# --Capstone Project-Healthcare--

## Problem statement:

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Build a model to accurately predict whether the patients in the dataset have diabetes or not.

## Dataset description:

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

| Variable | Description |
| --- | --- |
| Pregnancies | Number of times pregnant |
| Glucose | Plasma glucose concentration in an oral glucose tolerance test |
| BloodPressure | Diastolic blood pressure (mm Hg) |
| SkinThickness | Triceps skinfold thickness (mm) |
| Insulin | Two hour serum insulin |
| BMI | Body Mass Index |
| DiabetesPedigreeFunction | Diabetes pedigree function |
| Age | Age in years |
| Outcome | Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0 |

# Task to be performed:

# Week:1

## Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

   - Glucose
   - BloodPressure
   - SkinThickness
   - Insulin
   - BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.

3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

   a. Perform preliminary data inspection and report the findings on the structure of the data, missing values, duplicates, etc.

   b. Based on these findings, remove duplicates (if any) and treat missing values using an appropriate strategy.

# Week:2

## Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

3. Perform correlation analysis. Visually explore it using a heat map.

# Week:3

## Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

# Week:4

## Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

## Data Reporting:

2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

    a. Pie chart to describe the diabetic or non-diabetic population

```
    b. Scatter charts between relevant variables to analyze the relationships

    c. Histogram or frequency charts to analyze the distribution of the data

    d. Heatmap of correlation analysis among the relevant variables

    e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age
brackets             using a bubble chart.
```

# Solution:-

# Week 1-Data Exploration

## 1. Preliminary analysis:

### Importing appropriate libraries

In [166…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import statistics as st
```

### Load the dataset

In [167…

```
df = pd.read_csv("C:\\Users\\Lenovo\\Desktop\\DATA SCIENCE\\7)Capstone\\Project_2\\Project 2\\Healthcare - Diabetes\\health care
```

### Info of the dataset

In [168…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

# Shape of the dataset

In [169…

```
df.shape
```

Out[169]:

```
(768, 9)
```

# Head the dataset

In [170…

```
df.head()
```

Out[170]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

## Tail of the dataset

In [171…
```
df.tail()
```

Out[171]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

## Checking the null values of the dataset

In [172…
```
df.isna().sum()
```

Out[172]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

## Checking the duplicates of the dataset

In [173…
```
df.duplicated().sum()
```

Out[173]:
```
0
```

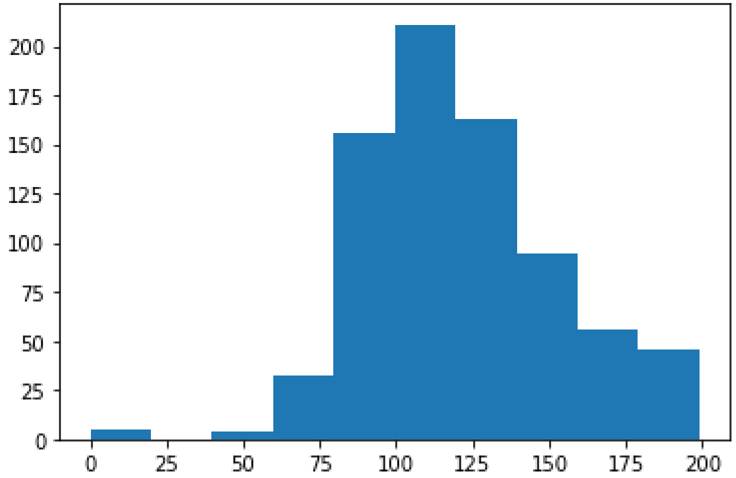
# 2. Visually explore these variables using histograms:

In [174…

```
plt.hist(df['Glucose'])
```

Out[174]:

