Support Vector Machines (SVM)

- Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection
- Goal is to create the best line or decision boundary called optimal hyperplane that can segregate n-dimensional space into classes
- SVM finds the hyperplane using `support vectors` (essential training tuples) and `margins` (defined by the support vectors)
- Support Vector Machines tries to produce linear decision boundaries
- is robust to outliers
- based on statistical approach
- can better handle highly-dimensional data
- used for classification as well as Regression problems, but mostly used for Classification
- is a Classifier, forward neural network, supervised learning algorithm
- Can handle linear as well as non-linear data
- SVM Algorithm types
    - Linear SVM
        - for linearly separable data
        - a single stright line can entirely divide the data points into their respective classes
    - Kernel SVM
        - for non-linear separable data
        - used when data points cannot be separated with a single straight line
        - original data id transformed by these kernel functions into a higher dimensional feature space where the features can be linearly separable

- `Hyperplane`
  - the best decision boundary differentiating classes
  - it can be linear/straight line for linearly separable data / 2 features
  - it can be non-linear plane / 2-D plane for non-linearly separable data / 3 features
  - equation is `w^Tx + b = 0`

- `Optimal Hyperplane`
  - the hyperplanee with maximum margin is called the optimal Hyperplane

- `Support Vectors`
  - vectors / data points closest to the hyperplane are called Support Vectors

- `Margin`
  - distance between the hyperplane and the support vectors (nearest data point) is called Margin
  - margin is expected to be as large as possible to find the optimal hyperplance
  - two types of margins

  1. Hard Margin
     - `Maximum Margin Hyperplane` or `Hard Margin Hyperplane` is a hyperplane that properly separates the data points of different categories without any misclassification
  2. Soft Margin
     - when data is not perfectly separable or contains outliers, then Soft margin is used
     - Each data point has a slack variable introduced by the soft margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations
     - it discovers compromise between increasing the margin and reducing violations

- `Kernel`
  - mathematical function in SVM used to map the original input data points to high dimensional feature spaces
  - makes it easy to find the hyperplane even if the data points are not linearly separable

  - Kernel refers to a method that allows us to apply linear classifiers to non-linear problems by mapping non-linear data into a higher-dimeansional space without the need to visit or understand that higher-dimensional space
  - Common kernel functions are linear, polynomial, rbf(radial basis function), sigmoid, neural net
- `C`
  - called as regularization parameter used to balance magin maximization and misclassification fines
  - penalty for misclassifications is decided by regularization parameter
  - more the value of `C` , stricter the penaly, so leading to smaller margin and fewer misclassifications
- Model performance can be altered by changing the value of hyperparameters which are `C (Regularization factor)`, `gamma`, and `kernel`

Advantages of SVM

- works better when data is linear, effective in high dimensional data
- robust to outliers
- can help us with image classification
- memory efficient as it uses subset of training data points in the decision function called support vectors
- different kernel functions can be specified and it is possible to specify custom kernels

Disadvantages of SVM

- Choosing a good kernel is not easy
- does not show good results on a bigger data set
- Hyperparameters of SVM are `C` and `gamma`, & it is not easy to fine tune these hyper parameters

Note for Kernel Trick

- when there is no separating plane
    - Use bigger set of features, makes use of kernel trick
        - it would make computation hopelessly slow, but using kernel trick we can make computation fast even with huge number of features
    - extend the definition of maximum margin to allow non-separating planes
        - this can be done by using the "Slack" variables, thereby using soft margin technique
        - slack variables are constrained to be non-negative

import libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D9data3.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D9data3.csv')
dataset.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
dataset.shape
```

```
(400, 3)
```

```
dataset.describe()
```

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              400 non-null    int64
 1   EstimatedSalary  400 non-null    int64
 2   Purchased        400 non-null    int64
dtypes: int64(3)
memory usage: 9.5 KB
```

Imputation (Null check)

```
dataset.isnull().sum()
```

```
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

identify X & Y

```
x = dataset.iloc[ : , :-1].values
x[:5]
```

```
array([[   19, 19000],
       [   35, 20000],
       [   26, 43000],
       [   27, 57000],
       [   19, 76000]], dtype=int64)
```

```
y = dataset.iloc[ : , -1].values
y[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

splitting

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

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
x_train[:5]
```

```
array([[    58, 144000],
       [    59,  83000],
       [    24,  55000],
       [    26,  35000],
       [    58,  38000]], dtype=int64)
```

```
y_train[:5]
```

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

Preprocessing

Feature scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_train = sc.fit_transform(x_train)
x_train[:5]
```

```
array([[ 1.92295008,  2.14601566],
       [ 2.02016082,  0.3787193 ],
       [-1.3822153 , -0.4324987 ],
       [-1.18779381, -1.01194013],
       [ 1.92295008, -0.92502392]])
```

```
x_test = sc.fit_transform(x_test)
x_test[:5]
```

```
array([[-0.49618606,  0.56021375],
       [ 0.2389044 , -0.59133674],
       [-0.03675452,  0.18673792],
       [-0.49618606,  0.31122986],
       [-0.03675452, -0.59133674]])
```

