Breast Cancer EDA

DataSet : https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_breast_cancer
```

```
data = pd.read_csv("data.csv") #import dataset
```

```
data.shape
```

```
    (569, 33)
```

```
data.head(5)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
0	842302	M	17.99	10.38	122.80	1001.0
1	842517	M	20.57	17.77	132.90	1326.0
2	84300903	M	19.69	21.25	130.00	1203.0
3	84348301	M	11.42	20.38	77.58	386.1
4	84358402	M	20.29	14.34	135.10	1297.0

5 rows × 33 columns

```
# Viewing the column heading
data.columns
```

```
    Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
           'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean'
           'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
           'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se
           'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se'
           'fractal_dimension_se', 'radius_worst', 'texture_worst',
           'perimeter_worst', 'area_worst', 'smoothness_worst',
           'compactness_worst', 'concavity_worst', 'concave points_worst',
           'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
          dtype='object')
```

```
data.diagnosis.value_counts()
```

```
    B    357
    M    212
    Name: diagnosis, dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
data.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	sm
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 32 columns

```
data.describe(include='object')
```

	diagnosis
count	569
unique	2
top	B
freq	357

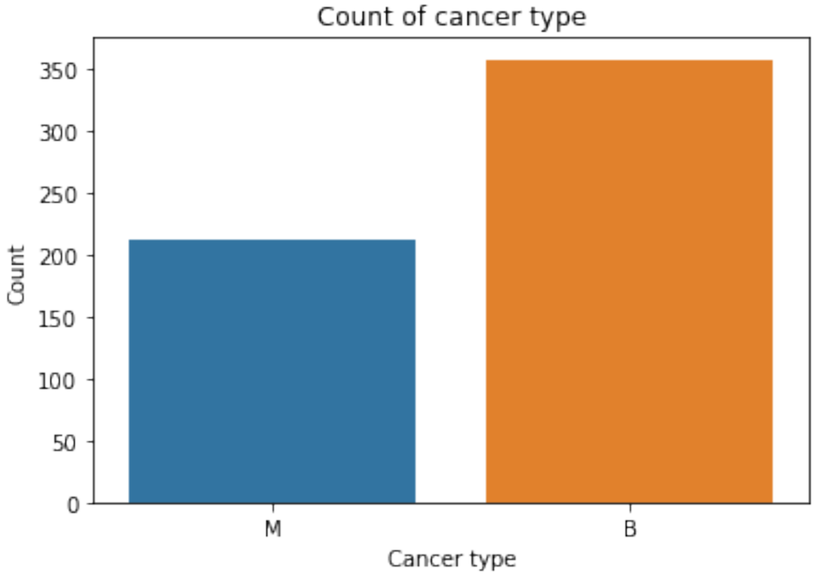
```
data.isnull().sum()
```

```
    id                           0
    diagnosis                    0
    radius_mean                  0
    texture_mean                 0
    perimeter_mean               0
    area_mean                    0
    smoothness_mean              0
    compactness_mean             0
    concavity_mean               0
    concave points_mean          0
    symmetry_mean                0
    fractal_dimension_mean       0
    radius_se                    0
    texture_se                   0
    perimeter_se                 0
    area_se                      0
    smoothness_se                0
    compactness_se               0
    concavity_se                 0
    concave points_se            0
    symmetry_se                  0
    fractal_dimension_se         0
    radius_worst                 0
    texture_worst                0
    perimeter_worst              0
    area_worst                   0
    smoothness_worst             0
    compactness_worst            0
    concavity_worst              0
    concave points_worst         0
    symmetry_worst               0
    fractal_dimension_worst      0
    Unnamed: 32                569
    dtype: int64
```

```
data.drop(['Unnamed: 32', 'id'], axis=1, inplace=True) #drop the attribute.
```