```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```

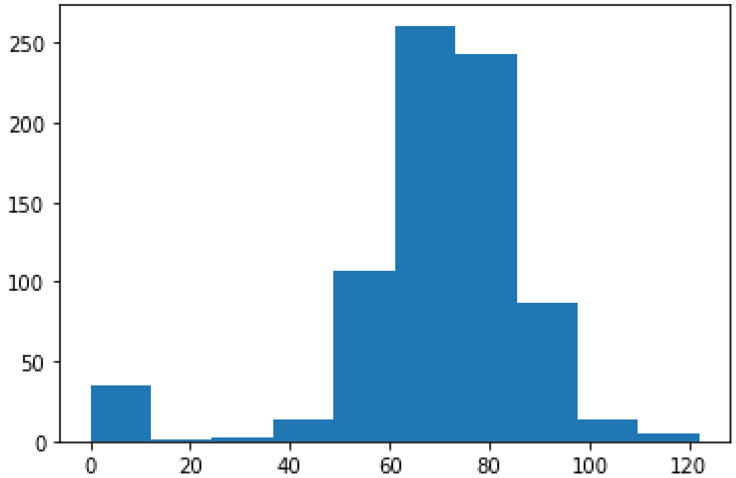



In [175…

```
plt.hist(df['BloodPressure'])
```

Out[175]:

```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <BarContainer object of 10 artists>)
```

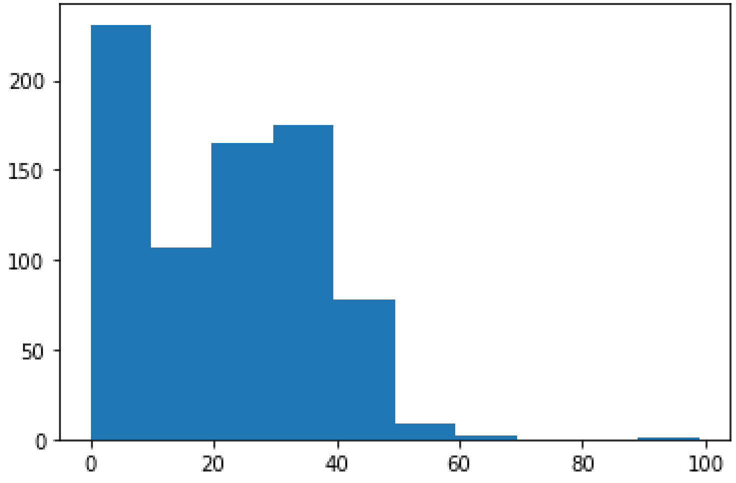


In [176…

```
plt.hist(df['SkinThickness'])
```

Out[176]:

```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```

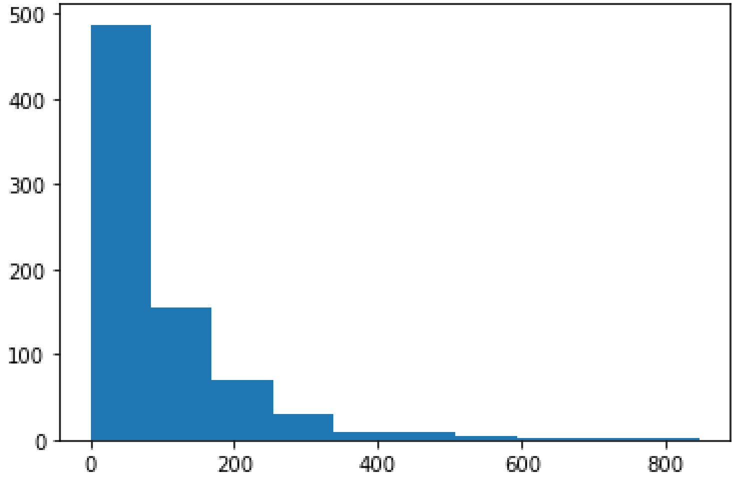


In [177…

```
plt.hist(df['Insulin'])
```

Out[177]:

```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```

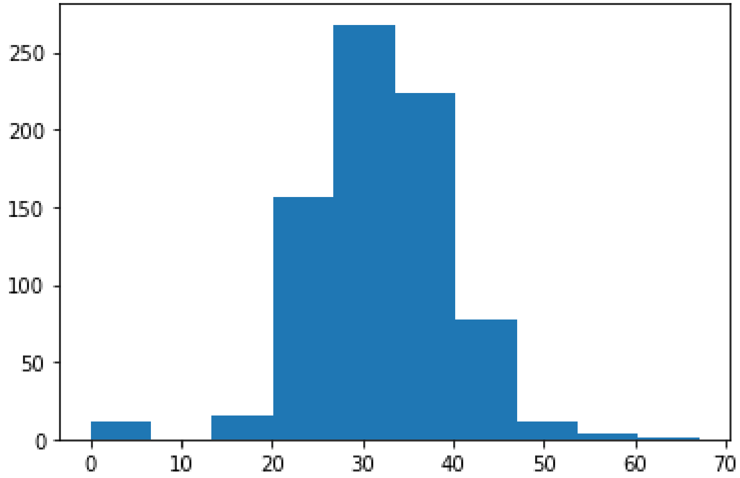


In [178…

```
plt.hist(df['BMI'])
```

Out[178]:

```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```




# 3. Create a count (frequency) plot describing the data types and the count of v

In [179…

