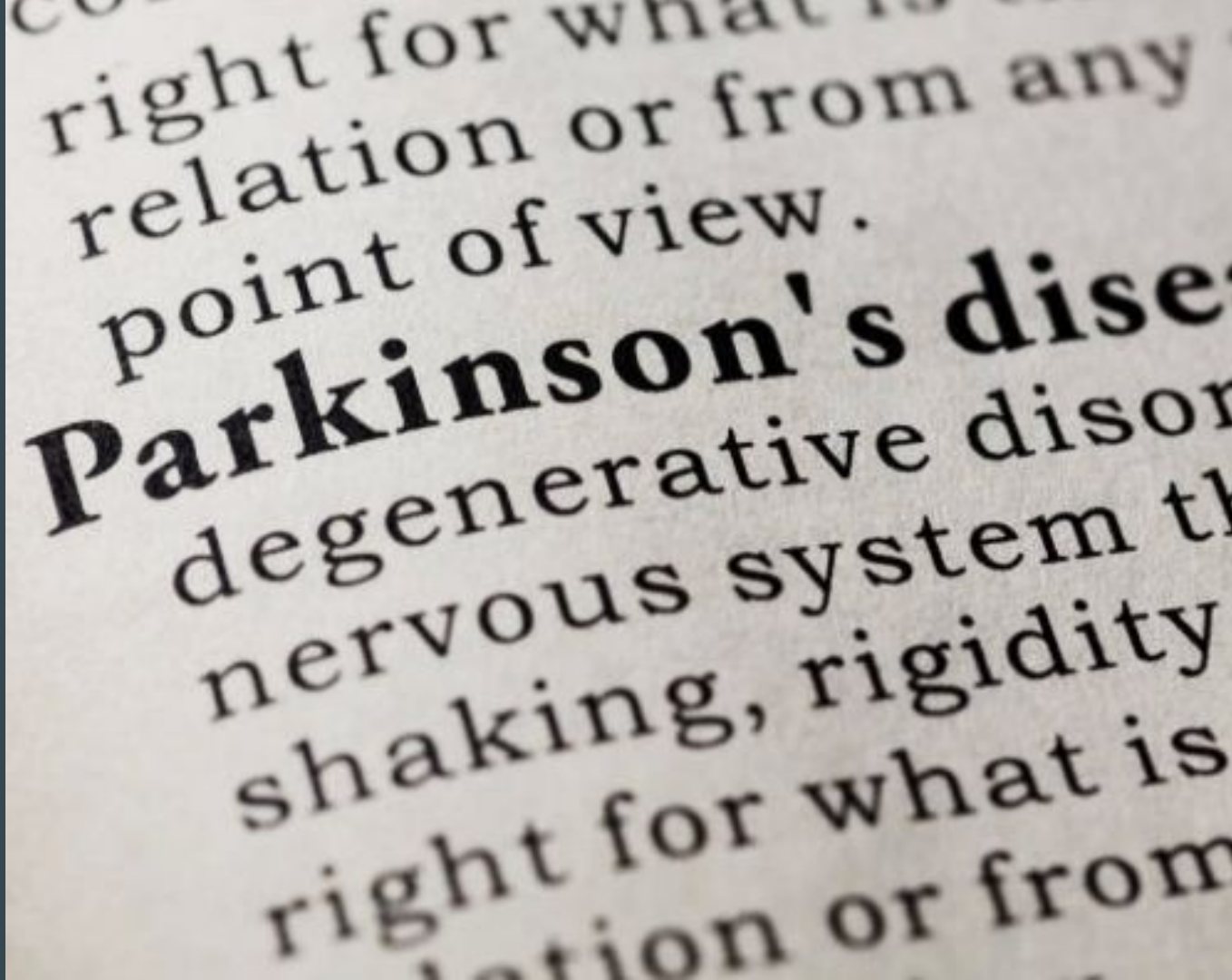
PARKINSON'S DISEASE DETECTION

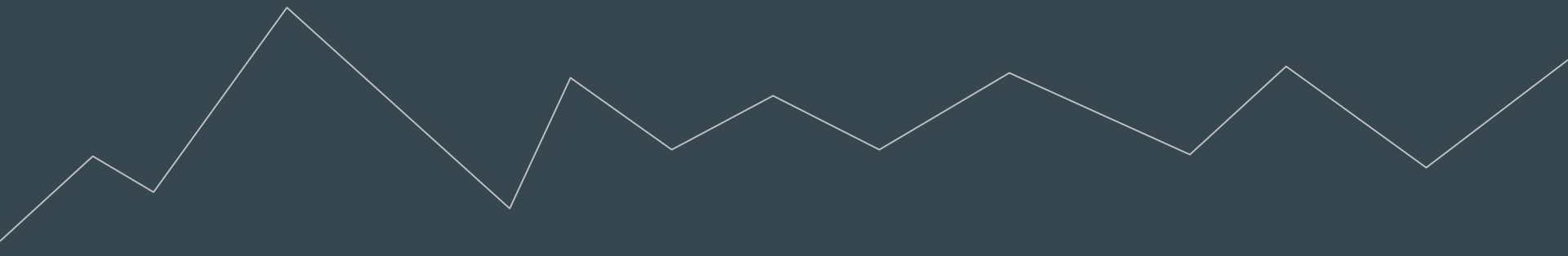
What Is Parkinson's?

Parkinson's disease (PD) is a neurodegenerative disorder

Which affects predominately dopamine-producing (“dopaminergic”) neurons in a specific area of the brain called substantia nigra..



GOAL:



Create a model with high accuracy to determine if a patient is likely to have Parkinson’s or not.

DATASET CONTAINS:

Source:

The data information for this project is available at
https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/

SIZE:

Parkinsson disease.csv(37.38 KB)

ATTRIBUTES:

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

DATASET:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shir
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	
5	phon_R01_S01_6	120.552	131.162	113.787	0.00968	0.00008	0.00463	0.00750	0.01388	0.04701	...	
6	phon_R01_S02_1	120.267	137.244	114.820	0.00333	0.00003	0.00155	0.00202	0.00466	0.01608	...	
7	phon_R01_S02_2	107.332	113.840	104.315	0.00290	0.00003	0.00144	0.00182	0.00431	0.01567	...	
