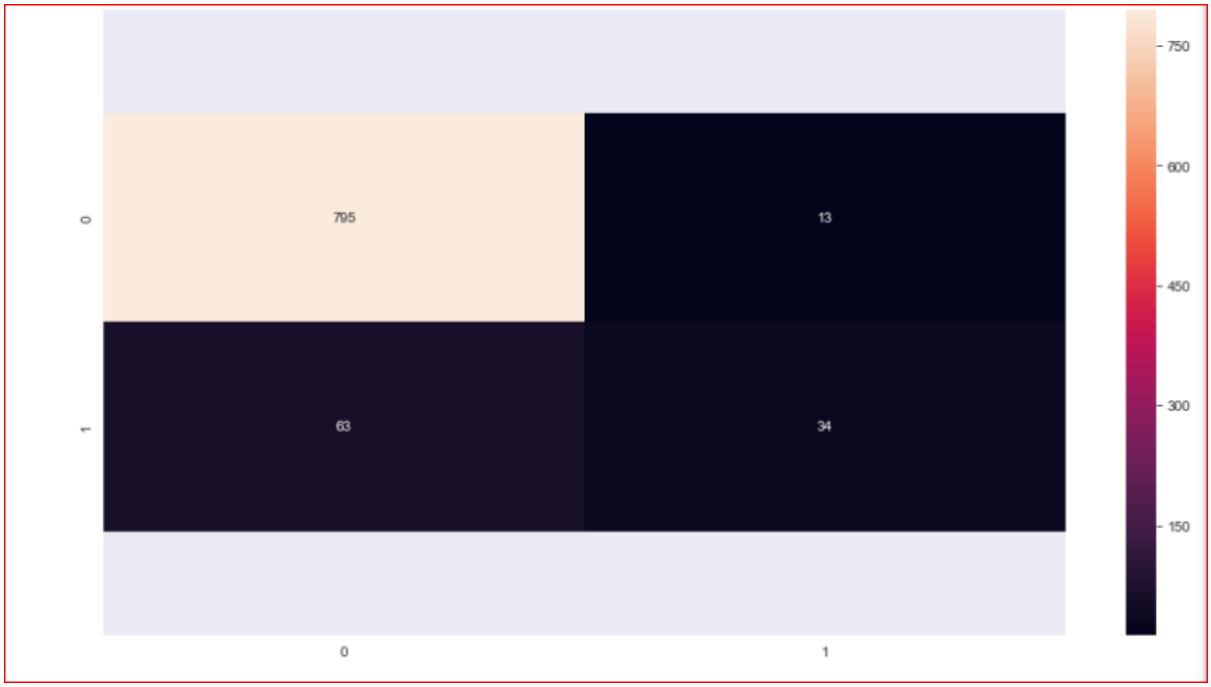
Adaboost (*Adaptive Boosting*) Classifier

```
# ADABOOST CLASSIFIER

acc, cf_matrix, precision, recall, f1, model = fit_ml_algo(
    AdaBoostClassifier(), X_train, y_train, X_test, y_test)
y_pred_train = model.predict(X_train)
acc_train = round(accuracy_score(y_pred_train, y_train) * 100, 2)
fig,ax=plt.subplots(figsize=(15,8))
sns.set_style('dark')
ax = sns.heatmap(cf_matrix, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
print("Testing Accuracy: ", acc,'%')
print("Training Accuracy :",acc_train,'%')
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)


```

```
Testing Accuracy:  91.6 %
Training Accuracy : 89.96 %
Precision:  0.723404255319149
Recall:  0.35051546391752575
F1 Score:  0.47222222222222215
```



Ada-boost or Adaptive Boosting combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations.

As, the above snippet shows Adaboost seems to perform quite nicely over the data. The model gives approx. 92% accuracy on testing set which seems to be the maximum till now also the other parameters like `precision, recall` and F1-score also seems to be fairly good and as the confusion matrix also gives us enough details which proves that Adaboost is slightly outperforming other models that we've fitted on our data.

Summary and Conclusion:

Model Name	Training Accuracy	Testing Accuracy
Logistic Regression	89.85 %	91.60%
Random Forest Classifier	100.00%	90.39%
Support Vector Classifier	89.57%	91.38%
Gradient Boosting	92.89 %	90.94 %
Adaboost Classifier	89.96 %	91.66 %

As evident from the table above, all the algorithms except Random forest give acceptable results in both training and testing phases. Adabooost classifier gives the best possible accuracy at 91.66% with good precision and low misclassification rates while Logistic Regression is the second-best algorithm with 91.60% closely followed by Support vector classifier at 91.38%.

Random Forest Classifier is not a consideration as the accuracy scores in training part is extremely high i.e. maybe overfitting.