PIMA INDIAN DIABETES DATA ANALYSIS

Exploratory Data Analysis

In [142…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
#from pandas_profiling import ProfileReport
%matplotlib inline
sns.set(style="darkgrid")
```

In [143…
```
df = pd.read_csv("D:\perfectplan_practice_problems\machine_learning\pima-indians-diabetes.csv")
df.head(5)
df.shape
```

Out[143… (767, 9)

In [144…
```
df.describe()

# comparison of mean & median (50%)
# look into min values to find out presence of zeroes
# comparison of various percentiles with max value to find outliers
```

Out[144…

	Pregnancy_count	Plasma_glucose_conc	BP	Triceps_thickness	Serum_insulin	BMI	Pedigree_function	Age	Class_variable
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000
mean	3.842243	120.859192	69.101695	20.517601	79.903520	31.990482	0.471674	33.219035	0.348110
std	3.370877	31.978468	19.368155	15.954059	115.283105	7.889091	0.331497	11.752296	0.476682
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243500	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000	0.371000	29.000000	0.000000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.000000	1.000000

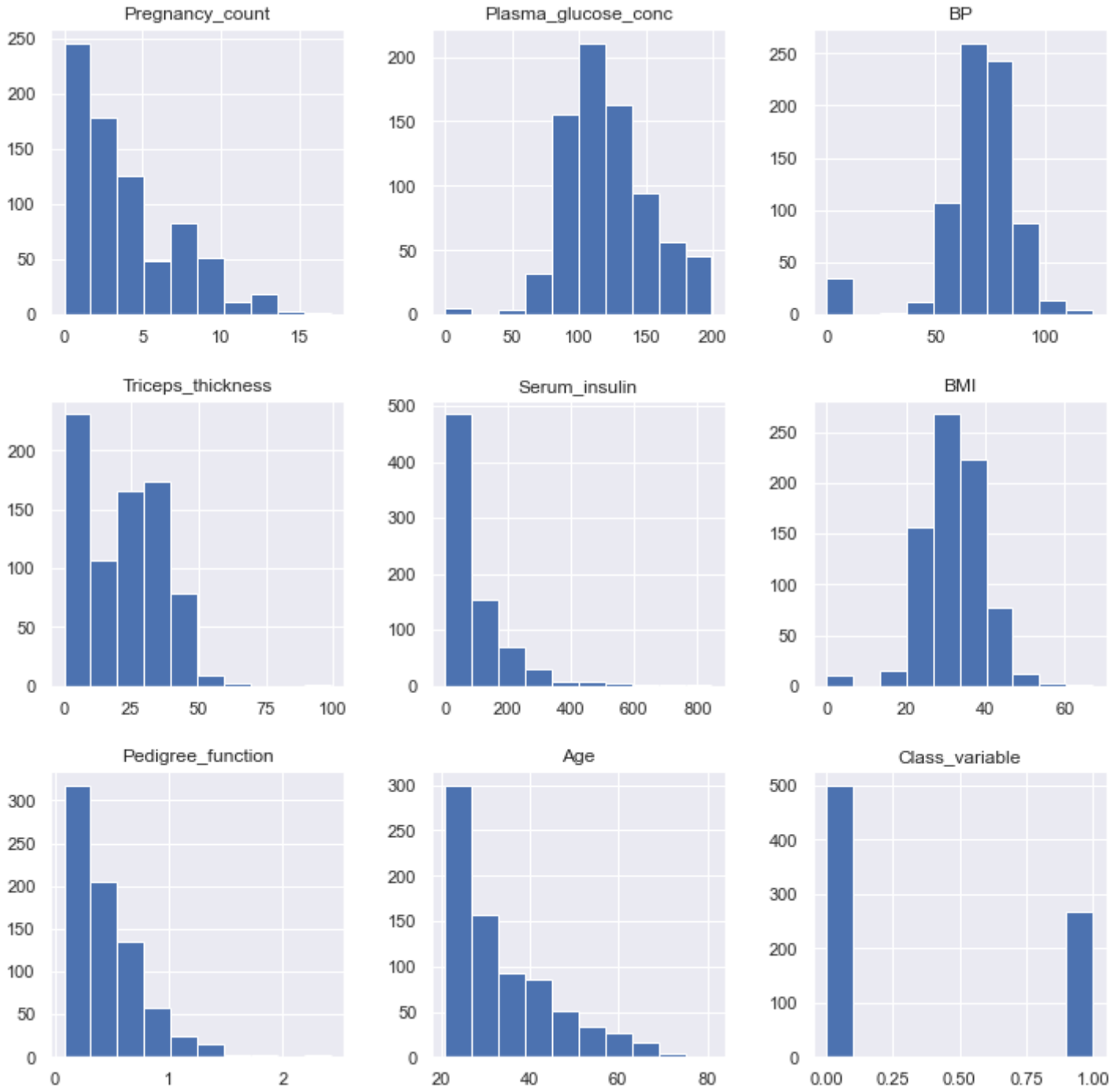
	Pregnancy_count	Plasma_glucose_conc	BP	Triceps_thickness	Serum_insulin	BMI	Pedigree_function	Age	Class_variable
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

NORMAL DISTRIBUTION

Check for normal Distribution using HISTOGRAMS

In [145…

```
#plot a histogram to check for normal deviation

df.hist(bins=10,figsize=(12,12))
plt.show()