Linear SVM

- for linearly separable data

Modeling - Linear SVM

```
from sklearn.svm import SVC
```

```
lsvmclassifier = SVC(C=1, kernel='linear', random_state=0)
# C-Support Vector Classification
# C : Regularization parameter
```

Training - Linear SVM

```
lsvmclassifier.fit(x_train, y_train)
```

```
SVC
SVC(C=1, kernel='linear', random_state=0)
```

Prediction - Linear SVM

```
lsvmclassifier.predict(sc.transform([[30, 78000]]))
# prediction with custom test case
```

```
array([0], dtype=int64)
```

```
y_pred_linear = lsvmclassifier.predict(x_test)
y_pred_linear[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Linear SVM

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred_linear)
```

```
array([[52,  6],
       [ 3, 19]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_linear))
```

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.92        58
           1       0.76      0.86      0.81        22

    accuracy                           0.89        80
   macro avg       0.85      0.88      0.86        80
weighted avg       0.89      0.89      0.89        80
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred_linear)
```

```
0.8875
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, y_pred_linear)
```

```
0.76
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, y_pred_linear)
```

```
0.8636363636363636
```

Kernel SVM (c=1)

- for non-linearly separable data

Modeling - Kernel SVM (c=1)

```
from sklearn.svm import SVC
```

```
ksvmclassifier = SVC(C=1, kernel='rbf', random_state=0)
```

Training - Kernel SVM (c=1)

```
ksvmclassifier.fit(x_train, y_train)
```

```
▾ SVC
SVC(C=1, random_state=0)
```

Prediction - Kernel SVM (c=1)

```
y_pred_kernel = ksvmclassifier.predict(x_test)
y_pred_kernel[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation Kernel SVM (c=1)

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred_kernel)
```

```
array([[54,  4],
       [ 1, 21]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_kernel))
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.96        58
```

```
           1       0.84      0.95      0.89        22

    accuracy                           0.94        80
   macro avg       0.91      0.94      0.92        80
weighted avg       0.94      0.94      0.94        80
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred_kernel)
# higher accuracy means mis-classification has reduced
```

```
0.9375
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, y_pred_kernel)
```

```
0.84
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, y_pred_kernel)
```

```
0.9545454545454546
```

Kernel SVM (c=100)

Modeling - Kernel SVM (c=100)

```
from sklearn.svm import SVC
```

```
ksvmclassifier100 = SVC(C=100, kernel='rbf', random_state=0)
```

Training - Kernel SVM (c=100)

```
ksvmclassifier100.fit(x_train, y_train)
```

```
SVC
SVC(C=100, random_state=0)
```

Prediction - Kernel SVM(c=100)

```
y_pred_kernel100 =  ksvmclassifier100.predict(x_test)
y_pred_kernel100[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Kernel SVM (c=100)

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred_kernel100)
```

```
array([[54,  4],
       [ 1, 21]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_kernel100))
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.96        58
           1       0.84      0.95      0.89        22

    accuracy                           0.94        80
   macro avg       0.91      0.94      0.92        80
weighted avg       0.94      0.94      0.94        80
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred_kernel100)
```

```
0.9375
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, y_pred_kernel100)
```

```
0.84
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, y_pred_kernel100)
```

```
0.9545454545454546
```

SVM Application

import libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D13data1.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D13data1.csv')
dataset.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewnes of th DM-SN curv
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.24222
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.39358
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.17190
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.59366

```
dataset.shape
```

```
(17898, 9)
```

```
1 dataset.describe()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	k
count	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17
mean	111.079968	46.549532	0.477857	1.770279	12.614400	26.326515	
std	25.652935	6.843189	1.064040	6.167913	29.472897	19.470572	
min	5.812500	24.772042	-1.876011	-1.791886	0.213211	7.370432	
25%	100.929688	42.376018	0.027098	-0.188572	1.923077	14.437332	
50%	115.078125	46.947479	0.223240	0.198710	2.801839	18.461316	
75%	127.085938	51.023202	0.473325	0.927783	5.464256	28.428104	

```
1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 #   Column                                         Non-Null Count  Dtype  
---  ------                                         --------------  -----  
 0    Mean of the integrated profile                17898 non-null  float64
 1    Standard deviation of the integrated profile  17898 non-null  float64
 2    Excess kurtosis of the integrated profile     17898 non-null  float64
 3    Skewness of the integrated profile            17898 non-null  float64
 4    Mean of the DM-SNR curve                      17898 non-null  float64
 5    Standard deviation of the DM-SNR curve        17898 non-null  float64
 6    Excess kurtosis of the DM-SNR curve           17898 non-null  float64
 7    Skewness of the DM-SNR curve                  17898 non-null  float64
 8   target_class                                   17898 non-null  int64  
```

```
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

imputation

