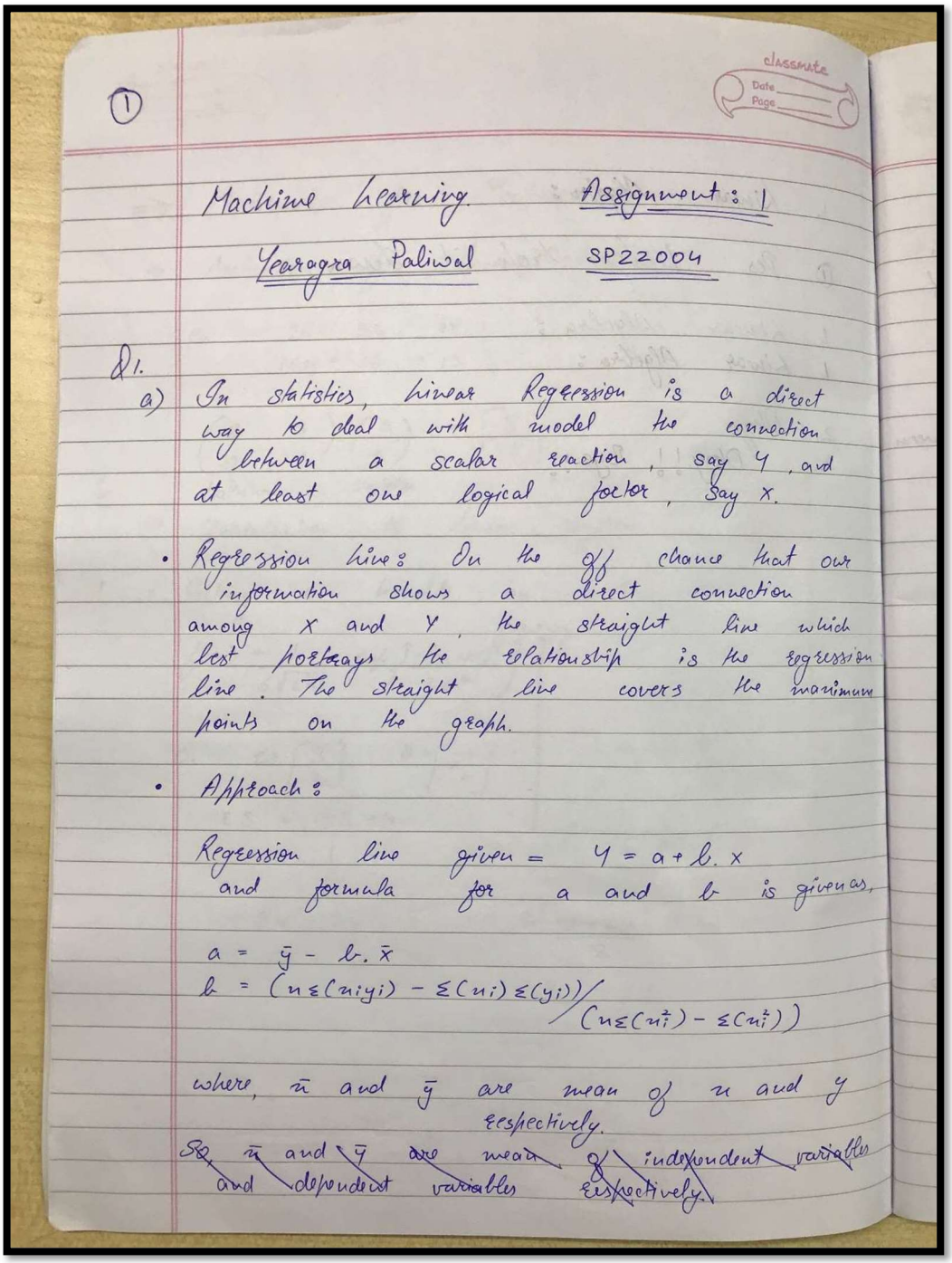
Assignment 1
(Machine Learning)

Submitted By:
Yearagra Paliwal
(SP22004)

Screen Shot of the Question 1:

A.