8	phon_R01_S02_3	95.730	132.068	91.754	0.00551	0.00006	0.00293	0.00332	0.00880	0.02093	...	
9	phon_R01_S02_4	95.056	120.103	91.226	0.00532	0.00006	0.00268	0.00332	0.00803	0.02838	...	

10 rows × 24 columns

DATA PREPROCESSING

drop column:

```
In [8]: data = data.drop(['name'], axis=1)
        data_corr = data.corr()
```

Dropped name column as it does not contribute to model building

EDA: Exploratory data analysis

DATASET_CORR:

```
data_corr.head()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer
MDVP:Fo(Hz)	1.000000	0.400985	0.596546	-0.118003	-0.382027	-0.076194	-0.112165	-0.076213	-0.098374
MDVP:Fhi(Hz)	0.400985	1.000000	0.084951	0.102086	-0.029198	0.097177	0.091126	0.097150	0.002281
MDVP:Flo(Hz)	0.596546	0.084951	1.000000	-0.139919	-0.277815	-0.100519	-0.095828	-0.100488	-0.144543
MDVP:Jitter(%)	-0.118003	0.102086	-0.139919	1.000000	0.935714	0.990276	0.974256	0.990276	0.769063
MDVP:Jitter(Abs)	-0.382027	-0.029198	-0.277815	0.935714	1.000000	0.922911	0.897778	0.922913	0.703322

5 rows × 23 columns

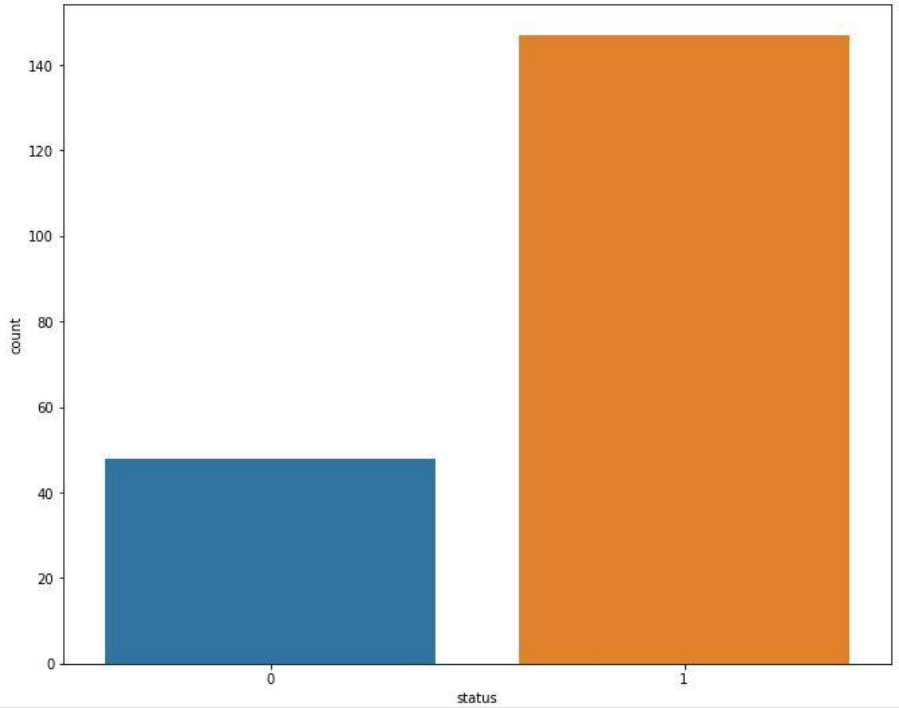
DATASET_head:

```
data.head(10)
```

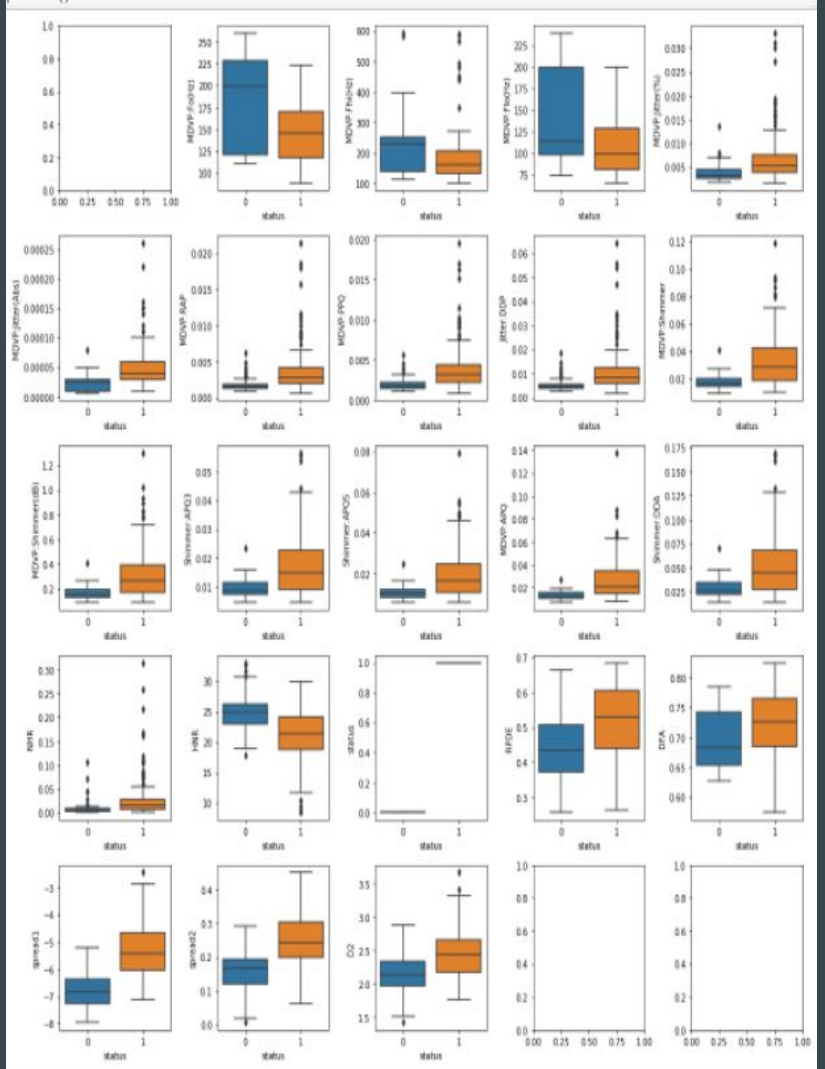
	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...
5	phon_R01_S01_6	120.552	131.162	113.787	0.00968	0.00008	0.00463	0.00750	0.01388	0.04701	...
6	phon_R01_S02_1	120.267	137.244	114.820	0.00333	0.00003	0.00155	0.00202	0.00466	0.01608	...
7	phon_R01_S02_2	107.332	113.840	104.315	0.00290	0.00003	0.00144	0.00182	0.00431	0.01567	...
8	phon_R01_S02_3	95.730	132.068	91.754	0.00551	0.00006	0.00293	0.00332	0.00880	0.02093	...
9	phon_R01_S02_4	95.056	120.103	91.226	0.00532	0.00006	0.00268	0.00332	0.00803	0.02838	...

In [10]:
```
fig, ax = plt.subplots(figsize=(11, 9))
sns.countplot(data['status'])
#plt.xlim(10)
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22cba4e38c8>
```



COUNTPLOT_TARGET:



From the boxplot shown above it is very evident that if a patient has a lower rate of

'HNR','

MDVP:Flo(Hz)',

'MDVP:Fhi(Hz)',

'MDVP:Fo(Hz)' ,

then he/she is affected by parkinson's disease.

MODEL USED:

Random Forest Classifier :

1. What is Random Forest algorithm?
2. Why Random Forest algorithm?

What is Random Forest ?

Random forest is a supervised learning algorithm which is used for both classification as well as regression.

But however, it is mainly used for classification problems.

As we know that a forest is made up of trees and more trees means more robust forest.
random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting.

It is an method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

1. Why Random Forest ?

Overfitting is one critical problem that may make the results worse, but for Random Forest algorithm,