```
plt.title('Count of cancer type')
sns.countplot(data['diagnosis'])
plt.xlabel('Cancer type')
plt.ylabel('Count')
plt.show()
```

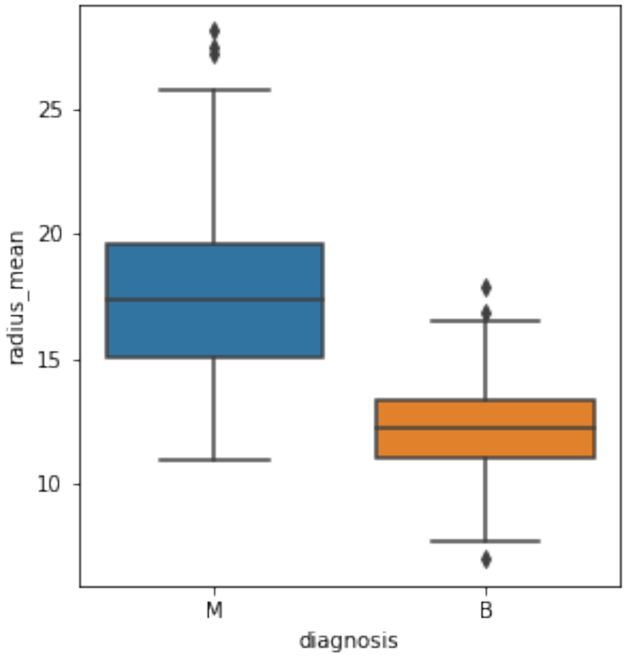
```
/Users/bhumika/opt/anaconda3/lib/python3.7/site-packages/seaborn/_decorator
  FutureWarning
```



- Observation: We have around 350 malignant cases and 210 benign cases so our dataset is imbalanced, we can use various re-sampling algorithms like under-sampling, over-sampling, SMOTE, etc.

```
# Plotting correlation between diagnosis and radius
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.boxplot(x="diagnosis", y="radius_mean", data=data)
```

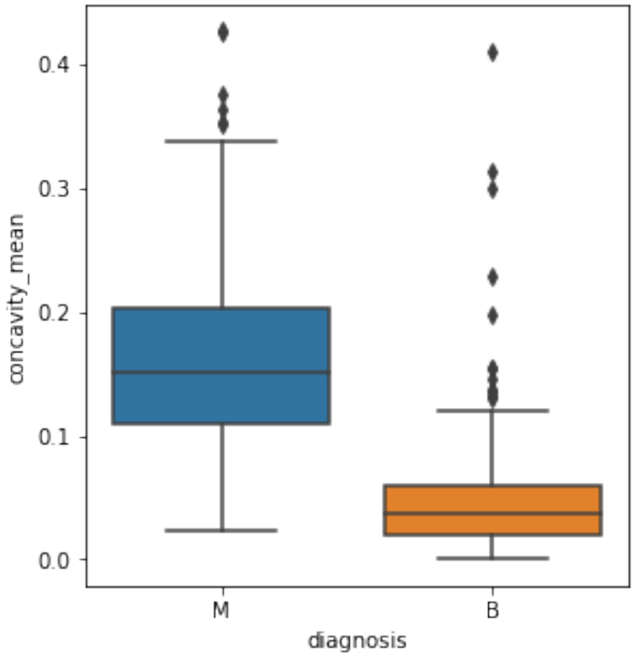
```
<AxesSubplot:xlabel='diagnosis', ylabel='radius_mean'>
```



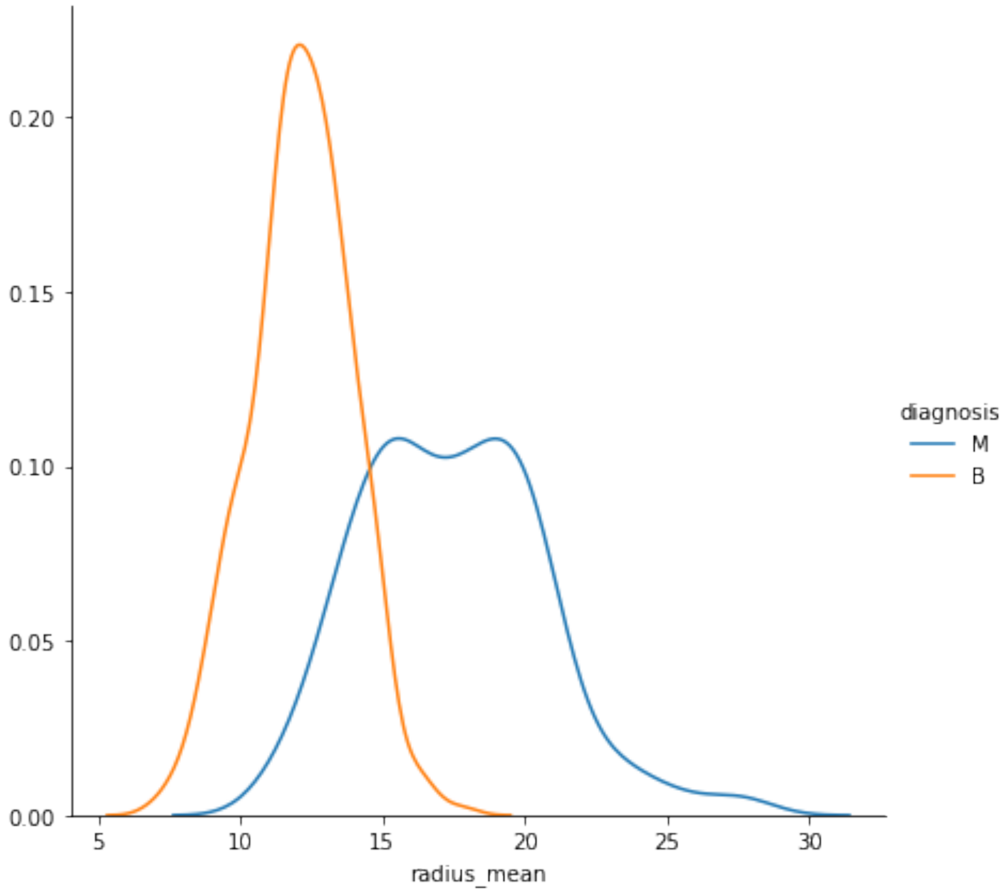
```
# Plotting correlation between diagnosis and concativity