```
df['Glucose'].value_counts().head()
```

Out[179]:

```
99     17
100    17
111    14
129    14
125    14
Name: Glucose, dtype: int64
```

In [180…

```
df['BloodPressure'].value_counts().head()
```

Out[180]:
```
70    57
74    52
78    45
68    45
72    44
Name: BloodPressure, dtype: int64
```

In [181…
```
df['SkinThickness'].value_counts().head()
```

Out[181]:
```
0     227
32     31
30     27
27     23
23     22
Name: SkinThickness, dtype: int64
```

In [182…
```
df['Insulin'].value_counts().head()
```

Out[182]:
```
0      374
105     11
130      9
140      9
120      8
Name: Insulin, dtype: int64
```

# On the columns below, a value of zero does not make sense and thus indicates ı

## Missing value treatment:

In [183…
```
df['BMI'].value_counts().head()
```

Out[183]:
```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
Name: BMI, dtype: int64
```

In [184…
```
print(df['Glucose'].mean())
print(df['BloodPressure'].mean())
```

```
print(df['SkinThickness'].mean())
print(df['Insulin'].mean())
print(df['BMI'].mean())
```

```
120.89453125
69.10546875
20.536458333333332
79.79947916666667
31.992578124999977
```

In [185…

```
df['Glucose'] = df['Glucose'].replace(0, 120)
df['BloodPressure'] = df['BloodPressure'].replace(0, 69)
df['SkinThickness'] = df['SkinThickness'].replace(0, 20)
df['Insulin'] = df['Insulin'].replace(0, 79)
df['BMI'] = df['BMI'].replace(0, 31)
```

# Week 2-Data Exploration

## 1. Check the balance of the data by plotting the count of outcomes by their val

In [186…

```
df['Glucose'].value_counts().head()
```

Out[186]:

```
100    17
99     17
120    16
129    14
125    14
Name: Glucose, dtype: int64
```

In [187…

```
df['BloodPressure'].value_counts().head()
```

Out[187]:

```
70    57
74    52
78    45
68    45
72    44
Name: BloodPressure, dtype: int64
```

In [188…
```
df['SkinThickness'].value_counts().head()
```

Out[188]:
```
20    240
32     31
30     27
27     23
23     22
Name: SkinThickness, dtype: int64
```

In [189…
```
df['Insulin'].value_counts().head()
```

Out[189]:
```
79     376
105     11
130      9
140      9
120      8
Name: Insulin, dtype: int64
```

In [190…
```
df['BMI'].value_counts().head()
```

Out[190]:
```
32.0    13
31.0    13
31.2    12
31.6    12
32.4    10
Name: BMI, dtype: int64
```

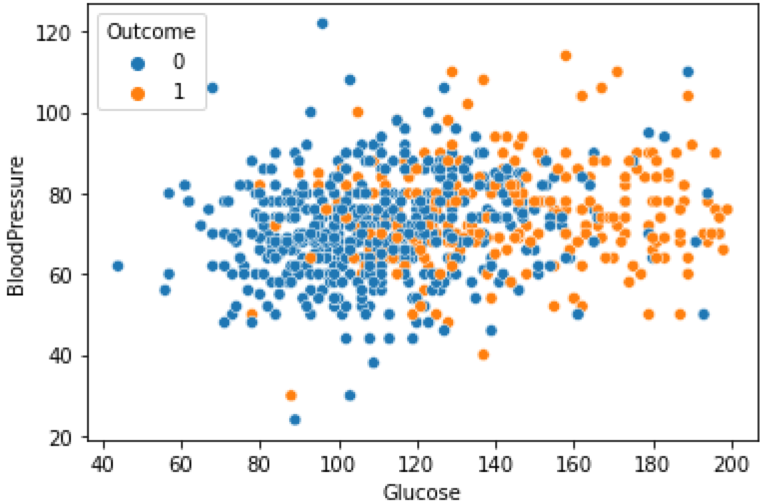
In [191…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

## 2. Create scatter charts between the pair of variables to understand the relatio

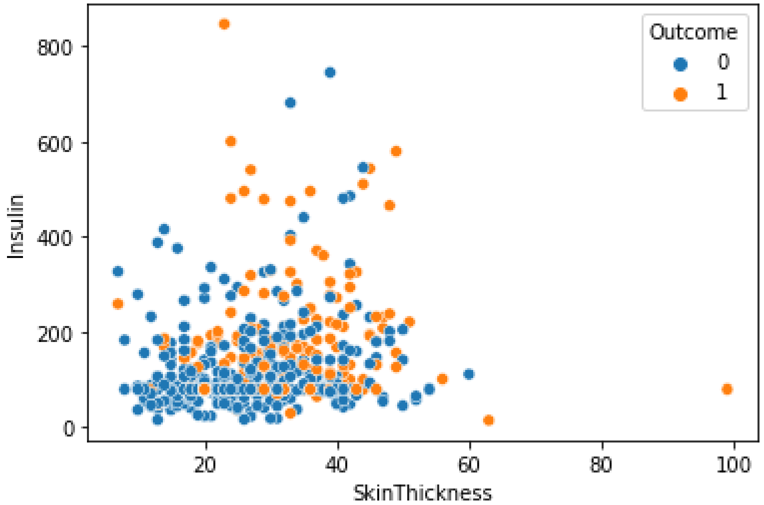
In [192…