#using histogram find normal distribution possible or not (if skewed right then not normal distribution found !! )
```


Pregnancy_count
Plasma_glucose_conc
BP
Triceps_thickness
Serum_insulin
BMI
Pedigree_function
Age
Class_variable

Check For normal distribution Using SHAPIRO WILK TEST

In [146…
```
# find normal distribution or not using SHAPIRO WILK TEST OF NULL HYPOTHESIS
from scipy.stats import shapiro

#glucose concentration
stat1,p1 = shapiro(df["Plasma_glucose_conc"])
print("Statistics={:.3f} P value={:.3f}".format(stat1,p1))

#BP
stat2,p2 = shapiro(df["BP"])
print("Statistics={:.3f} P value={:.3f}".format(stat2,p2))

#BMI
stat3,p3 = shapiro(df["BMI"])
print("Statistics={:.3f} P value={:.3f}".format(stat3,p3))

stat4,p4 = shapiro(df["Triceps_thickness"])
print("Statistics={:.3f} P value={:.3f}".format(stat4,p4))


# p<0.005, hence at 5% Level of Significance , we reject null hypothesis & proved variable doesn't follows normal distribution
```

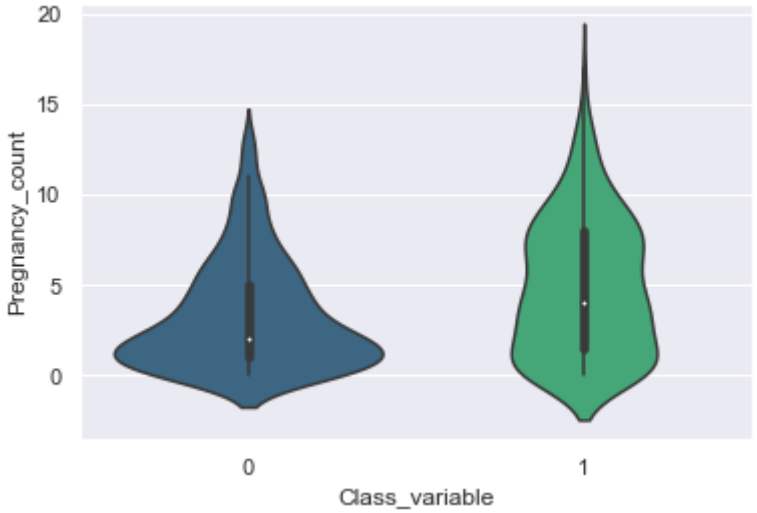
```
Statistics=0.970 P value=0.000
Statistics=0.819 P value=0.000
Statistics=0.950 P value=0.000
Statistics=0.905 P value=0.000
```

CHECKING & HANDLING MISSING VALUES

Check for 0's using Violinplot

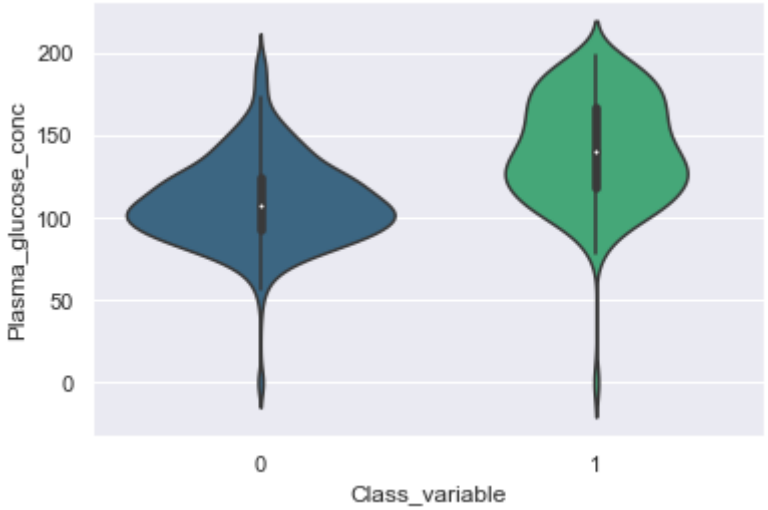
In [147…
```
#Check for zeros in Pregnancy count (a women can have 0 pregnancy count )
sns.violinplot(y="Pregnancy_count",x="Class_variable",palette="viridis",split=True,data=df)
```

Out[147…
```
<AxesSubplot:xlabel='Class_variable', ylabel='Pregnancy_count'>
```



In [148…

```
#Check for zeros in Plasma glucose concentration (a women can never have 0 glucose level )

sns.violinplot(y="Plasma_glucose_conc",x="Class_variable",palette="viridis",split=True,data=df)
plt.show()


#OBSERVATION 1
# box plot of class_variable 1 is little away from the horizontal line than the box plot of class_variable 0
# CONCLUSION --> Diabetic person have higher glucose in the plasma

#OBSERVATION 2
# Bottom tail of the violin is slightly broader
# CONCLUSION --> Zeroes must be replace in the column
```



Imputation - using df.replace(old_value,median_value)

In [149…

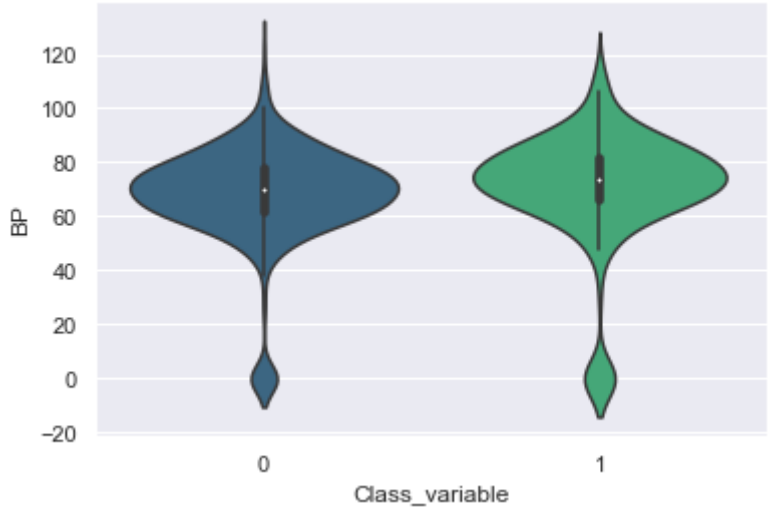
```
# Handle 0's present in plasma glucose column

df1 = df.loc[df["Class_variable"]==1]

df2 = df.loc[df["Class_variable"]==0]