if there are enough trees in the forest, the classifier won't overfit the model.
The third advantage is the classifier of Random Forest can handle missing values,

and the last advantage is that the Random Forest classifier can be modeled for categorical values.

K-Nearest Neighbors (KNN)

1. What is KNN?
2. Why KNN?

1. What is knn ?

KNN is a lazy learning, non-parametric algorithm. It uses data with several classes to predict the classification of the new sample point. KNN is non-parametric since it doesn't make any assumptions on the data being studied, i.e., the model is distributed from the data.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

> "Birds of a feather flock together."

1. Why KNN algorithm?

KNN can be used in both regression and classification predictive problems. However, when it comes to industrial problems, it's mostly used in classification since it fairs across all parameters evaluated when determining the usability of a technique

1. **Prediction Power**
2. **Calculation Time**
3. **Ease to Interpret the Output**

KNN algorithm fairs across all parameters of considerations. But mostly, it is used due to its ease of interpretation and low calculation time.

PERFORMANCE ANALYSIS

RANDOM FOREST ACCURACY:

In [23]:
```
z=0
b=0
for i in np.arange(10,150):
    rfc = RandomForestClassifier(n_estimators = i, max_depth=15)
    rfc.fit(X_train, y_train)
    preds_rfc=rfc.predict(X_test)
    acc=accuracy_score(y_test,preds_rfc)
    if acc>z:
        z=acc
        b=i
print("For",b,"number of trees,accuracy is",z)
```

