INTRODUCTION

- NAME - DEBANGSHU DASGUPTA
- SCHOOL OF COMPUTER SCIENCE ENGINEERING
- KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

1. This project predicts if a person is suffering from parkinson's disease based on the audio/voice measures. In this project I performed various data visualizations to find the relation between different segments of the dataset.
2. Furthermore performed train test split and used various ML models like Random Forest Classifier, Logistic Regression, SVM etc to predict the accuracy of the model.

PREDICTING IF A PERSON IS SUFFERING FROM PARKINSON'S DISEASE BASED ON AUDIO/VOICE MEASURES.

```
# Importing necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
# Reading the dataset
from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/My Drive/PROJECTS/PARKINSONS DISEASE PREDICTION/Dataset/parkinsons.data"
df=pd.read_csv(path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
```

```
# loading the dataset
df
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:F
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.003
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.004
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.005
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.005
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.006
...	...	...	...	...	...	...	
190	phon_R01_S50_2	174.188	230.978	94.261	0.00459	0.00003	0.002
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	0.00003	0.003
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	0.00008	0.006
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	0.00004	0.003
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	0.00003	0.002

195 rows × 24 columns

```
df.shape
# Here we get the number of rows and columns in (row, col) format
```

```
(195, 24)
```

```
df.isnull().sum()
# Gives the total number of null values in each column
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

```
df.info()
```

```
# Using df.info we can find the presence of missing values at the same time view the types of columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64  
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
df.describe()
# This method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or Da
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PP
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.00000
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.00344
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.00275
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.00092
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.00186
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.00269
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.00395
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.01958

```
df.columns
# Displaying the column names of the dataframe
```

```
Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter(%)',
       'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
       'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

```
# These are the column names as seen in the dataset.
```

Attribute Information:

Matrix column entries (attributes):