Machine Learning Assignment : 1

Yearagra Paliwal SP22004

Q1.

a) In statistics, linear Regression is a direct way to deal with model the connection between a scalar reaction, say Y, and at least one logical factor, say X.

- Regression line: On the off chance that our information shows a direct connection among X and Y, the straight line which best portrays the relationship is the regression line. The straight line covers the maximum points on the graph.

- Approach:

Regression line given = $Y = a + b.X$
and formula for a and b is given as,

$$a = \bar{y} - b.\bar{x}$$

$$b = (n\Sigma(x_i y_i) - \Sigma(x_i)\Sigma(y_i)) / (n\Sigma(x_i^2) - \Sigma(x_i^2))$$

where, $\bar{x}$ and $\bar{y}$ are mean of x and y respectively.

So, $\bar{x}$ and $\bar{y}$ are mean of independent variables and dependent variables respectively.

B.

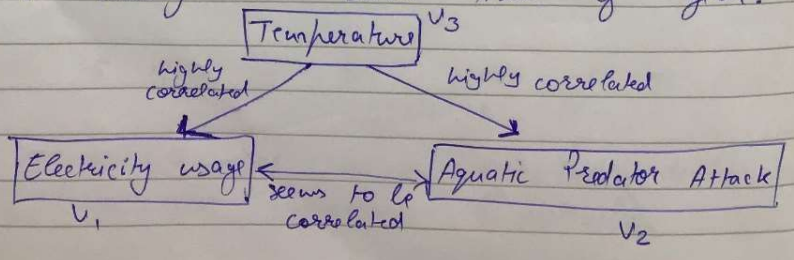
(2)

b) In statistics, a third variable problem happens when a noticed relationship between two ~~factor~~ variables can really be made sense of by a third variable that hasn't been represented

At the point when this ~~third~~ third variable isn't considered, the relationship between the two variables under study can be misdirecting and surprisingly confounding.

- for example:
  Electricity usage and aquatic predators attack.

A research finds that electricity usage and aquatic predators attacks are highly positively correlated.
However, these two variables are only correlated because they both have a high correlation with third variable: temperature.
So,
When its warmer out, ~~two~~ usage of electricity increases and more people swim in the ocean which explains why the value for both electricity and aquatic predator attack tend to increase during the same time of year.



C.

(3)

c) Weak law of large Number

For random variable x_1, x_2, x_n the sample mean, denoted by $\bar{x}$

$$\bar{x} = x_1 + x_2 + x_3 + \dots + x_n$$

In addition,

$$\text{Var}(x) = \text{Var}\left(\frac{x_1 + x_2 + \dots + x_n}{n}\right)$$

$$= \text{Var}\left(\frac{x_1}{n}\right) + \dots + \text{var}\left(\frac{x_n}{n}\right)$$

$$= \frac{\sigma^2}{n^2} + \dots + \frac{\sigma^2}{n^2}$$

$$= \frac{\sigma^2}{n}$$

By Chebyshev inequality,

$$P(|x - \mu| \geq \epsilon) \leq \frac{\text{Var}(x)}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2}$$

As per Bernoulli's Theorem

$$\lim_{n \to \infty} P(|x - \mu| \geq \epsilon) = 0$$

D.

(4)

d) Maximum A Posteriori (MAP)

An elective estimator is the MAP estimator, which finds the boundary theta that maximizes the posterior. As per the Bayes rule, the ~~back~~ posterior can be disintegrated into the result of the probability and earlier. The MAP estimator starts with this thought and is characterised as underneath.

$$\theta_{MAP} = \arg\max_{\theta} p(\theta/x)$$

$$= \arg\max_{\theta} p(x/\theta) \cdot p(\theta) \quad \{ \text{Bayes Rule}$$

$$= \arg\max_{\theta} \log p(x/\theta) + \log p(\theta) \quad \{ \text{logarithm}$$

$$\boxed{\theta_{MAP} = \arg\max_{\theta} \sum_{i=1}^{m} \{ \log p(x^{(i)}/\theta) \} + \log p(\theta)}$$