```
g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",
              hue="Outcome",
              data=df);
```



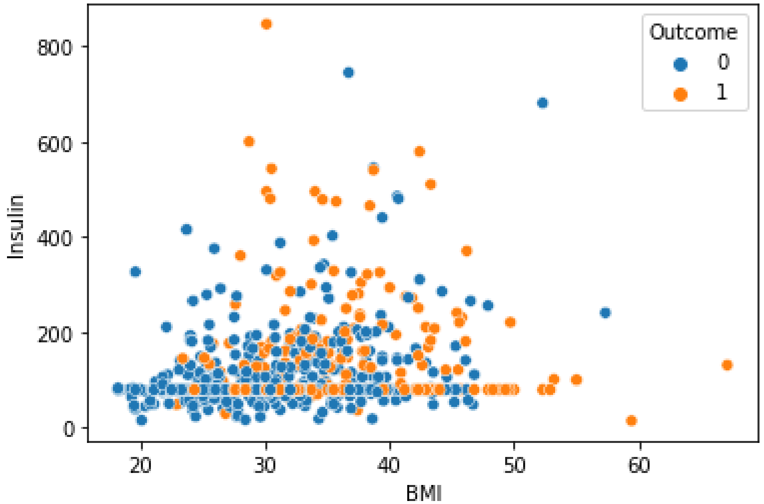

In [193…

```
g =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",
              hue="Outcome",
              data=df);
```




In [194…