```
dataset.isnull().sum()
```

```
 Mean of the integrated profile                   0
 Standard deviation of the integrated profile     0
 Excess kurtosis of the integrated profile        0
 Skewness of the integrated profile               0
 Mean of the DM-SNR curve                         0
 Standard deviation of the DM-SNR curve           0
 Excess kurtosis of the DM-SNR curve              0
 Skewness of the DM-SNR curve                     0
target_class                                      0
dtype: int64
```

handling column names

```
dataset.columns
# column names have spaces that can be trimmed & names can be shortened
```

```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

stripping column names

```
dataset.columns = dataset.columns.str.strip()
# stripping column names
dataset.columns
```

```
Index(['Mean of the integrated profile',
       'Standard deviation of the integrated profile',
       'Excess kurtosis of the integrated profile',
       'Skewness of the integrated profile', 'Mean of the DM-SNR curve',
       'Standard deviation of the DM-SNR curve',
       'Excess kurtosis of the DM-SNR curve', 'Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

renaming column names

```
dataset.columns = ['Mean', 'SD', 'Kurtosis', 'Skewness', 'Mean_DM-SNR', 'SD_DM-SNR', 'Kurtosis_DM-SNR', 'Skewness_DM-SNR', 'target
dataset.columns
```

```
Index(['Mean', 'SD', 'Kurtosis', 'Skewness', 'Mean_DM-SNR', 'SD_DM-SNR',
       'Kurtosis_DM-SNR', 'Skewness_DM-SNR', 'target_class'],
      dtype='object')
```

checking dataset after handling column names

```
dataset.head()
```

	Mean	SD	Kurtosis	Skewness	Mean_DM-SNR	SD_DM-SNR	Kurtosis_DM-SNR	Skewness_DI SI
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.24222
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.39358
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.17190

understanding target

```
dataset['target_class'].value_counts()
```

```
target_class
0    16259
1     1639
Name: count, dtype: int64
```

EDA

```
import seaborn as sns
```

hist plot for target

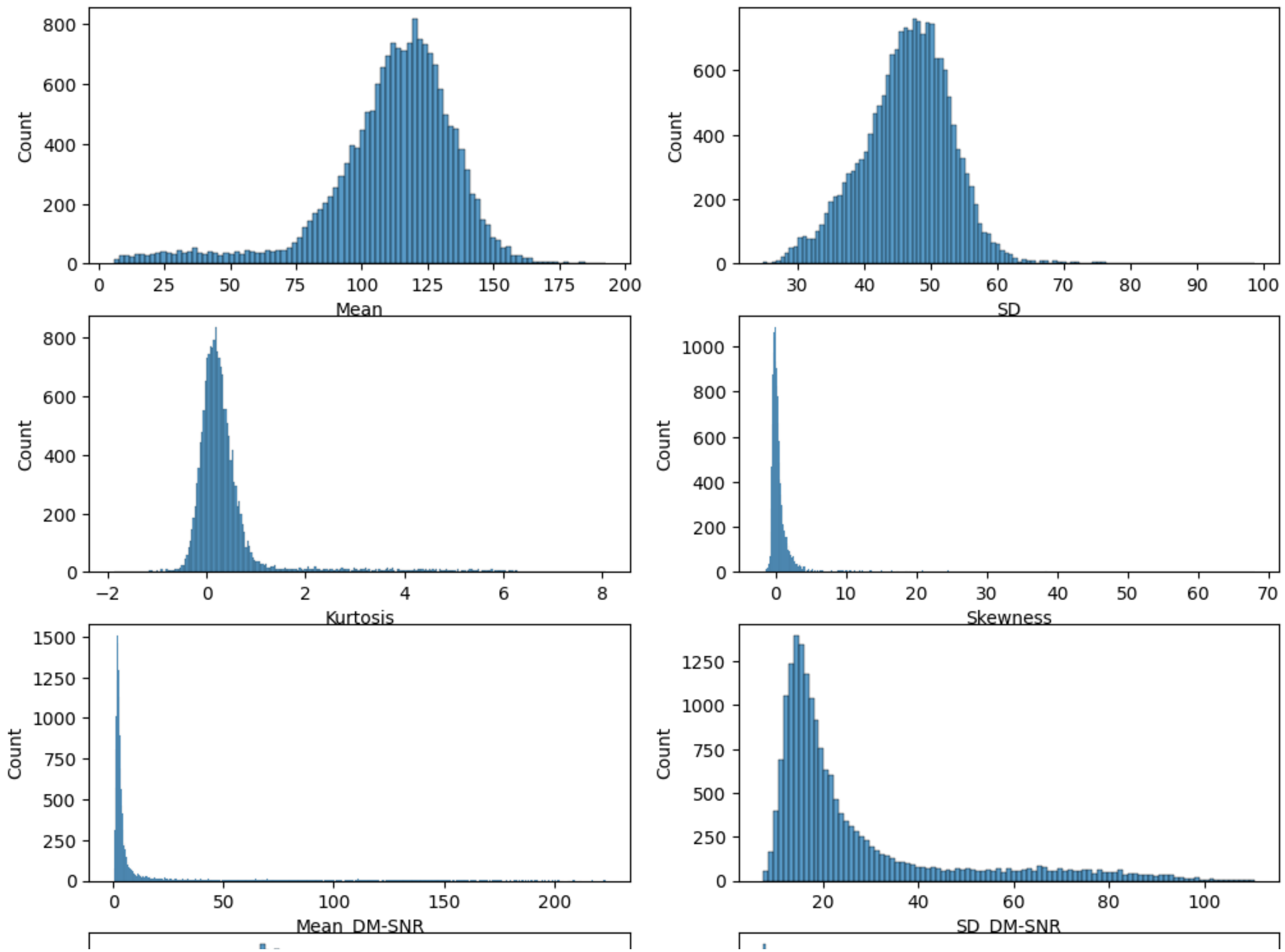
```
sns.histplot(x=dataset['target_class'])
```