name - ASCII subject name and recording number

MDVP:Fo(Hz) - Average vocal fundamental frequency MDVP:Fhi(Hz) - Maximum vocal fundamental frequency MDVP:Flo(Hz) - Minimum vocal fundamental frequency MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several measures of variation in fundamental frequency MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude NHR,HNR - Two measures of ratio of noise to tonal components in the voice status - Health status of the subject (one) - Parkinson's, (zero) - healthy RPDE,D2 - Two nonlinear dynamical complexity measures DFA - Signal fractal scaling exponent spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

```
df['status']
# The Status column is the target column.
```

```
0      1
1      1
2      1
3      1
4      1
      ..
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

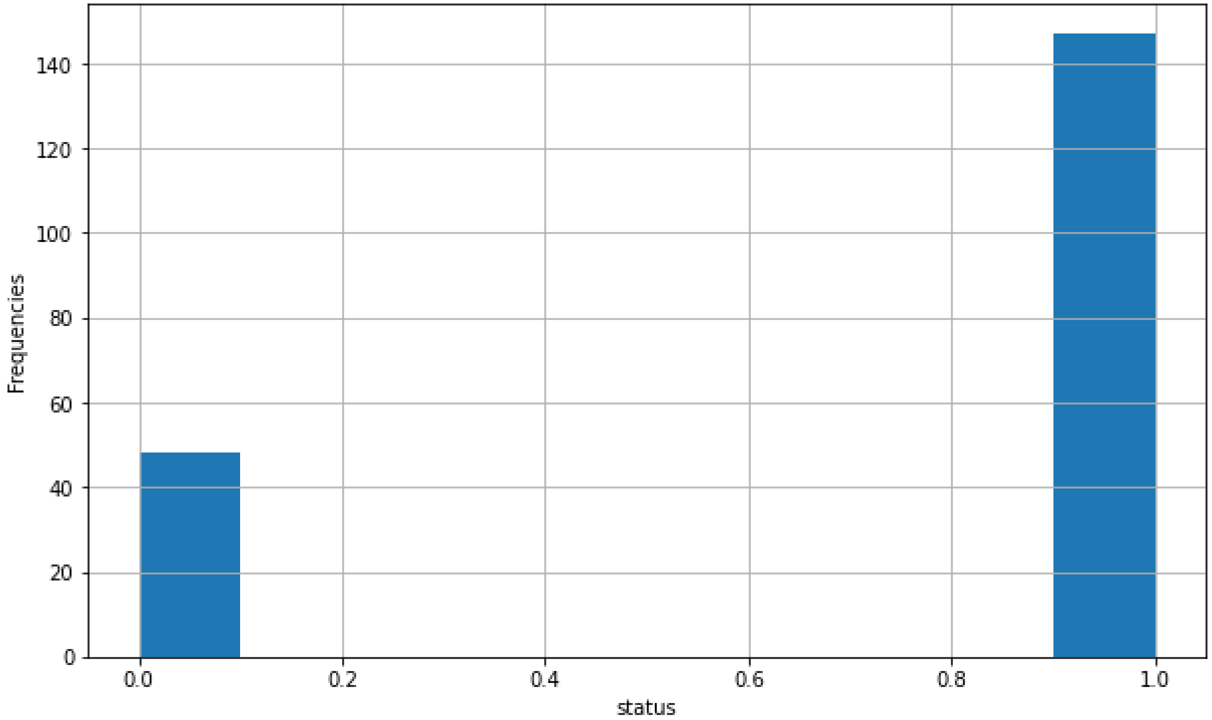
```
# status - Health status of the subject (one) - Parkinson's, (zero) - healthy
```

VISUALIZING THE DATA

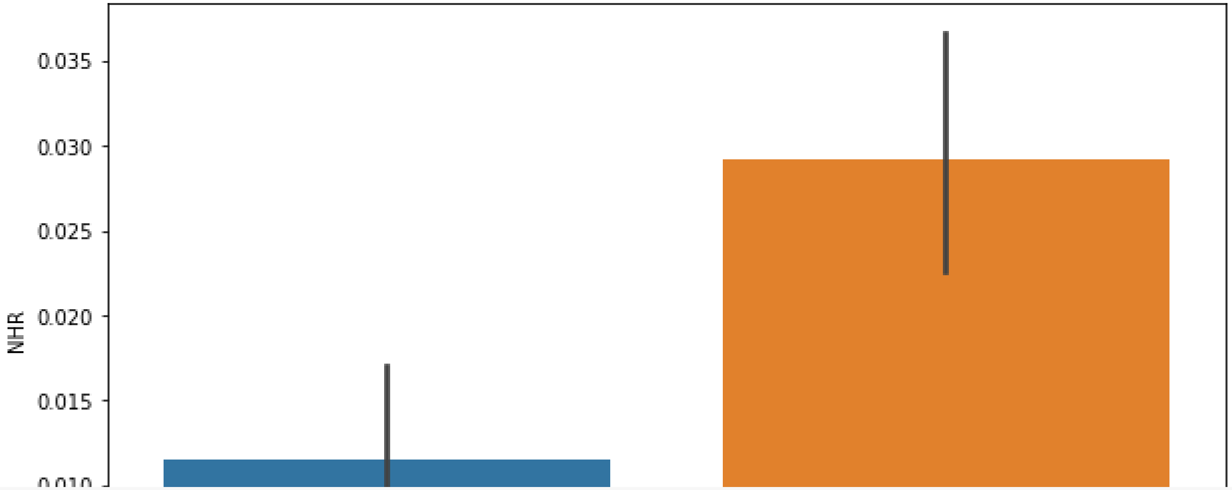
```
plt.figure(figsize=(10, 6))
df.status.hist()
plt.xlabel('status')
plt.ylabel('Frequencies')
```