Now, Gaussian Prior $P(u) = N(u, \lambda^2)$

$$P(u/u, \lambda) = \frac{1}{\lambda\sqrt{2\pi}} e^{\frac{-(u-u)^2}{2\lambda^2}}$$

$$\boxed{\hat{u}_{MAP} = \frac{\frac{1}{\sigma^2}\sum_{i=1}^{u} u_i + u/\lambda^2}{u/\sigma^2 + \frac{1}{\lambda^2}}}$$

Code Screen Shots Question 3:

DATASET

```
import io
df = pd.read_csv(io.BytesIO(uploaded['Dry_Bean_Dataset.csv']))
df
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	0.913358	0.007332	0.00314
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272751	0.783968	0.984986	0.887034	0.953861	0.006979	0.00356
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	0.908774	0.007244	0.00304
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	0.928329	0.007017	0.00321
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	0.970516	0.006697	0.00366
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	42508	231.515799	0.714574	0.990331	0.916603	0.801865	0.006858	0.00174
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	42494	231.526798	0.799943	0.990752	0.922015	0.822252	0.006688	0.00188
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	42569	231.631261	0.729932	0.989899	0.918424	0.822730	0.006681	0.00188
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	42667	231.653247	0.705389	0.987813	0.907906	0.817457	0.006724	0.00185
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	42600	231.686223	0.788962	0.989648	0.888380	0.784997	0.007001	0.00164

13611 rows × 17 columns

REPORT