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
sns.boxplot(x="diagnosis", y="concavity_mean", data=data)
```

```
<AxesSubplot:xlabel='diagnosis', ylabel='concavity_mean'>
```



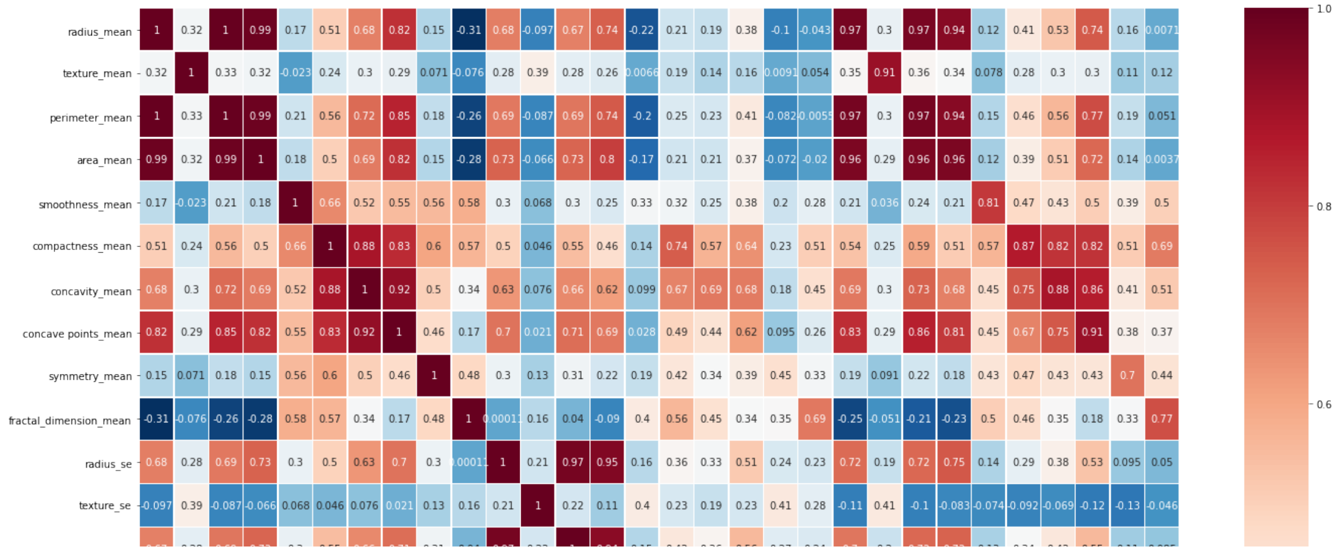
```
# Distribution density plot KDE (kernel density estimate)