#replace 0 with median value in the plasma glucose column

df1 = df1.replace({"Plasma_glucose_conc":0},np.median(df1["Plasma_glucose_conc"]))
df2 = df2.replace({"Plasma_glucose_conc":0},np.median(df2["Plasma_glucose_conc"]))

df3 = [df1,df2]
df = pd.concat(df3)
print(df)

sns.violinplot(y="Plasma_glucose_conc",x="Class_variable",palette="viridis",split=True,data=df)
plt.show()
```

```
     Pregnancy_count  Plasma_glucose_conc  BP  Triceps_thickness  \
1                  8                  183  64                  0
3                  0                  137  40                 35
5                  3                   78  50                 32
7                  2                  197  70                 45
8                  8                  125  96                  0
```

```
..               ...                   ...  ..                ...
761                9                    89  62                  0
762               10                   101  76                 48
763                2                   122  70                 27
764                5                   121  72                 23
766                1                    93  70                 31

     Serum_insulin   BMI  Pedigree_function  Age  Class_variable
1                0  23.3              0.672   32               1
3              168  43.1              2.288   33               1
5               88  31.0              0.248   26               1
7              543  30.5              0.158   53               1
8                0   0.0              0.232   54               1
..             ...   ...                ...  ...             ...
761              0  22.5              0.142   33               0
762            180  32.9              0.171   63               0
763              0  36.8              0.340   27               0
764            112  26.2              0.245   30               0
766              0  30.4              0.315   23               0

[767 rows x 9 columns]
```

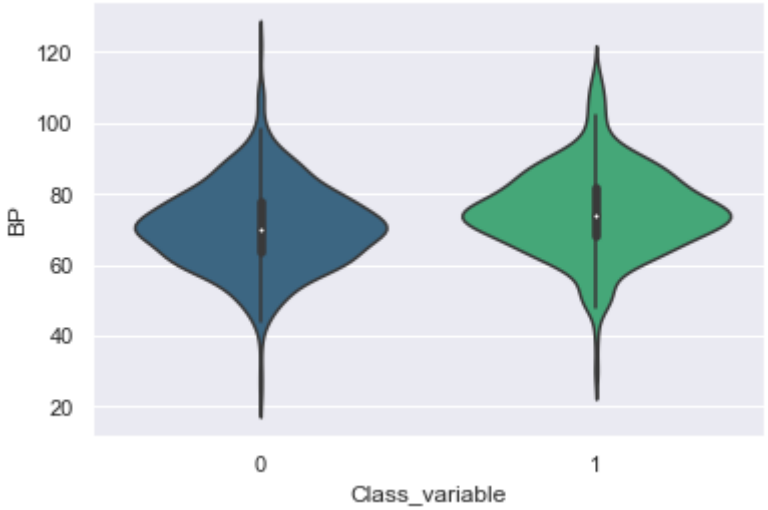
In [150…

```
sns.violinplot(x="Class_variable",y="BP",split=True,palette="viridis",data=df)
plt.show()
```



In [151…

```
#Replace 0 with median of Bp
df1 = df.loc[df["Class_variable"] == 0]
df2 = df.loc[df["Class_variable"] == 1]

df1 = df1.replace({"BP":0},np.median(df1["BP"]))
df2 = df2.replace({"BP":0},np.median(df2["BP"]))

df = pd.concat([df1,df2])