```
<Axes: xlabel='target_class', ylabel='Count'>
```

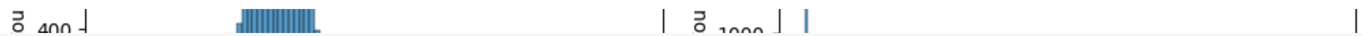
hist plot for features

```
plt.figure(figsize=(12, 12))
plt.subplot(4, 2, 1)
dp1 = sns.histplot(x=dataset['Mean'])

plt.subplot(4, 2, 2)
dp2 = sns.histplot(x=dataset['SD'])

plt.subplot(4, 2, 3)
dp3 = sns.histplot(x=dataset['Kurtosis'])

plt.subplot(4, 2, 4)
dp4 = sns.histplot(x=dataset['Skewness'])

plt.subplot(4, 2, 5)
dp5 = sns.histplot(x=dataset['Mean_DM-SNR'])
```

```

plt.subplot(4, 2, 6)
dp6 = sns.histplot(x=dataset['SD_DM-SNR'])

plt.subplot(4, 2, 7)
dp7 = sns.histplot(x=dataset['Kurtosis_DM-SNR'])

plt.subplot(4, 2, 8)
dp8 = sns.histplot(x=dataset['Skewness_DM-SNR'])