```
g =sns.scatterplot(x= "BMI" ,y= "Insulin",
              hue="Outcome",
              data=df);
```

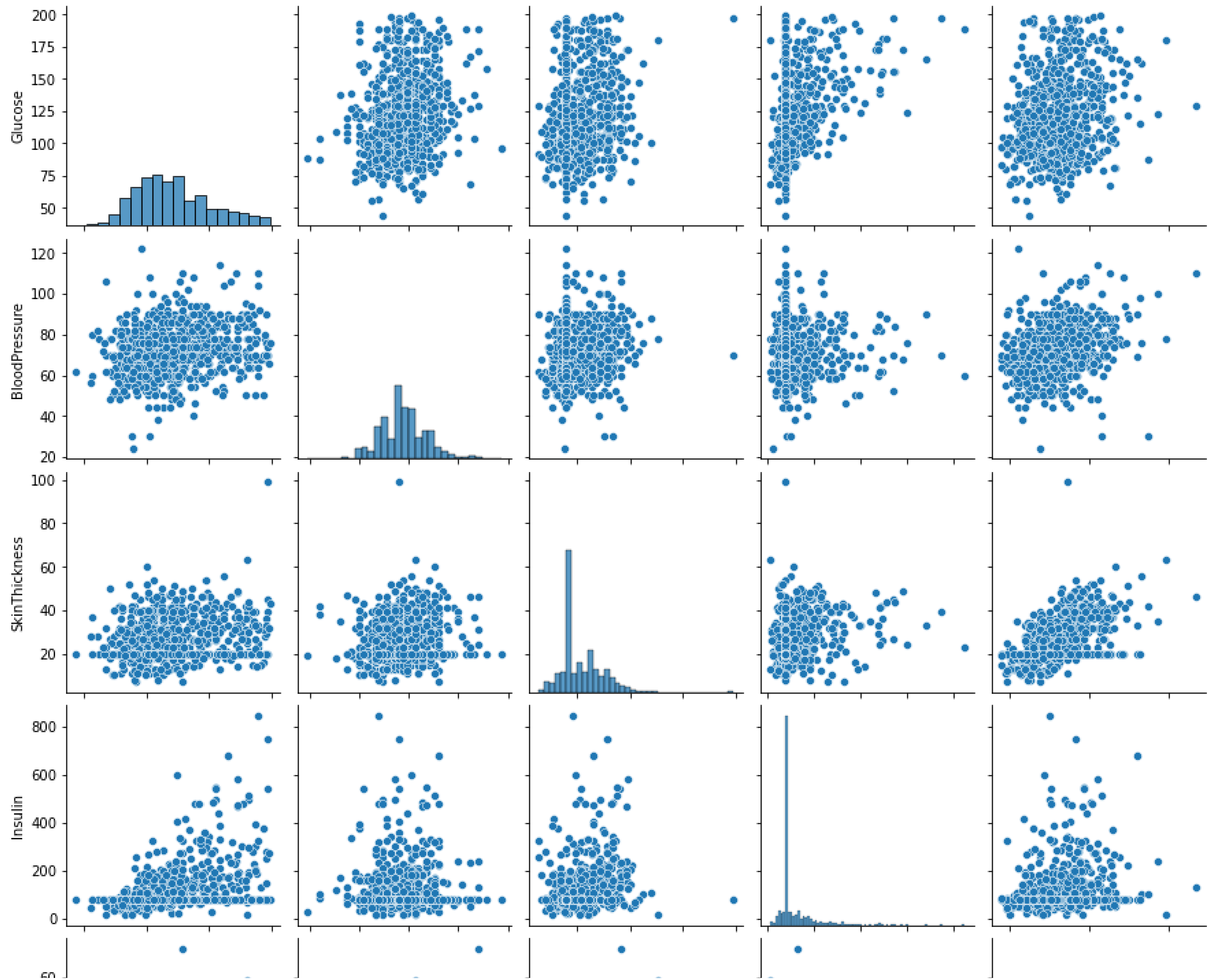



In [195…

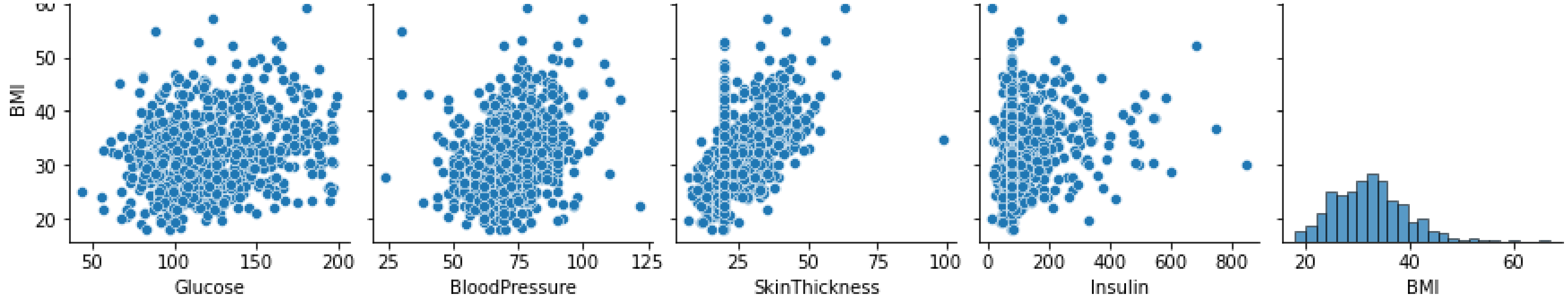
```
cols = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
sns.pairplot(df[cols])
```

Out[195]:

```
<seaborn.axisgrid.PairGrid at 0x14cd409dbe0>
```



Glucose
BloodPressure
SkinThickness
Insulin




# 3. Perform correlation analysis. Visually explore it using a heat map:

In [196…