sns.FacetGrid(data, hue="diagnosis", height=6).map(sns.kdeplot, "radius_mean").a
plt.show()
```

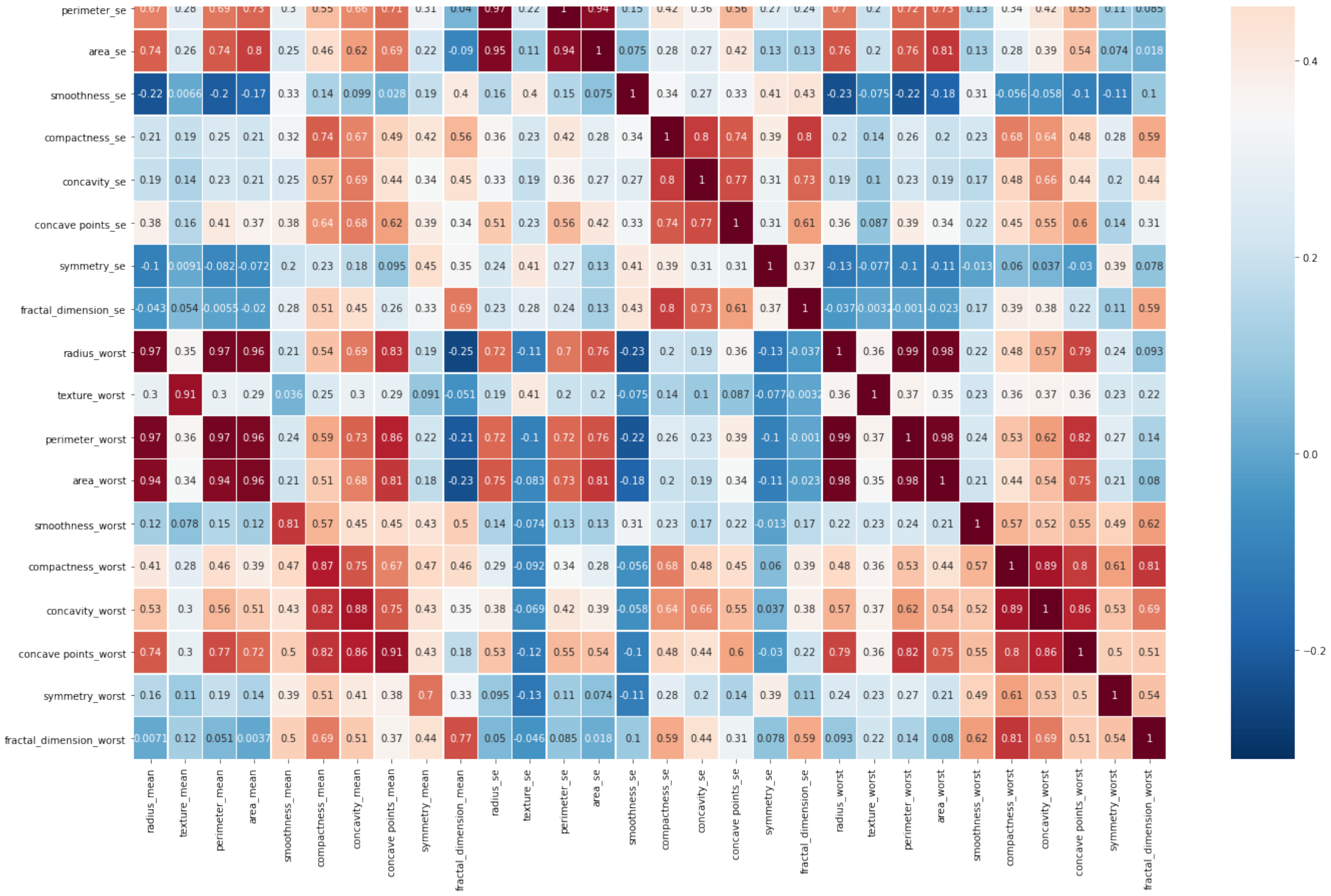


```
corr = data.corr()
corr.shape
```

```
    (30, 30)
```

```
plt.figure(figsize=(20,20))
sns.heatmap(corr, annot=True,cmap='RdBu_r',linewidths=.5)
plt.tight_layout()
```





LogisticRegression