sns.violinplot(x="Class_variable",y="BP",split=True,palette="viridis",data=df)
plt.show()
```



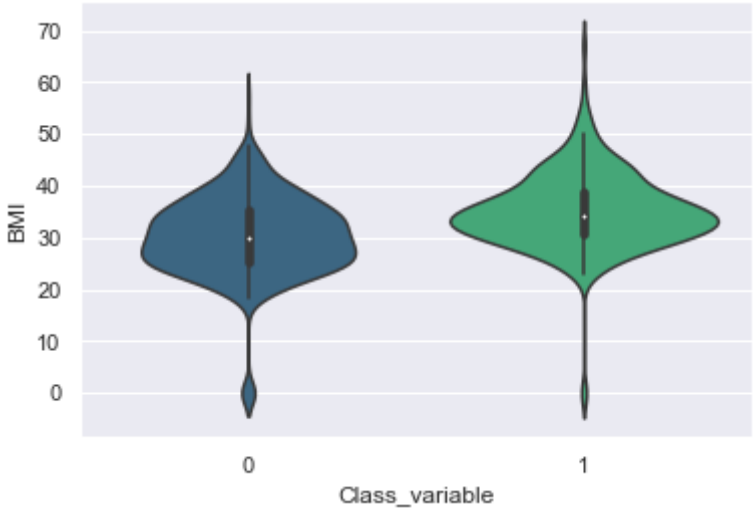
In [152…
```
df.describe()
```

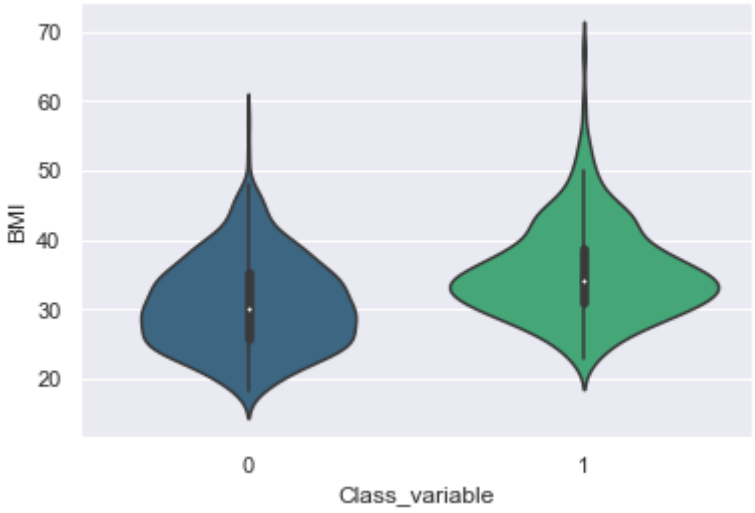
Out[152…

	Pregnancy_count	Plasma_glucose_conc	BP	Triceps_thickness	Serum_insulin	BMI	Pedigree_function	Age	Class_variable
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000
mean	3.842243	121.642764	72.379400	20.517601	79.903520	31.990482	0.471674	33.219035	0.348110
std	3.370877	30.469180	12.112322	15.954059	115.283105	7.889091	0.331497	11.752296	0.476682
min	0.000000	44.000000	24.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.500000	64.000000	0.000000	0.000000	27.300000	0.243500	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000	0.371000	29.000000	0.000000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [153…
```
sns.violinplot(x="Class_variable",y="BMI",split=True,palette="viridis",data=df)
plt.show()

#Replace 0 with median of Bmi
df1 = df.loc[df["Class_variable"] == 0]
df2 = df.loc[df["Class_variable"] == 1]
```

```
df1 = df1.replace({"BMI":0},np.median(df1["BMI"]))
df2 = df2.replace({"BMI":0},np.median(df2["BMI"]))

df = pd.concat([df1,df2])

sns.violinplot(x="Class_variable",y="BMI",split=True,palette="viridis",data=df)
plt.show()
```

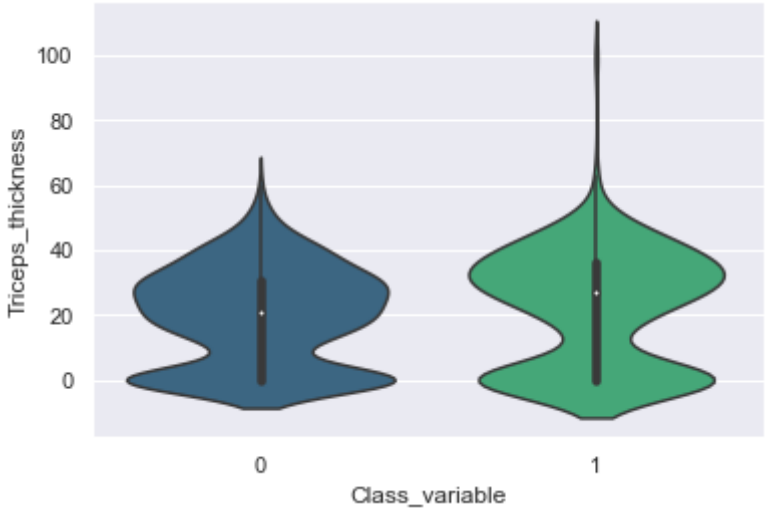


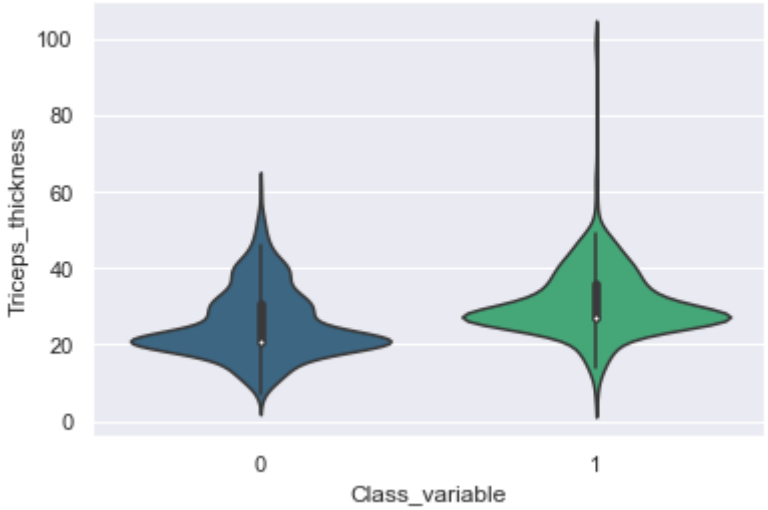


In [154…