```
df.corr()
```

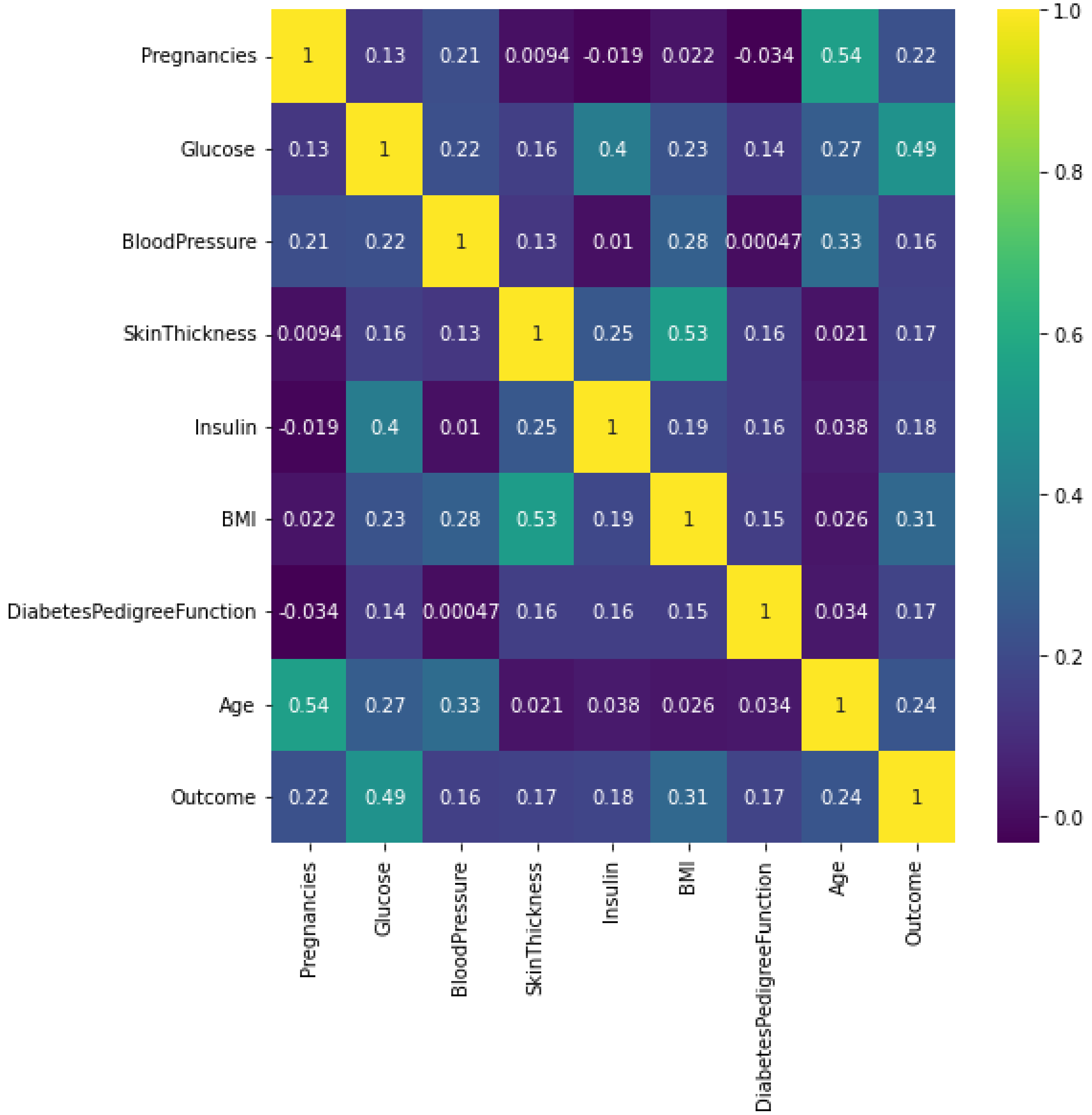
Out[196]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **Pregnancies** | 1.000000 | 0.128022 | 0.208987 | 0.009393 | -0.018780 | 0.021500 | -0.033523 | 0.544341 | 0.221898 |
| **Glucose** | 0.128022 | 1.000000 | 0.219765 | 0.158060 | 0.396137 | 0.232581 | 0.137158 | 0.266673 | 0.492884 |
| **BloodPressure** | 0.208987 | 0.219765 | 1.000000 | 0.130403 | 0.010492 | 0.281060 | 0.000471 | 0.326791 | 0.162879 |
| **SkinThickness** | 0.009393 | 0.158060 | 0.130403 | 1.000000 | 0.245410 | 0.533655 | 0.157196 | 0.020582 | 0.171857 |
| **Insulin** | -0.018780 | 0.396137 | 0.010492 | 0.245410 | 1.000000 | 0.190717 | 0.158243 | 0.037676 | 0.178696 |
| **BMI** | 0.021500 | 0.232581 | 0.281060 | 0.533655 | 0.190717 | 1.000000 | 0.153705 | 0.026231 | 0.312890 |
| **DiabetesPedigreeFunction** | -0.033523 | 0.137158 | 0.000471 | 0.157196 | 0.158243 | 0.153705 | 1.000000 | 0.033561 | 0.173844 |
| **Age** | 0.544341 | 0.266673 | 0.326791 | 0.020582 | 0.037676 | 0.026231 | 0.033561 | 1.000000 | 0.238356 |
| **Outcome** | 0.221898 | 0.492884 | 0.162879 | 0.171857 | 0.178696 | 0.312890 | 0.173844 | 0.238356 | 1.000000 |

In [197…