```
[5] profile = ProfileReport(df, title="Data Analysis Report", html={'style':{'full_width':False}})
profile.to_notebook_iframe()
```

Summarize dataset: 100% 286/286 [00:53<00:00, 4.19it/s, Completed]
Generate report structure: 100% 1/1 [00:09<00:00, 9.88s/it]
Render HTML: 100% 1/1 [00:09<00:00, 9.21s/it]

Data Analysis Report — Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Overview

Overview | Alerts 18 | Reproduction

Dataset statistics

Number of variables	17
Number of observations	13611

Variable types

Numeric	16
Categorical	1

Data Analysis Report — Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Overview

Overview | Alerts 18 | Reproduction

Dataset statistics

Number of variables	17
Number of observations	13611
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	68
Duplicate rows (%)	0.5%
Total size in memory	1.8 MiB
Average record size in memory	136.0 B

Variable types

Numeric	16
Categorical	1

Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

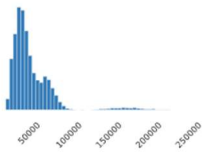
Variables

Area

Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION

Distinct	12011	Minimum	20420
Distinct (%)	88.2%	Maximum	254616
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	53048.28455	Memory size	106.5 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	20420
5-th percentile	27660.5
Q1	36328
median	44652

Descriptive statistics

Standard deviation	29324.09572
Coefficient of variation (CV)	0.552781225
Kurtosis	10.80081397
Mean	53048.28455

Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Mean 53048.28455 Memory size 106.5 KiB

Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	20420
5-th percentile	27660.5
Q1	36328
median	44652
Q3	61332
95-th percentile	89824.5
Maximum	254616
Range	234196
Interquartile range (IQR)	25004

Descriptive statistics

Standard deviation	29324.09572
Coefficient of variation (CV)	0.552781225
Kurtosis	10.80081397
Mean	53048.28455
Median Absolute Deviation (MAD)	10386
Skewness	2.952930971
Sum	722040201
Variance	859902589.6
Monotonicity	Not monotonic

Activate Windows
Go to Settings to activate Windows.

Perimeter

Distinct	13351	Minimum	524.736

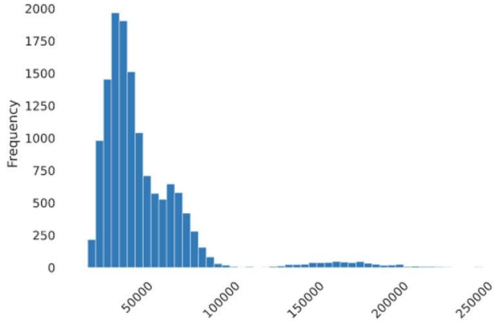
Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Mean 53048.28455 Memory size 106.5 KiB

Toggle details

Statistics | Histogram | Common values | Extreme values



Histogram with fixed size bins (bins=50)

Activate Windows
Go to Settings to activate Windows.

Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Mean 53048.28455 Memory size 106.5 KiB

Toggle details

Statistics | Histogram | Common values | Extreme values

Value	Count	Frequency (%)
35442	4	< 0.1%
38426	4	< 0.1%
34774	4	< 0.1%
40504	4	< 0.1%
52266	4	< 0.1%
34594	4	< 0.1%
33518	4	< 0.1%
28122	4	< 0.1%
38273	4	< 0.1%
36109	4	< 0.1%
Other values (12001)	13571	99.7%

Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Toggle details

Statistics | Histogram | Common values | Extreme values

Minimum 10 values | Maximum 10 values

Value	Count	Frequency (%)
20420	1	< 0.1%
20464	1	< 0.1%
20548	1	< 0.1%
20711	1	< 0.1%
20786	1	< 0.1%
20942	1	< 0.1%
21101	1	< 0.1%
21314	1	< 0.1%
21348	1	< 0.1%
21397	1	< 0.1%

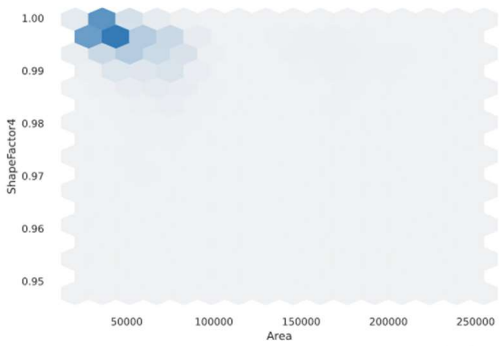
Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Interactions

Area
Perimeter
MajorAxisLength
MinorAxisLength
AspectRation
Eccentricity
ConvexArea
EquivDiameter
Extent
Solidity
roundness
Compactness
ShapeFactor1
ShapeFactor2
ShapeFactor3
ShapeFactor4

ShapeFactor4
Area
Perimeter
MajorAxisLength
MinorAxisLength
AspectRation
Eccentricity
ConvexArea
EquivDiameter
Extent
Solidity
roundness
Compactness
ShapeFactor1
ShapeFactor2
ShapeFactor3



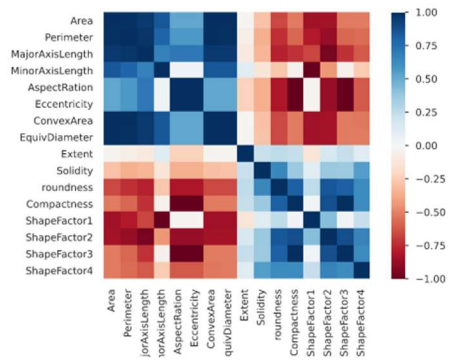
Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Correlations

Spearman's ρ | Pearson's r | Kendall's τ | Phik (φk)

Toggle correlation descriptions

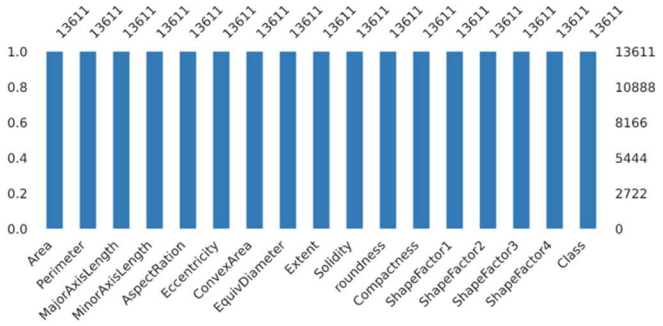


Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Missing values

Count | Matrix



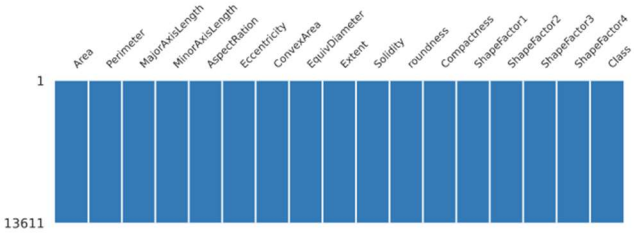
A simple visualization of nullity by column.

Data Analysis Report

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

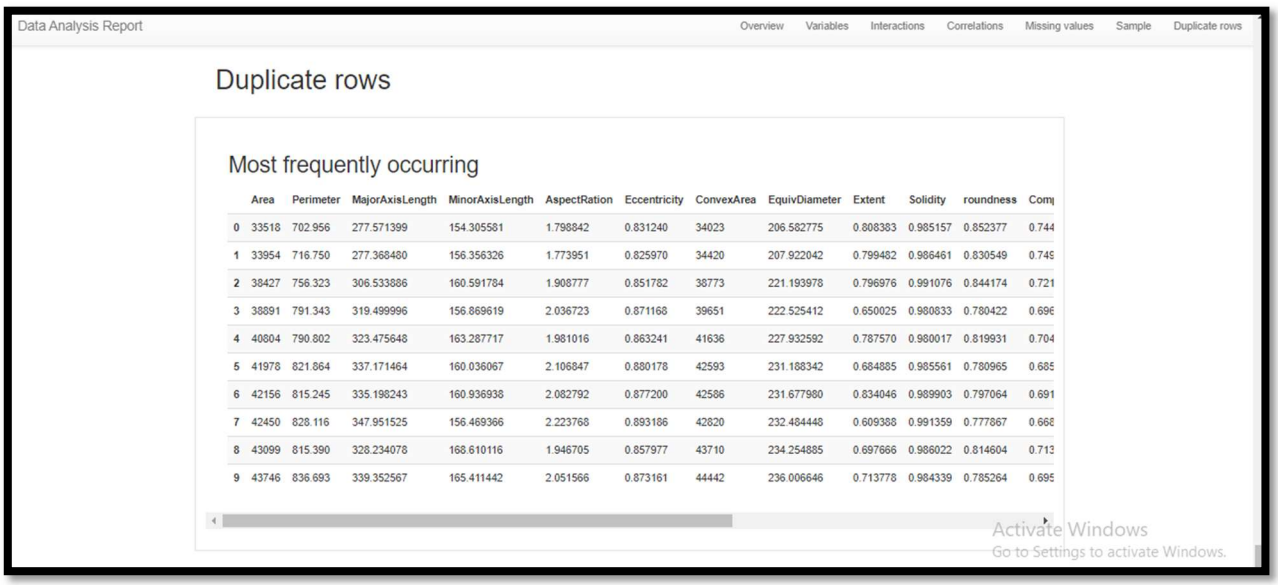
Missing values

Count | Matrix



Nullity matrix is a data-dense display which lets you quickly visually pick out patterns in data completion.

Sample



SAMPLING TEST/TRAIN