plt.show()
```

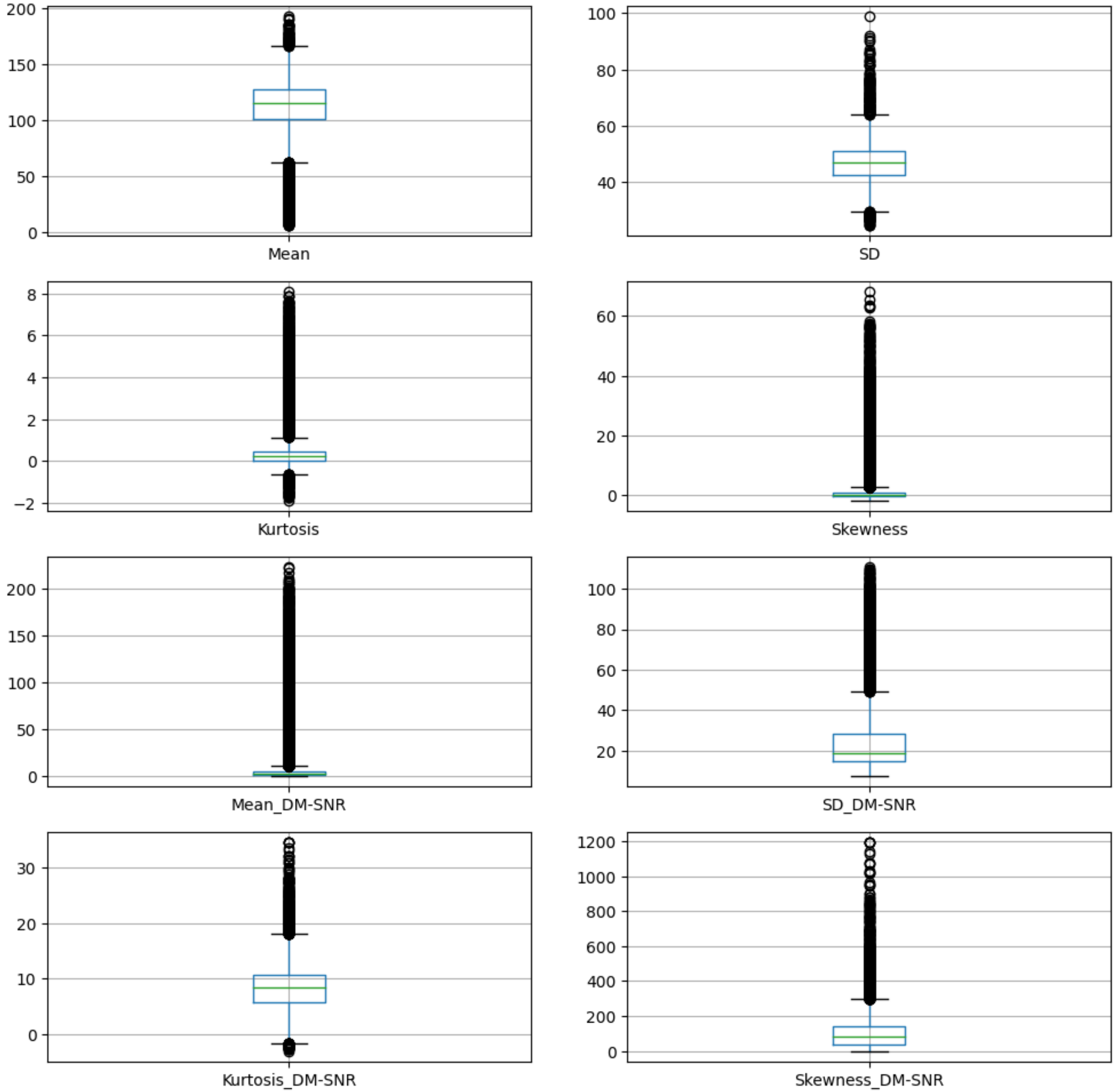


box plot for features



```
# sns.set_style(5)
plt.figure(figsize=(12, 12))
plt.subplot(4, 2, 1)
f1 = dataset.boxplot(column='Mean')

plt.subplot(4, 2, 2)
```

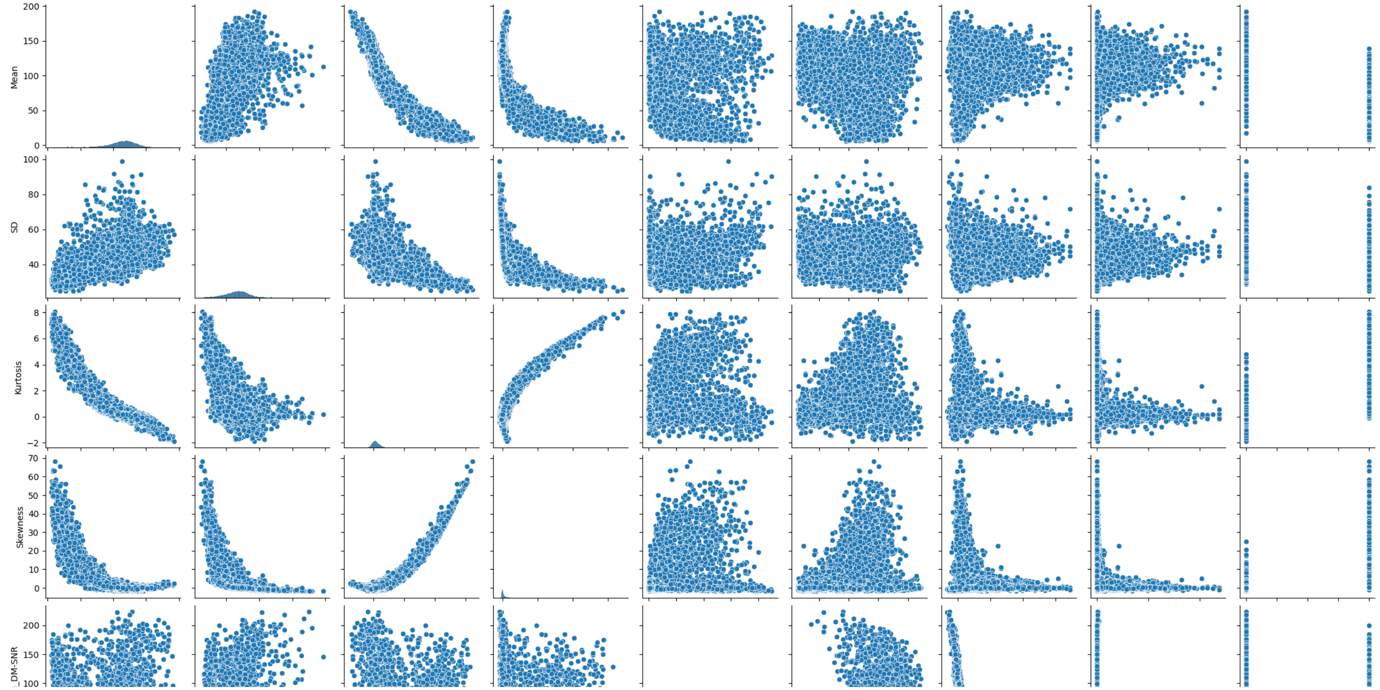
```
f2 = dataset.boxplot(column='SD')

plt.subplot(4, 2, 3)
f3 = dataset.boxplot(column='Kurtosis')

plt.subplot(4, 2, 4)
f4 = dataset.boxplot(column='Skewness')

plt.subplot(4, 2, 5)
f5 = dataset.boxplot(column='Mean_DM-SNR')

plt.subplot(4, 2, 6)
f6 = dataset.boxplot(column='SD_DM-SNR')

plt.subplot(4, 2, 7)
f7 = dataset.boxplot(column='Kurtosis_DM-SNR')

plt.subplot(4, 2, 8)
f8 = dataset.boxplot(column='Skewness_DM-SNR')

plt.show()
```


Mean
SD
Kurtosis
Skewness
Mean_DM-SNR
SD_DM-SNR
Kurtosis_DM-SNR
Skewness_DM-SNR

Pairplot