```
sns.violinplot(x="Class_variable",y="Triceps_thickness",split=True,palette="viridis",data=df)
plt.show()
```

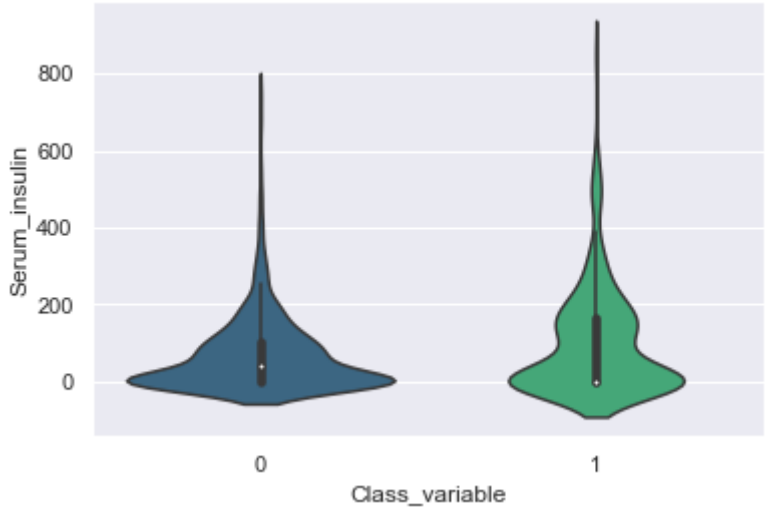
```
#Replace 0 with median of Bp
df1 = df.loc[df["Class_variable"] == 0]
df2 = df.loc[df["Class_variable"] == 1]
#print(df1.describe())
#print(df2.describe())
df1 = df1.replace({"Triceps_thickness":0},np.median(df1["Triceps_thickness"]))
df2 = df2.replace({"Triceps_thickness":0},np.median(df2["Triceps_thickness"]))

df = pd.concat([df1,df2])

sns.violinplot(x="Class_variable",y="Triceps_thickness",split=True,palette="viridis",data=df)
plt.show()
```

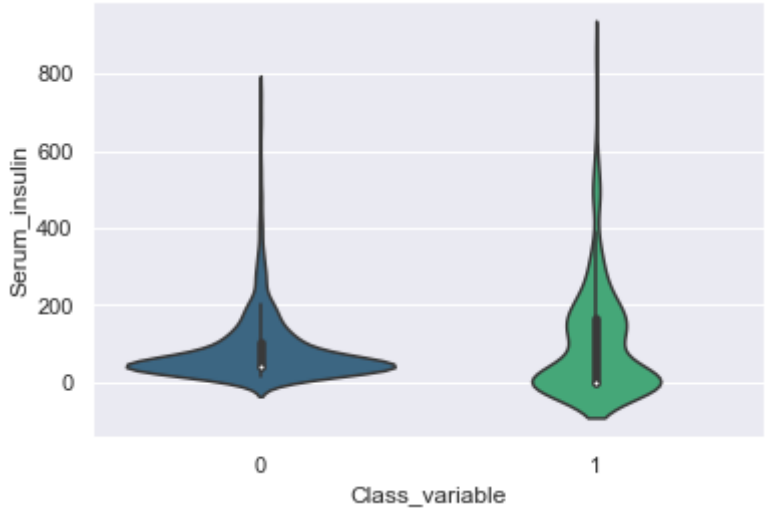




In [155…

```
sns.violinplot(x="Class_variable",y="Serum_insulin",split=True,palette="viridis",data=df)
plt.show()

#Replace 0 with median of Bp
df1 = df.loc[df["Class_variable"] == 0]
df2 = df.loc[df["Class_variable"] == 1]
#print(df1.describe())
#print(df2.describe())
df1 = df1.replace({"Serum_insulin":0},np.median(df1["Serum_insulin"]))
df2 = df2.replace({"Serum_insulin":0},np.median(df2["Serum_insulin"]))


df = pd.concat([df1,df2])

sns.violinplot(x="Class_variable",y="Serum_insulin",split=True,palette="viridis",data=df)
plt.show()
```





In [156...
```
df.describe()
```

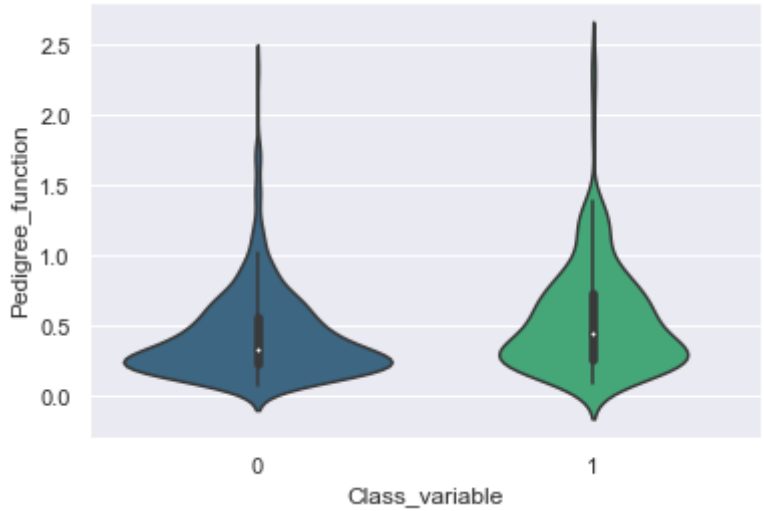
Out[156...

	Pregnancy_count	Plasma_glucose_conc	BP	Triceps_thickness	Serum_insulin	BMI	Pedigree_function	Age	Class_variable
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000
mean	3.842243	121.642764	72.379400	27.421121	91.903520	32.432529	0.471674	33.219035	0.348110
std	3.370877	30.469180	12.112322	9.323528	108.140785	6.885060	0.331497	11.752296	0.476682

	Pregnancy_count	Plasma_glucose_conc	BP	Triceps_thickness	Serum_insulin	BMI	Pedigree_function	Age	Class_variable
min	0.000000	44.000000	24.000000	7.000000	0.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.500000	64.000000	21.000000	39.000000	27.500000	0.243500	24.000000	0.000000
50%	3.000000	117.000000	72.000000	27.000000	39.000000	32.000000	0.371000	29.000000	0.000000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [157…