```
X= df.drop(columns='Class')
Y=df['Class']
def split_data(X,Y):
  X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
  data_splitted= [X_train, X_test,Y_train,Y_test]
  return data_splitted
arr=split_data(X,Y)
X_train=arr[0]
X_test=arr[1]
Y_train=arr[2]
Y_test=arr[3]
print(X.shape , X_train.shape , X_test.shape)
print(Y.shape,Y_train.shape , Y_test.shape)
```

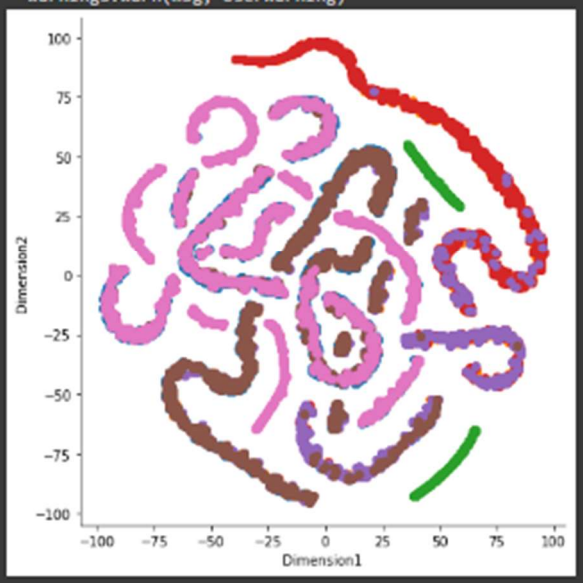
```
(13611, 16) (10888, 16) (2723, 16)
(13611,) (10888,) (2723,)
```

T-SNE