```
plt.plot()
# The dataset has high number of patients effected with Parkinson's disease.
```

[]

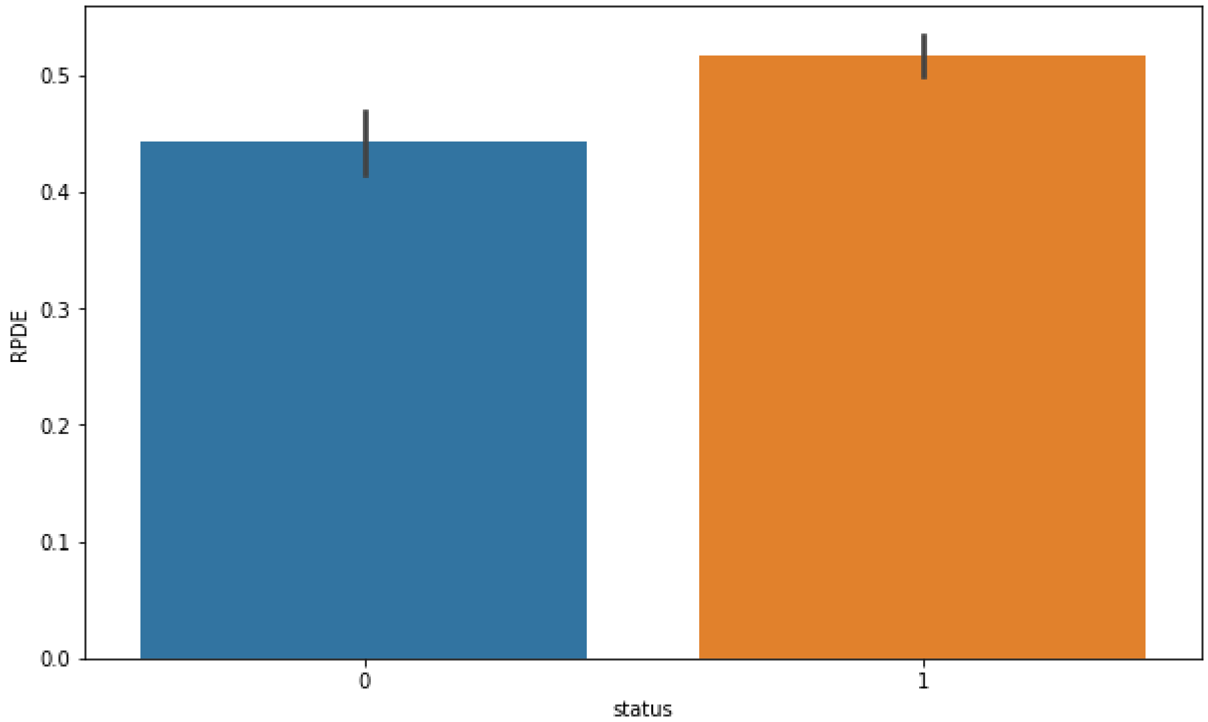



```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="NHR",data=df);
# The patients effected with Parkinson's disease have high NHR that is the measures of ratio of noise to tonal components in the voic
```




```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="HNR",data=df);
# The patients effected with Parkinson's disease have high HNR that is the measures of ratio of noise to tonal components in the voic
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x="status",y="RPDE",data=df);
# The nonlinear dynamical complexity measure RPDE is high in the patients effected with Parkinson's disease.
```



Distribution plot

```
rows=3
cols=7
fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(16,4))
col=df.columns
index=1
for i in range(rows):
    for j in range(cols):
```

```
        sns.distplot(df[col[index]],ax=ax[i][j])
        index=index+1

plt.tight_layout()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
```

A distribution plot displays a distribution and range of a set of numeric values plotted against a dimension

```
df.drop(['name'],axis=1,inplace=True)
# Removing  name column for machine learning algorithms.
```

```
X=df.drop(labels=['status'],axis=1)
Y=df['status']
X.head()
### Spitting the dataset into x and y
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitte
0	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0
1	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0
2	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0
3	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0
4	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0


```
X.head()
# Displaying X head
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitte
0	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0
1	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0
2	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0
3	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0
4	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0

```
Y.head()
# Displaying Y head
```

```
0    1
1    1
2    1
3    1
```

```
4    1
Name: status, dtype: int64
```

Splitting the data

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=40)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
# Splitting the data into x_train, y_train, x_test, y_test
```

```
(156, 22) (39, 22) (156,) (39,)
```

MACHINE LEARNING

Logistic Regression

```
log_reg = LogisticRegression().fit(X_train, Y_train)

#predict on train
train_preds = log_reg.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds))

#predict on test
test_preds = log_reg.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds))
print('-'*50)

#Confusion matrix
```

```
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds))
```

```
Model accuracy on train is:  0.8717948717948718
Model accuracy on test is:  0.8461538461538461
--------------------------------------------------
confusion_matrix train is:  [[ 24  16]
 [  4 112]]
confusion_matrix test is:  [[ 5  3]
 [ 3 28]]
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

RANDOM FOREST