```
# no 0's found in pedigree function but diabetic shows higher pedigree function

sns.violinplot(x="Class_variable",y="Pedigree_function",data=df,palette="viridis",split=True)
plt.show()
```



CORRELATION BETWEEN VARIABLES

Find correlationship between cells using corr()

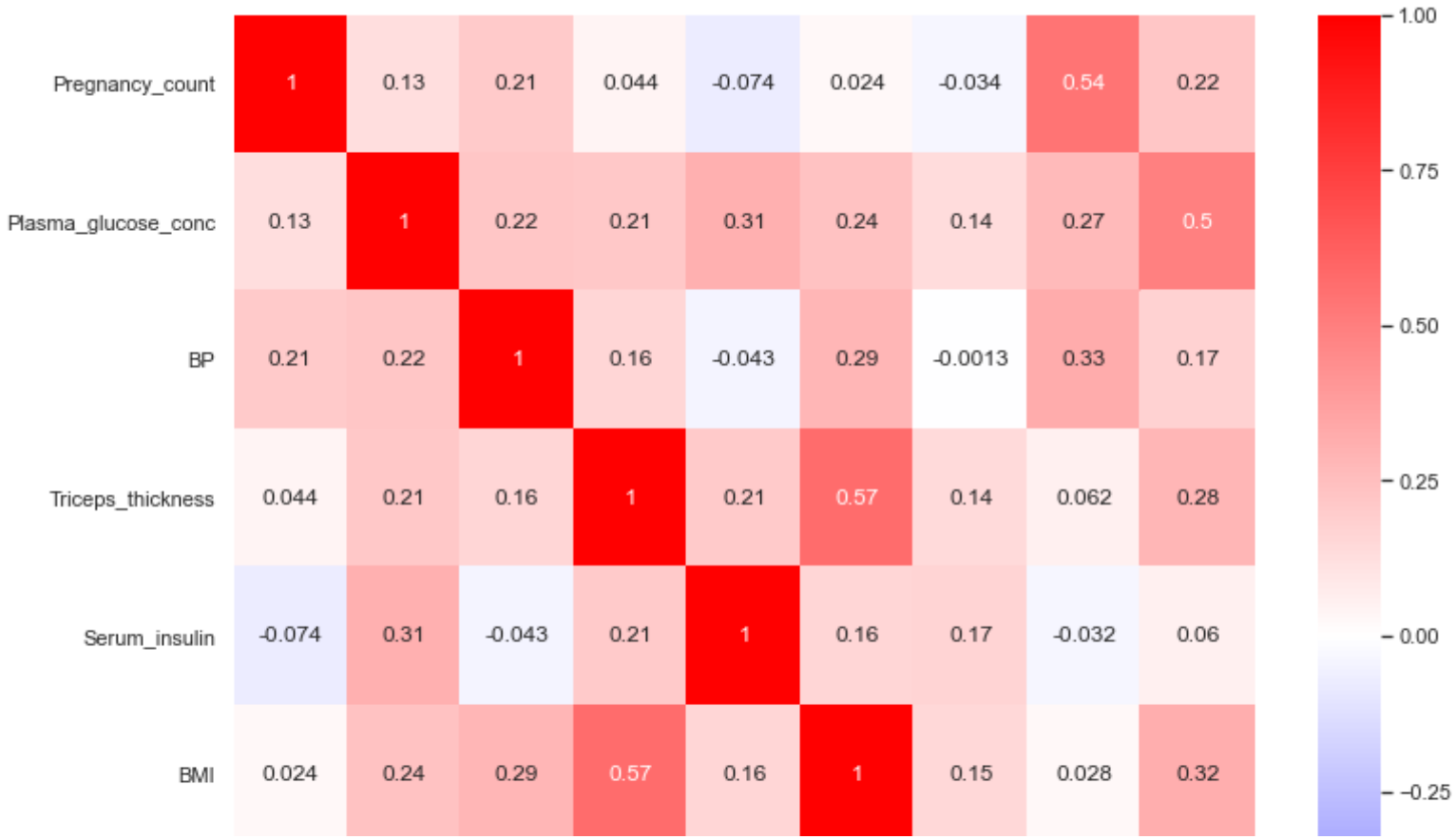
In [158…

```
df10=df.corr()
df10["Age"]
```

Out[158…
```
Pregnancy_count         0.544018
Plasma_glucose_conc     0.267788
BP                      0.325796
Triceps_thickness       0.062446
Serum_insulin          -0.032176
BMI                     0.027609
Pedigree_function       0.032738
Age                     1.000000
Class_variable          0.236417
Name: Age, dtype: float64
```

Plot Heatmap using corr() dataframe

In [159…
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),cmap="bwr",vmax=1,vmin=-1,annot=True)
plt.show()
```



In [160…

```
#Co-efficient of correlation (r) if r>0.70 then multi collinearity exists

df10["Class_variable"]
```

Out[160…

```
Pregnancy_count         0.221087
Plasma_glucose_conc     0.495295
BP                      0.173583
Triceps_thickness       0.281861
Serum_insulin           0.059559
BMI                     0.315718
Pedigree_function       0.173245
Age                     0.236417
Class_variable          1.000000
Name: Class_variable, dtype: float64
```