```
[8]  points=X[0:13611:]
     data_label=Y[0:13611:]
     tsne = TSNE(n_components=2, random_state = 10)
     tsne_emb = tsne.fit_transform(points)

     tsne_data=np.vstack((tsne_emb.T , data_label)).T
     tsne_df=pd.DataFrame(data=tsne_data, columns=("Dimension1","Dimension2","label"))
     sns.FacetGrid(tsne_df , hue="label" , size=6).map(plt.scatter , 'Dimension1','Dimension2')
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The defa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The defa
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `size` parame
  warnings.warn(msg, UserWarning)
```



NAIVE BAYES (Bernoulli)

```
BernNB = BernoulliNB(binarize = True)
BernNB.fit(X_train, Y_train)
print(BernNB)
print()

Y_expect = Y_test
Y_pred = BernNB.predict(X_test)
print ("Accuracy :",accuracy_score(Y_expect, Y_pred))
print()
print ("Precision: ",precision_score(Y_test, Y_pred, average='macro'))
print()
print ("F1: ",f1_score(Y_test, Y_pred, average='macro'))
print()
print ("Recall: ",recall_score(Y_expect, Y_pred, average=None))
```

```
BernoulliNB(binarize=True)

Accuracy : 0.25156077855306647

Precision:  0.035937254079009495

F1:  0.05742790073775989

Recall:  [0. 0. 0. 1. 0. 0. 0.]
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.p
  _warn_prf(average, modifier, msg_start, len(result))
```

NAIVE BAYES (Gaussian)

```
GausNB = GaussianNB()
GausNB.fit(X_train, Y_train)
print (GausNB)
print()

Y_expect = Y_test
Y_pred =GausNB.predict(X_test)
print ("Accuracy: ",accuracy_score(Y_expect, Y_pred))
print()
print ("Precision: ",precision_score(Y_expect, Y_pred, average='macro'))
print()
print ("F1: ",f1_score(Y_expect, Y_pred, average='macro'))
print()
print ("Recall: ",recall_score(Y_expect, Y_pred, average=None))
```

```
GaussianNB()

Accuracy:  0.7623944179214102

Precision:  0.7709230622173386

F1:  0.7647969270732836

Recall:  [0.45833333 1.         0.76948052 0.84963504 0.78947368 0.71359223
 0.78125    ]
```

PCA FOR 1st COMPONENT

```
#PCA
pca = PCA(n_components = 4, random_state = 10)
X_pca = pca.fit_transform(X)
X_pca = pca.transform(X)
print(X.shape)
print(X_pca.shape)

#Splitting of reduced dimension dataset
X= X_pca
Y= Y
def split_data(X,Y):
  X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
  data_splitted= [X_train, X_test,Y_train,Y_test]
  return data_splitted
arr=split_data(X,Y)
X_train=arr[0]
X_test=arr[1]
Y_train=arr[2]
Y_test=arr[3]
print(X.shape , X_train.shape , X_test.shape)
print(Y.shape,Y_train.shape , Y_test.shape)

split_data(X,Y)