```
RF=RandomForestClassifier().fit(X_train,Y_train)
#predict on train
train_preds2 = RF.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds2))

#predict on test
test_preds2 = RF.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds2))

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds2))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds2))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  0.8717948717948718
confusion_matrix train is:  [[ 40   0]
 [  0 116]]
confusion_matrix test is:  [[ 5  3]
 [ 2 29]]
```

```
# Wrong Predictions made.
print((Y_test !=test_preds2).sum(),'/',((Y_test == test_preds2).sum()+(Y_test != test_preds2).sum()))
```

```
5 / 39
```

```
# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds2))
```

```
KappaScore is:  0.587737843551797
```

```
## Let us go ahead and compare the predicted and actual values
```

```
test_preds2
```

```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1])
```

```
test_preds2,Y_test
```

```
(array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1]), 96     1
 5      1
 116    1
 35     0
 178    1
 185    0
 54     1
```

```
134    1
90     1
187    0
139    1
142    1
175    0
26     1
89     1
140    1
155    1
23     1
132    1
37     1
151    1
28     1
85     1
93     1
172    0
75     1
18     1
105    1
121    1
130    1
33     0
46     0
166    0
163    1
11     1
164    1
81     1
111    1
67     1
Name: status, dtype: int64)
```

```
## Saving the actual and predicted values to a dataframe
```

```
ddf=pd.DataFrame(data=[test_preds2,Y_test])
```

```
ddf.T
```

	0	1
0	1	1
1	1	1
2	1	1
3	0	0
4	1	1
5	1	0
6	1	1
7	1	1
8	1	1
9	1	0
10	1	1
11	1	1
12	1	0
13	1	1
14	1	1
15	1	1
16	1	1
17	1	1

```
# Above 0 means Predicted Value and 1 is True Value.
```

```
# Random forest model performs better compared to other models.
```

```
# Random forest model gives us an accuracy of 94 percent compared to logistic regression which gave us 84 percent accuracy
```

Applying other machine learning models to see if there is improvement in accuracy.

[ ] ↳ *1 cell hidden*

K-NEAREST NEIGHBOURS

28 1 1

```
#fit the model on train data
KNN = KNeighborsClassifier().fit(X_train,Y_train)
#predict on train
train_preds5 = KNN.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds5))

#predict on test
test_preds5 = KNN.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds5))
```

```
Model accuracy on train is:  0.9102564102564102
Model accuracy on test is:  0.8461538461538461
```

```
#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds5))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds5))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds5).sum(),'/',((Y_test == test_preds5).sum()+(Y_test != test_preds5).sum()))

print('-'*50)
```

```
confusion_matrix train is:  [[ 30  10]
 [  4 112]]
confusion_matrix test is:  [[ 4  4]
 [ 2 29]]
Wrong predictions out of total
--------------------------------------------------
6 / 39
--------------------------------------------------
```

```
# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds5))
```

```
KappaScore is:  0.48
```

SUPPORT VECTOR MACHINE

```
#fit the model on train data
SVM = SVC(kernel='linear')
SVM.fit(X_train, Y_train)

#predict on train
train_preds6 = SVM.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds6))

#predict on test
test_preds6 = SVM.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds6))
```

```
Model accuracy on train is:  0.8782051282051282
Model accuracy on test is:  0.8974358974358975
```

```
#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds6))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds6))
print('Wrong predictions out of total')
print('-'*50)

print("recall", metrics.recall_score(Y_test, test_preds6))
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds6).sum(),'/',((Y_test == test_preds6).sum()+(Y_test != test_preds6).sum()))
print('-'*50)
```

```
confusion_matrix train is:  [[ 23  17]
 [  2 114]]
confusion_matrix test is:  [[ 5  3]
 [ 1 30]]
Wrong predictions out of total
--------------------------------------------------
recall 0.967741935483871
--------------------------------------------------
4 / 39
--------------------------------------------------
```

```
# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds6))
```

```
KappaScore is:  0.6533333333333333
```