PIMA INDIAN DIABETES DETECTION USING ML ALGORITHM

Split independent & dependent Variables

In [163…
```
y = df.Class_variable
x = df.drop("Class_variable",axis=1)
```

Preprocessing technique --> StandardScaler()

-> Import from sklearn.preprocessing
-> Used to treat outliers
-> It transforms [scaler = x-mean/std]
-> Hence non-normally distributed variable get close to std normal distribution with mean to 0

In [176…
```
col = x.columns


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(x)

data_x = pd.DataFrame(X,columns=col)
print(data_x)
```

```
     Pregnancy_count  Plasma_glucose_conc        BP  Triceps_thickness  \
0          -0.843726            -1.203402 -0.527030           0.169454
1          -0.843726            -1.072036 -0.527030          -0.474499
2           0.343683            -0.185317  0.133885          -0.689150
3           1.827945            -0.218158 -0.196573          -0.689150
4           0.046831            -0.382365  1.620945          -0.689150
..               ...                  ...       ...                ...
762        -0.843726             0.208781  1.290487           1.242709
763        -1.140579             0.044574 -0.031344          -0.045197
764         0.640535             2.244952  1.620945          -0.045197
765         1.531092             1.588123  0.133885           0.384105
766        -0.843726             0.143098 -1.022717          -0.045197

     Serum_insulin       BMI  Pedigree_function       Age
0        -0.489529 -0.847681          -0.364265 -0.188940
1         0.019399 -0.629676          -0.919684 -1.040393
2        -0.489529 -0.993018          -0.817052 -0.274086
3        -0.489529  0.416749          -1.019297 -0.359231
4        -0.489529  0.751024          -0.847238 -0.274086
..             ...       ...                ...       ...
762       0.167451  0.591153           1.766855  0.321931
763      -0.850405  0.562086          -0.644993  1.599111
764      -0.850405  0.445816          -0.584621  2.791145
765      -0.850405  1.681179          -0.207298  0.832803
```

```
766      -0.850405 -0.339002          -0.370302  1.173384

[767 rows x 8 columns]
```

Model Selection for training & testing

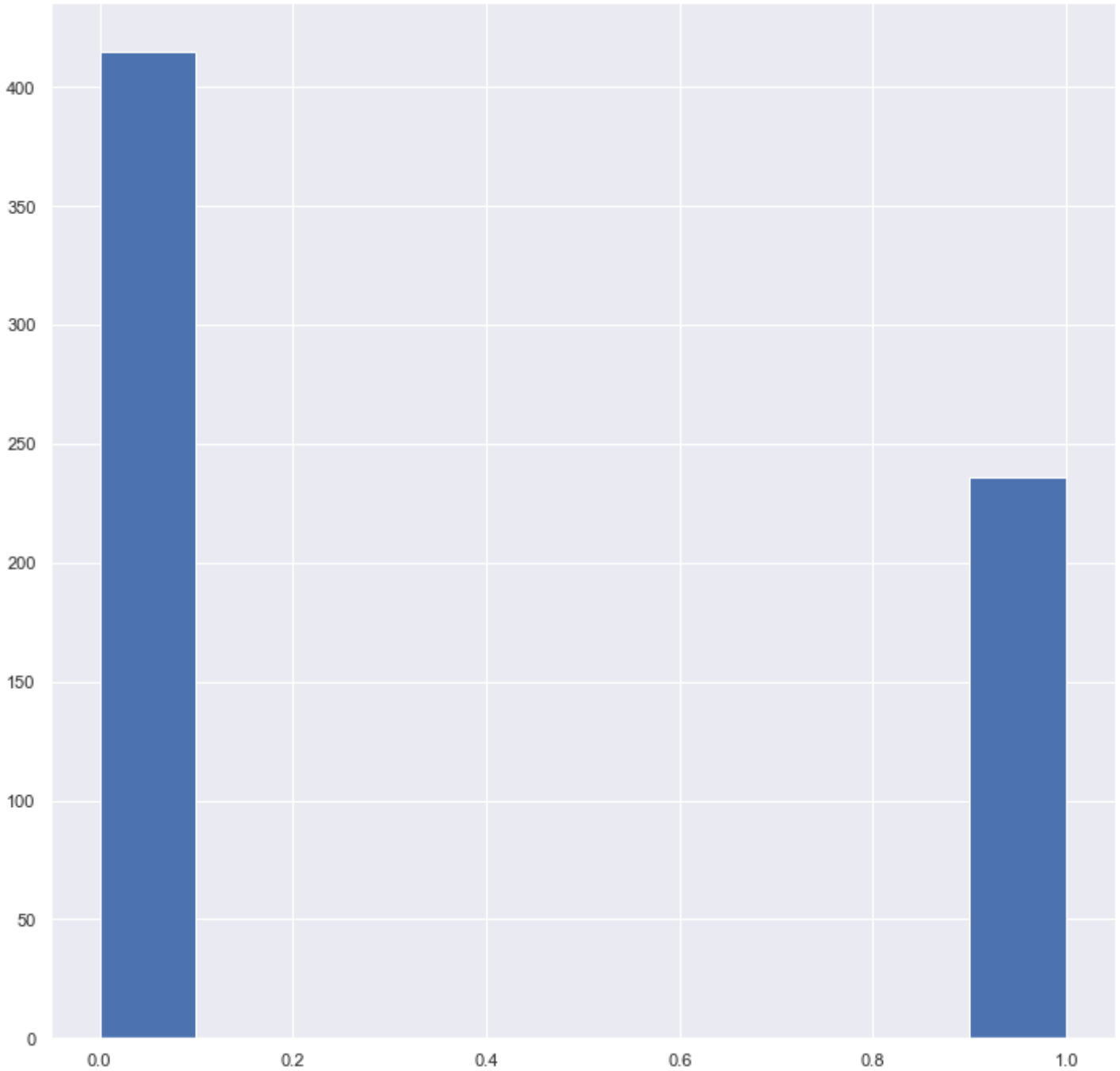
In [185…

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(data_x,y,test_size=0.15,random_state=45)
```

Check for Balance in training data of Y (0/1)

In [186…