#Training
logistic_regression = LogisticRegression(max_iter=10000)
logistic_regression.fit(X_train,Y_train)

Y_pred_train=logistic_regression.predict(X_train)
Y_pred_test=logistic_regression.predict(X_test)

print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
print()

print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16)
(13611, 4)
(13611, 4) (10888, 4) (2723, 4)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  90.34153507161218

Precision for Test Set:  91.71079963859577

F1 for Test Set:  91.51792264470042

Recall for Test Set:  [ 85.41666667 100.          91.88311688  93.43065693  94.73684211
  93.93203883  80.51470588]
```

PCA FOR 2nd COMPONENT

```
#PCA
pca = PCA(n_components = 6, random_state = 10)
X_pca = pca.fit_transform(X)
X_pca = pca.transform(X)
print(X.shape)
print(X_pca.shape)

#Splitting of reduced dimension dataset
X= X_pca
Y= Y
def split_data(X,Y):
  X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
  data_splitted= [X_train, X_test,Y_train,Y_test]
  return data_splitted
arr=split_data(X,Y)
X_train=arr[0]
X_test=arr[1]
Y_train=arr[2]
Y_test=arr[3]
print(X.shape , X_train.shape , X_test.shape)
print(Y.shape,Y_train.shape , Y_test.shape)

split_data(X,Y)

#Training
logistic_regression = LogisticRegression(max_iter=10000)
logistic_regression.fit(X_train,Y_train)

Y_pred_train=logistic_regression.predict(X_train)
Y_pred_test=logistic_regression.predict(X_test)

print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
print()

print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16)
(13611, 6)
(13611, 6) (10888, 6) (2723, 6)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  92.10429673154609

Precision for Test Set:  93.78208779772119

F1 for Test Set:  93.31164604405993

Recall for Test Set:  [ 86.45833333 100.         95.45454545  93.13868613  94.21052632
  94.17475728  87.31617647]
```

PCA FOR 3rd COMPONENT

```
[112] #PCA
      pca = PCA(n_components = 8, random_state = 10)
      X_pca = pca.fit_transform(X)
      X_pca = pca.transform(X)
      print(X.shape)
      print(X_pca.shape)

      #Splitting of reduced dimension dataset
      X= X_pca
      Y= Y
      def split_data(X,Y):
        X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
        data_splitted= [X_train, X_test,Y_train,Y_test]
        return data_splitted
      arr=split_data(X,Y)
      X_train=arr[0]
      X_test=arr[1]
      Y_train=arr[2]
      Y_test=arr[3]
      print(X.shape , X_train.shape , X_test.shape)
      print(Y.shape,Y_train.shape , Y_test.shape)

      split_data(X,Y)

      #Training
      logistic_regression = LogisticRegression(max_iter=10000)
      logistic_regression.fit(X_train,Y_train)

      Y_pred_train=logistic_regression.predict(X_train)
      Y_pred_test=logistic_regression.predict(X_test)

      print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
      print()

      print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
      print()

      print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
      print()

      print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16)
(13611, 8)
(13611, 8) (10888, 8) (2723, 8)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  92.14102093279472

Precision for Test Set:  93.76543015299886

F1 for Test Set:  93.3508357256948

Recall for Test Set:  [ 87.5        100.         94.80519481  92.99270073  94.21052632
  94.17475728  87.5       ]
```

PCA FOR 4th COMPONENT

```
#PCA
pca = PCA(n_components = 10, random_state = 10)
X_pca = pca.fit_transform(X)
X_pca = pca.transform(X)
print(X.shape)
print(X_pca.shape)