```
sns.pairplot(dataset)
```

```
c:\users\surya\appdata\local\programs\python\python39\lib\site-packages\seaborn\axisgrid.p
  self._figure.tight_layout(*args, **kwargs)
<seaborn.axisgrid.PairGrid at 0x1f76ae0cc10>
```



identify X & Y

```
x = dataset.iloc[ : , :-1].values
x[:2]
```

```
array([[140.5625    ,  55.68378214,  -0.23457141,  -0.6996484 ,
          3.19983278,  19.11042633,   7.97553179,  74.24222492],
       [102.5078125 ,  58.88243001,   0.46531815,  -0.51508791,
          1.67725752,  14.86014572,  10.57648674, 127.3935796 ]])
```

```
y = dataset.iloc[ : , -1].values
y[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Splitting

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

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
x_train[:2]
```

```
array([[ 1.20640625e+02,  4.78429616e+01,  2.57962577e-01,
        -9.06199080e-02,  8.04849498e+00,  3.51982345e+01,
         4.81978426e+00,  2.35283829e+01],
       [ 1.16554688e+02,  4.87029915e+01,  1.97625250e-01,
         2.32600230e-01,  3.04180602e+00,  1.66106785e+01,
         8.16618510e+00,  8.48467094e+01]])
```

```
y_train[:5]
```

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

PreProcessing

Feature scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_train = sc.fit_transform(x_train)
x_train[:2]
```

```
array([[ 0.37710226,  0.18643192, -0.21304291, -0.30376654, -0.16236693,
         0.44549441, -0.76498842, -0.76290015],
       [ 0.21958464,  0.31196312, -0.26882602, -0.25234846, -0.32948472,
        -0.50223528, -0.02369848, -0.18468066]])
```

```
x_test = sc.fit_transform(x_test)
x_test[:2]
```

```
array([[-0.39574205, -0.106581  , -0.2062192 , -0.23805528, -0.33389687,
        -0.60628785,  0.24052962,  0.09688056],
       [ 0.3324866 ,  0.77538991, -0.28588937, -0.3480077 , -0.29508947,
        -0.32389767, -0.10358973, -0.30403506]])
```

Linear SVM

Modeling - Linear SVM

```
from sklearn.svm import SVC
```

```
lsvmc = SVC(C=1, kernel='linear', random_state=0)
```

Training - Linear SVM

```
lsvmc.fit(x_train, y_train)
```

```
SVC
SVC(C=1, kernel='linear', random_state=0)
```

Prediction - Linear SVM

```
lin_y_pred = lsvmc.predict(x_test)
lin_y_pred[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Linear SVM

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, lin_y_pred)
```

```
array([[3289,   17],
       [  37,  237]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, lin_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.93      0.86      0.90       274

    accuracy                           0.98      3580
   macro avg       0.96      0.93      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, lin_y_pred)
```

```
0.9849162011173185
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, lin_y_pred)
```

```
0.9330708661417323
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, lin_y_pred)
```

```
0.864963503649635
```

Kernel SVM (c=1, kernel='rbf')

Modeling - Kernel SVM (c=1, kernel='rbf')

```
from sklearn.svm import SVC
```

```
ksvmc = SVC(C=1, kernel='rbf', random_state=0)
```

Training - Kernel SVM (c=1, kernel='rbf')

```
ksvmc.fit(x_train, y_train)
```

```
SVC
SVC(C=1, random_state=0)
```

Prediction - Kernel SVM (c=1, kernel='rbf')

```
kernel_y_pred = ksvmc.predict(x_test)
kernel_y_pred[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Kernel SVM (c=1, kernel='rbf')

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, kernel_y_pred)
```

```
array([[3285,   21],
       [  40,  234]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, kernel_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.92      0.85      0.88       274

    accuracy                           0.98      3580
   macro avg       0.95      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, kernel_y_pred)
```

```
0.9829608938547486
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, kernel_y_pred)
```

```
0.9176470588235294
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, kernel_y_pred)
```

```
0.8540145985401459
```

roc_curve

```
from sklearn.metrics import roc_curve
```

```
fpr, tpr, thresholds = roc_curve(y_test, kernel_y_pred)
```

```
print("False Positive Rate: ", fpr)
print("True Positive Rate: ", tpr)
print("Thresholds: ", thresholds)
```

```
False Positive Rate:  [0.         0.00635209 1.        ]
True Positive Rate:  [0.        0.8540146 1.       ]
Thresholds:  [inf  1.  0.]
```

```
plt.plot(fpr, tpr, linewidth=2)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve - fpr vs tpr")
plt.show()
```