```
For 13 number of trees,accuracy is 0.9491525423728814
```

ACCURACY = 94%

KNN ACCURACY:

```
test_score ,train_score=[],[]
k_value =[]
for i in range(1,15):
  knn = KNS(i)
  knn.fit(X_train,y_train)
  train_score.append(knn.score(X_train,y_train))
  test_score.append(knn.score(X_test,y_test))
  k_value.append(i)
print('Accuracy=',round(max(test_score)*100,3),'% for K=',k_value[(test_score.index(max(test_score)))])
```

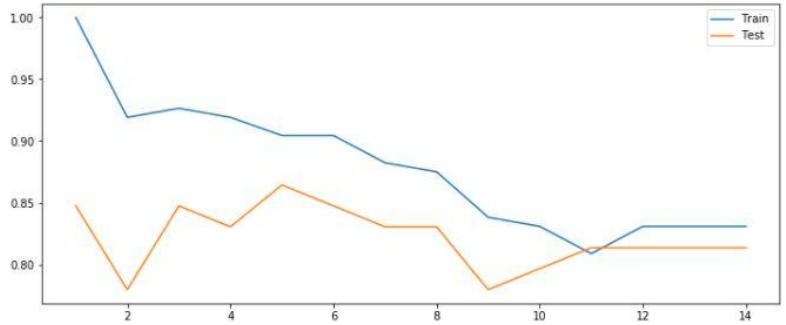
```
Accuracy= 86.441 % for K= 5
```

ACCURACY = 86%

MODEL BUILDING PHASE

KNN MODEL:

In [13]:
```
plt.figure(figsize=(12,5))
p=sns.lineplot(range(1,15),train_score , markers='*',label ='Train')
p =sns.lineplot(range(1,15),test_score, markers='o',label = 'Test')
```



In [14]:
```
knn = KNS(5)
```

In [15]:
```
knn.fit(X_train,y_train)
```

Out[15]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [16]:
```
prediction=knn.predict(X_test)
```

In [17]:
```
prediction
```

Out[17]:
```
array([1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1], dtype=int64)
```

RFC MODEL:

RANDOM FOREST CLASSIFIER

In [18]:
```
rfc = RandomForestClassifier(n_estimators=100,max_depth=15)
```

In [19]:
```
rfc = RandomForestClassifier(n_estimators=100, criterion='gini',random_state=None)
rfc.fit(X_train,y_train)
```

Out[19]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [20]:
```
rfc.score(X_train,y_train)
```

Out[20]:
```
1.0
```

In [21]:
```
preds_rfc = rfc.predict(X_test)
```

In [22]:
```
metrics.accuracy_score(y_test,preds_rfc)
```

Out[22]:
```
0.9322033898305084
```

COMPARISON OF MODEL

RFC

accuracy: 94%

Descion Tree

	precision	recall	f1-score	support
0	0.90	0.69	0.78	13
1	0.92	0.98	0.95	46
accuracy			0.92	59
macro avg	0.91	0.84	0.86	59
weighted avg	0.91	0.92	0.91	59

KNN

accuracy:86%

	precision	recall	f1-score	support
0	0.78	0.54	0.64	13
1	0.88	0.96	0.92	46
accuracy			0.86	59
macro avg	0.83	0.75	0.78	59
weighted avg	0.86	0.86	0.85	59

REFERENCE

https://www.ncbi.nlm.nih.gov/books/NBK48516/

https://ieeexplore.ieee.org

https://arxiv.org

CONCLUSION

RFC MODEL:

```
In [24]: predict = rfc.predict(X_test)
         predict
```

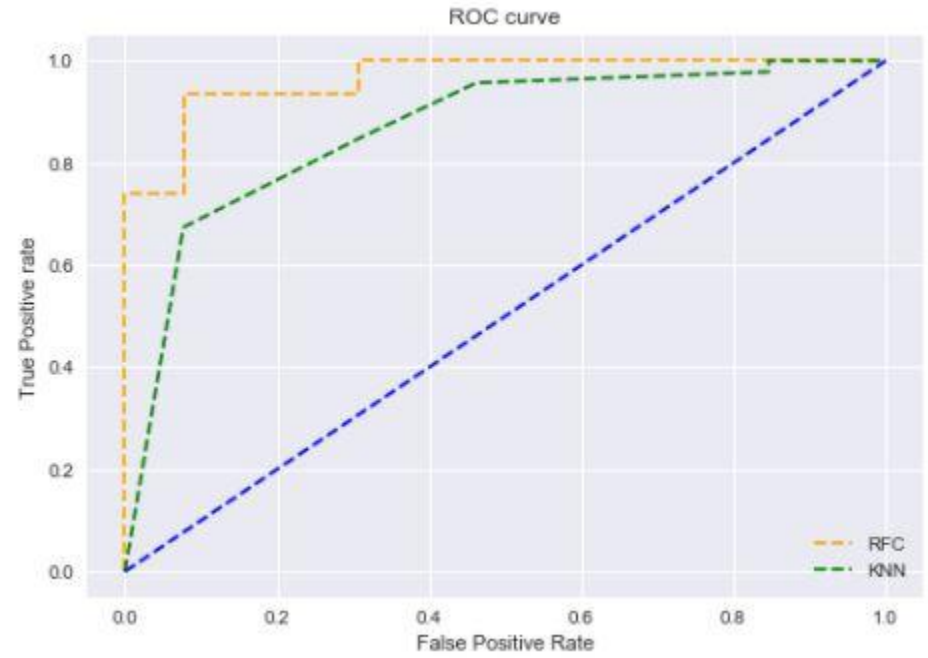
```
Out[24]: array([1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
                1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
                0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

KNN MODEL:

```
In [17]: prediction
```

```
Out[17]: array([1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1], dtype=int64)
```

roc curve:


ROC curve
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
RFC
KNN

AUC SCORE:

SCORE RFC : 0.9648829431438127

SCORE KNN : 0.8662207357859532

THANK YOU :)