#Splitting of reduced dimension dataset
X= X_pca
Y= Y
def split_data(X,Y):
  X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
  data_splitted= [X_train, X_test,Y_train,Y_test]
  return data_splitted
arr=split_data(X,Y)
X_train=arr[0]
X_test=arr[1]
Y_train=arr[2]
Y_test=arr[3]
print(X.shape , X_train.shape , X_test.shape)
print(Y.shape,Y_train.shape , Y_test.shape)

split_data(X,Y)

#Training
logistic_regression = LogisticRegression(max_iter=10000)
logistic_regression.fit(X_train,Y_train)

Y_pred_train=logistic_regression.predict(X_train)
Y_pred_test=logistic_regression.predict(X_test)

print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
print()

print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16)
(13611, 10)
(13611, 10) (10888, 10) (2723, 10)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  92.06757253029747

Precision for Test Set:  93.74251756373242

F1 for Test Set:  93.29720585313191

Recall for Test Set:  [ 86.80555556 100.         95.45454545  92.99270073  94.21052632
  94.17475728  87.13235294]
```

PCA FOR 5th COMPONENT

```
[116] #PCA
      pca = PCA(n_components = 12, random_state = 10)
      X_pca = pca.fit_transform(X)
      X_pca = pca.transform(X)
      print(X.shape)
      print(X_pca.shape)

      #Splitting of reduced dimension dataset
      X= X_pca
      Y= Y
      def split_data(X,Y):
        X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
        data_splitted= [X_train, X_test,Y_train,Y_test]
        return data_splitted
      arr=split_data(X,Y)
      X_train=arr[0]
      X_test=arr[1]
      Y_train=arr[2]
      Y_test=arr[3]
      print(X.shape , X_train.shape , X_test.shape)
      print(Y.shape,Y_train.shape , Y_test.shape)

      split_data(X,Y)

      #Training
      logistic_regression = LogisticRegression(max_iter=10000)
      logistic_regression.fit(X_train,Y_train)

      Y_pred_train=logistic_regression.predict(X_train)
      Y_pred_test=logistic_regression.predict(X_test)

      print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
      print()

      print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
      print()

      print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
      print()

      print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16)
(13611, 12)
(13611, 12) (10888, 12) (2723, 12)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  92.06757253029747

Precision for Test Set:  93.75827152640188

F1 for Test Set:  93.28931055853147

Recall for Test Set:  [ 86.45833333 100.          95.45454545  92.99270073  94.21052632
  94.17475728  87.31617647]
```

COMPARE LOGISTIC AND NAIVE BAYES

```
#Splitting the data
X= df.drop(columns='Class')
Y=df['Class']
def split_data(X,Y):
  X_train, X_test,Y_train,Y_test= train_test_split( X, Y, test_size=0.2, random_state=2)
  data_splitted= [X_train, X_test,Y_train,Y_test]
  return data_splitted
arr=split_data(X,Y)
X_train=arr[0]
X_test=arr[1]
Y_train=arr[2]
Y_test=arr[3]
print(X.shape , X_train.shape , X_test.shape)
print(Y.shape,Y_train.shape , Y_test.shape)

#Appling Logistic Regression
logistic_regression = LogisticRegression(max_iter=10000)
logistic_regression.fit(X_train,Y_train)

Y_pred_train=logistic_regression.predict(X_train)
Y_pred_test=logistic_regression.predict(X_test)

print("Accuracy for Test Set: ", metrics.accuracy_score(Y_test, Y_pred_test)*100)
print()

print("Precision for Test Set: ", metrics.precision_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("F1 for Test Set: ", metrics.f1_score(Y_test, Y_pred_test, average='macro')*100)
print()

print("Recall for Test Set: ", metrics.recall_score(Y_test, Y_pred_test, average=None)*100)
```

```
(13611, 16) (10888, 16) (2723, 16)
(13611,) (10888,) (2723,)
Accuracy for Test Set:  91.73705471905987

Precision for Test Set:  93.39522399852765

F1 for Test Set:  92.99588879260997

Recall for Test Set:  [ 86.11111111 100.          95.45454545  93.13868613  95.26315789
  93.68932039  85.29411765]
```

Thank You