```
from sklearn.model_selection import train_test_split
```

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

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random
```

```
# Import library for LogisticRegression
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
```

```
lg = LogisticRegression()
lg.fit(X_train, y_train)
```

```
/Users/bhumika/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_mod
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-regr
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression()
```

```
y_pred1 = lg.predict(X_test)
```

```
# Calculating the accuracy
acc_lg = round( metrics.accuracy_score(y_test, y_pred1) * 100, 2 )
print( 'Accuracy of Logistic Regression model : ', acc_lg )
```

```
Accuracy of Logistic Regression model :  95.91
```

+ Code + Text

Support Vector Classifier

```
from sklearn.model_selection import train_test_split
```

```
# Spliting target variable and independent variables
X = data.drop(['diagnosis'], axis = 1)
y = data['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.30, random_
from sklearn.svm import SVC
model=SVC()
model.fit(X_train,y_train)
```

```
    SVC()
```

```
y_pred2=model.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,y_pred2))
```

```
    [[107   1]
     [ 12  51]]
```

```
# Calculating the accuracy
acc_svc = round( metrics.accuracy_score(y_test, y_pred2) * 100, 2 )
print( 'Accuracy of SVC model : ', acc_svc )
```

```
    Accuracy of SVC model :  92.4
```

Decision Tree

```
# Import Decision tree classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Create a Decision tree classifier model
dt = DecisionTreeClassifier()

# Hyperparameter Optimization
parameters = {'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10, 50],
              'min_samples_split': [2, 3, 50, 100],
              'min_samples_leaf': [1, 5, 8, 10]
             }

# Run the grid search
grid_obj = GridSearchCV(dt, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dt = grid_obj.best_estimator_

# Train the model using the training sets
dt.fit(X_train, y_train)
```

```
    DecisionTreeClassifier(criterion='entropy', max_depth=10, max_features='sqr
                           min_samples_leaf=5)
```

```
y_pred3 = dt.predict(X_test)
```

```
# Calculating the accuracy
acc_dt = round( metrics.accuracy_score(y_test, y_pred3) * 100, 2 )
print( 'Accuracy of Decision Tree model : ', acc_dt )
```

```
    Accuracy of Decision Tree model :  92.98
```

Random Forest

```
# Import library of RandomForestClassifier model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Classifier
rf = RandomForestClassifier()

# Hyperparameter Optimization
parameters = {'n_estimators': [4, 6, 9, 10, 15],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1, 5, 8]
             }

# Run the grid search
grid_obj = GridSearchCV(rf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the rf to the best combination of parameters
rf = grid_obj.best_estimator_

# Train the model using the training sets
rf.fit(X_train,y_train)
```

```
    RandomForestClassifier(criterion='entropy', max_depth=10, max_features='log
                           min_samples_split=5, n_estimators=15)
```

```
y_pred4 = rf.predict(X_test)
```

```
# Calculating the accuracy
acc_rf = round( metrics.accuracy_score(y_test, y_pred4) * 100 , 2 )
print( 'Accuracy of Random Forest model : ', acc_rf )
```

```
    Accuracy of Random Forest model :  96.49
```

K - Nearest Neighbor

```
# Import library of KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier

# Create a KNN Classifier
knn = KNeighborsClassifier()

# Train the model using the training sets
knn.fit(X_train,y_train)
```

```
    KNeighborsClassifier()
```

```
# Prediction on test data
y_pred5 = knn.predict(X_test)
```

```
# Calculating the accuracy
acc_knn = round( metrics.accuracy_score(y_test, y_pred5) * 100, 2 )
print( 'Accuracy of KNN model : ', acc_knn )
```

```
    Accuracy of KNN model :  94.74
```

Evaluation and comparision of all the models

```
models = pd.DataFrame({
    'Model': ['Logistic Regression','Decision Tree', 'Random Forest', 'Support V
              'K - Nearest Neighbors'],
    'Score': [acc_lg, acc_dt, acc_rf, acc_svc, acc_knn]})
models.sort_values(by='Score',ascending=False)
```

	Model	Score
2	Random Forest	96.49
0	Logistic Regression	95.91
4	K - Nearest Neighbors	94.74
1	Decision Tree	92.98
3	Support Vector Classifiers	92.40