ROC curve - fpr vs tpr

roc_auc_score

```
from sklearn.metrics import roc_auc_score
```

```
roc_auc_score(y_test, kernel_y_pred)
# The higher the AUC, the better the model's performance at distinguishing
# between the positive and negative classes.
# An AUC score of 1 means the classifier can perfectly distinguish
# between all the Positive and the Negative class points.
```

```
0.9238312557129043
```

Kernel SVM (c=100, kernel='poly')

Modeling - Kernel SVM (c=100, kernel='poly')

tpr

```
from sklearn.svm import SVC
```

```
ksvm100poly = SVC(C=1, kernel='poly', random_state=0)
```

Training - Kernel SVM (c=100, kernel='poly')

```
ksvm100poly.fit(x_train, y_train)
```

SVC

Prediction - Kernel SVM (c=100, kernel='poly')

```
y_pred_ksvm100poly = ksvm100poly.predict(x_test)
y_pred_ksvm100poly[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Kernel SVM (c=100, kernel='poly')

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred_ksvm100poly)
```

```
array([[3286,   20],
       [  42,  232]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_ksvm100poly))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
```

```
           1       0.92      0.85      0.88       274

    accuracy                           0.98      3580
   macro avg       0.95      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred_ksvm100poly)
```

```
0.9826815642458101
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, y_pred_ksvm100poly)
```

```
0.9206349206349206
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, y_pred_ksvm100poly)
```

```
0.8467153284671532
```

roc_curve

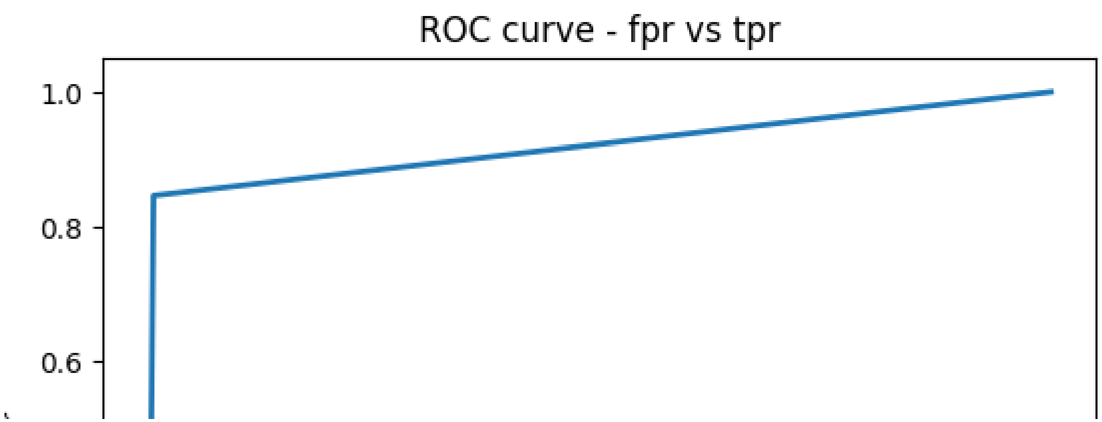
```
from sklearn.metrics import roc_curve
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_ksvm100poly)
```

```
print("False Positive Rate: ", fpr)
print("True Positive Rate: ", tpr)
print("Thresholds: ", thresholds)
```

```
False Positive Rate:  [0.         0.00604961 1.        ]
True Positive Rate:  [0.         0.84671533 1.        ]
Thresholds:  [inf  1.  0.]
```

```
plt.plot(fpr, tpr, linewidth=2)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve - fpr vs tpr")
plt.show()
```



roc_auc_score

```
from sklearn.metrics import roc_auc_score
```

```
roc_auc_score(y_test, y_pred_ksvm100poly)
# The higher the AUC, the better the model's performance at distinguishing
# between the positive and negative classes.
# An AUC score of 1 means the classifier can perfectly distinguish
# between all the Positive and the Negative class points.
```

```
0.9203328608457969
```

Kernel SVM (c=100, kernel='sigmoid')

Modeling - Kernel SVM (c=100, kernel='sigmoid')

```
from sklearn.svm import SVC
```

```
ksvm100sigmoid = SVC(C=100, kernel='sigmoid', random_state=0)
```

Training - Kernel SVM (c=100, kernel='sigmoid')

```
ksvm100sigmoid.fit(x_train, y_train)
```

```
SVC
SVC(C=100, kernel='sigmoid', random_state=0)
```