```
y_train.hist(bins=10,figsize=(12,12))
plt.show()
```

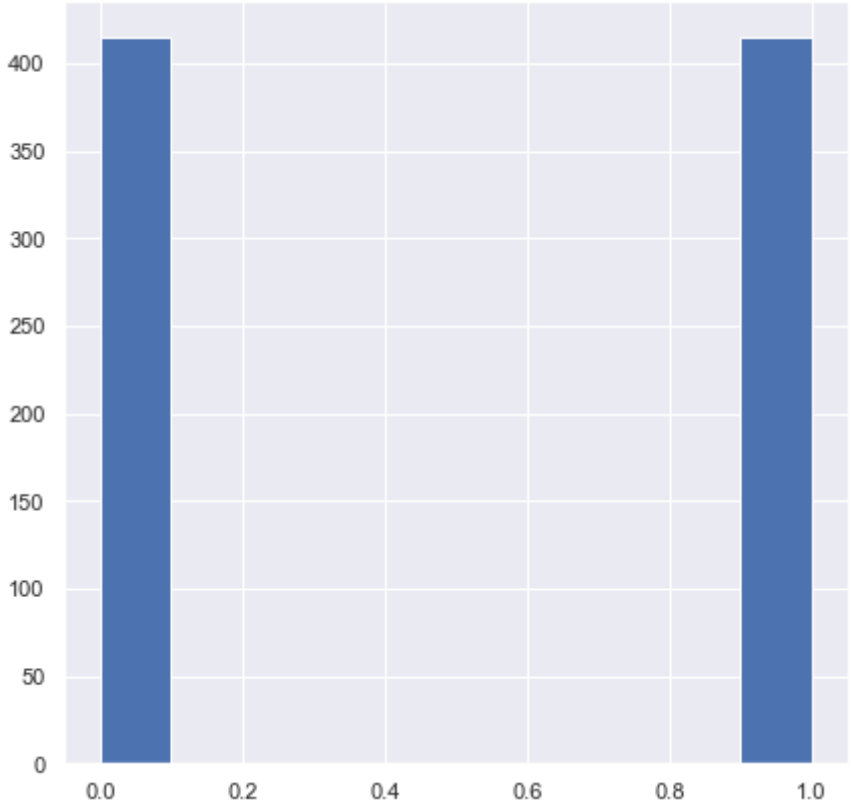

400
350
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0

To balance the imbalance in y_train data --> Use SMOTE

--> Synthetic Minority Oversampling Technique
--> Remove imbalance in the training data
--> It creates a sampling using current data
--> done only on training data since testing data should be a real life data

In [194…

```
from imblearn.over_sampling import SMOTE

smt = SMOTE()

x_train,y_train = smt.fit_sample(x_train,y_train)

# balanced training Y variable

y_train.hist(bins=10,figsize=(7,7))
plt.show()
```



MODEL FITTING

LOGISTIC REGRESSION

In [208…

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score,recall_score,precision_score

l = LogisticRegression()

l.fit(x_train,y_train)
score = l.score(x_test,y_test)
print("{:.3f}".format(score))
```

```
#using x_test predict y values
y_pred = l.predict(x_test)

print(f1_score(y_test,y_pred,average="macro"))
print(precision_score(y_test,y_pred,average="macro"))
print(recall_score(y_test,y_pred,average="macro"))
```

```
0.802
0.7586612392582542
0.7493055555555556
0.7724857685009487
```

SUPPORT VECTOR MACHINE

In [209…

```
from sklearn.svm import SVC
from sklearn.metrics import f1_score,recall_score,precision_score
l = SVC(kernel="rbf")

l.fit(x_train,y_train)
score = l.score(x_test,y_test)
print("{:.3f}".format(score))


#using x_test predict y values
y_pred = l.predict(x_test)

print(f1_score(y_test,y_pred,average="macro"))
print(precision_score(y_test,y_pred,average="macro"))
print(recall_score(y_test,y_pred,average="macro"))
```

```
0.819
0.7796472184531886
0.7694444444444444
0.7944971537001897
```

RANDOM FOREST CLASSIFIER

In [216…

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1_score,recall_score,precision_score
l = RandomForestClassifier(n_estimators=300,bootstrap=True,max_features='sqrt')

l.fit(x_train,y_train)
```

```
score = l.score(x_test,y_test)
print("{:.3f}".format(score))


#using x_test predict y values
y_pred = l.predict(x_test)

print(f1_score(y_test,y_pred,average="macro"))
print(precision_score(y_test,y_pred,average="macro"))
print(recall_score(y_test,y_pred,average="macro"))
```

```
0.905
0.8749877535024982
0.8887987012987013
0.8635673624288425
```

ACCURACY OF MODEL ON TEST DATA SET

* Logistic Regression from linear_model --> 80%

* Support Vector Machine(SVC) from svm --> 82%

* Random Forest Classifier from ensemble -> 90.5%

CONCLUSION

Since for data set or test sets with outliers , tree based models gives desired accurary and hence Random forest classifier has higher accuracy when compared to Logistic regression and Support Vector Machine models