Prediction - Kernel SVM (c=100, kernel='sigmoid')

```
y_pred_ksvm100sigmoid = ksvm100sigmoid.predict(x_test)
y_pred_ksvm100sigmoid[:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Evaluation - Kernel SVM (c=100, kernel='poly')

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred_ksvm100sigmoid)
```

```
array([[3013,  293],
       [ 188,   86]], dtype=int64)
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred_ksvm100sigmoid))
```

```
              precision    recall  f1-score   support

           0       0.94      0.91      0.93      3306
           1       0.23      0.31      0.26       274

    accuracy                           0.87      3580
   macro avg       0.58      0.61      0.59      3580
weighted avg       0.89      0.87      0.88      3580
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred_ksvm100sigmoid)
```

```
0.8656424581005586
```

precision_score

```
from sklearn.metrics import precision_score
```

```
precision_score(y_test, y_pred_ksvm100sigmoid)
```

```
0.22691292875989447
```

recall_score

```
from sklearn.metrics import recall_score
```

```
recall_score(y_test, y_pred_ksvm100sigmoid)
```

```
0.31386861313868614
```

roc_curve

- Reciever Operating Characteristic curve
- can be used to evaluate classification, mostly used for binary classifiers such as Logistic regression/classification or sigmoid classification

```
from sklearn.metrics import roc_curve
```

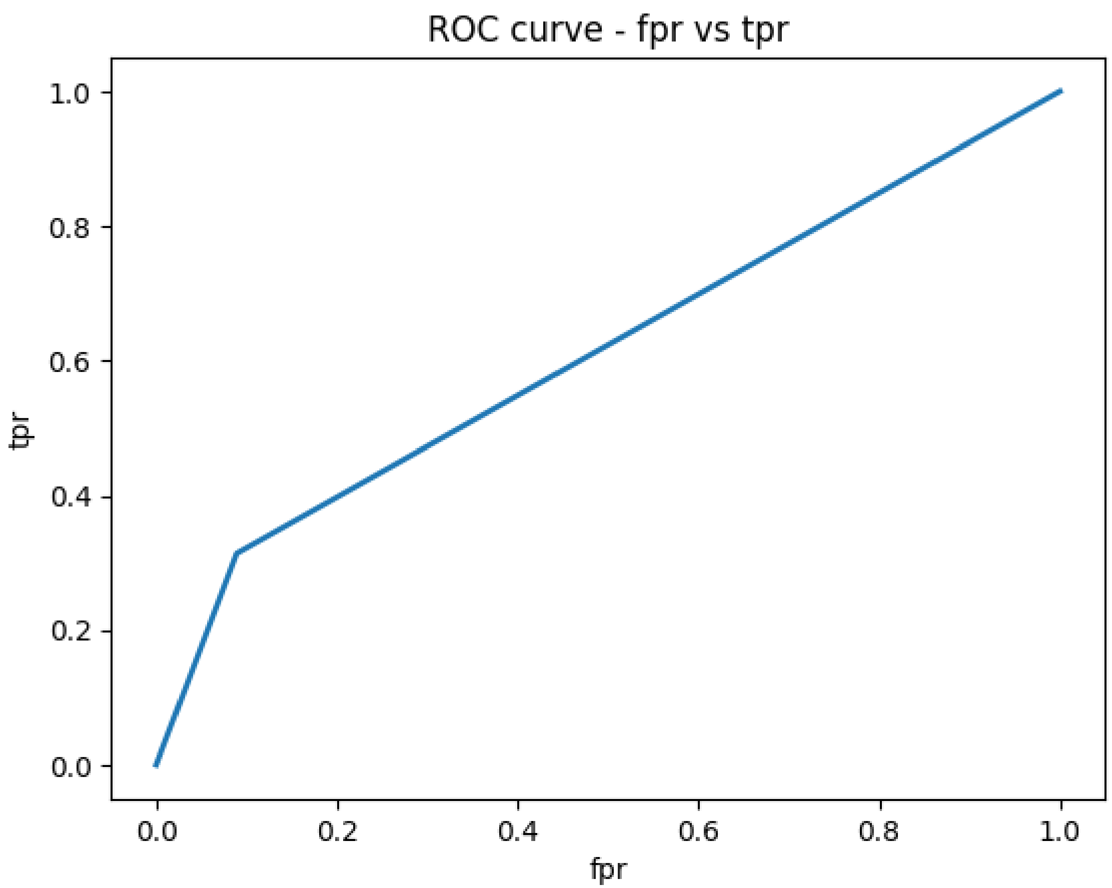
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_ksvm100sigmoid)
```

```
print("False Positive Rate: ", fpr)
print("True Positive Rate: ", tpr)
print("Thresholds: ", thresholds)
```

```
False Positive Rate:  [0.         0.08862674 1.        ]
True Positive Rate:  [0.         0.31386861 1.        ]
Thresholds:  [inf  1.  0.]
```

```
plt.plot(fpr, tpr, linewidth=2)
plt.xlabel("fpr")
```

```
plt.ylabel("tpr")
plt.title("ROC curve - fpr vs tpr")
plt.show()
```



roc_auc_score

- Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

```
from sklearn.metrics import roc_auc_score
```

```
roc_auc_score(y_test, y_pred_ksvm100sigmoid)
# Area Under the Receiver Operating Characteristic Curve (ROC AUC)
# from prediction scores.
# The higher the AUC, the better the model's performance at distinguishing
# between the positive and negative classes.
# An AUC score of 1 means the classifier can perfectly distinguish
# between all the Positive and the Negative class points.
```

```
0.612620936938369
```