```
plt.subplots(figsize=(8,8))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

Out[197]: <AxesSubplot:>




# Week 3-Data Modeling

# 1. Devise strategies for model building:

## Splitting the features and target:

In [198…

```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [199…

```
x
```

Out[199]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 79 | 33.6 | 0.627 | 50 |
| **1** | 1 | 85 | 66 | 29 | 79 | 26.6 | 0.351 | 31 |
| **2** | 8 | 183 | 64 | 20 | 79 | 23.3 | 0.672 | 32 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 |
| **764** | 2 | 122 | 70 | 27 | 79 | 36.8 | 0.340 | 27 |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 |
| **766** | 1 | 126 | 60 | 20 | 79 | 30.1 | 0.349 | 47 |
| **767** | 1 | 93 | 70 | 31 | 79 | 30.4 | 0.315 | 23 |

768 rows × 8 columns

In [200…

```
y
```

Out[200]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

## Training and Testing the datasets:

In [201…
```
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)
```

In [202…
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import  KNeighborsClassifier
```

## Models

In [203…
```
model1 = LogisticRegression()
model2 = RandomForestClassifier(n_estimators=11)
model3 = DecisionTreeClassifier()
model4 = SVC(kernel='rbf', gamma='auto')
model5 = KNeighborsClassifier(n_neighbors=7, metric='minkowski',p=2)
```

In [204…
```
print(model1.fit(x_train,y_train))
print(model2.fit(x_train,y_train))
print(model3.fit(x_train,y_train))
print(model4.fit(x_train,y_train))
print(model5.fit(x_train,y_train))
```

```
LogisticRegression()
RandomForestClassifier(n_estimators=11)
DecisionTreeClassifier()
SVC(gamma='auto')
KNeighborsClassifier(n_neighbors=7)
```

# Predicting the model:

In [205…

```
y_pred1_LR = model1.predict(x_test)
y_pred2_RF= model2.predict(x_test)
y_pred3_DT = model3.predict(x_test)
y_pred4_SVM= model4.predict(x_test)
y_pred5_KNN= model5.predict(x_test)
```

In [206…

```
print('LogisticRegression=',y_pred1_LR)
print('-'*125)
print('RandomForest=',y_pred2_RF)
print('-'*125)
print('DecisionTree=',y_pred3_DT)
print('-'*125)
print('SVM=',y_pred4_SVC)
print('-'*125)
print('KNN=',y_pred5_KNN)
```

```
LogisticRegression= [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0]
-----------------------------------------------------------------------------------------------------------------------------
RandomForest= [1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1
 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0
 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 0]
-----------------------------------------------------------------------------------------------------------------------------
DecisionTree= [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 1
 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 1
 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 0]
-----------------------------------------------------------------------------------------------------------------------------
SVM= [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]
-----------------------------------------------------------------------------------------------------------------------------
KNN= [1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1
 0 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1
 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0
 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0
 0 0 0 0 0 0]
```

In [207…

```
y_test
```

Out[207]:

```
661    1
122    0
113    0
14     1
529    0
      ..
476    1
482    0
230    1
527    0
380    0
Name: Outcome, Length: 154, dtype: int64
```

# --Model Evaluation--

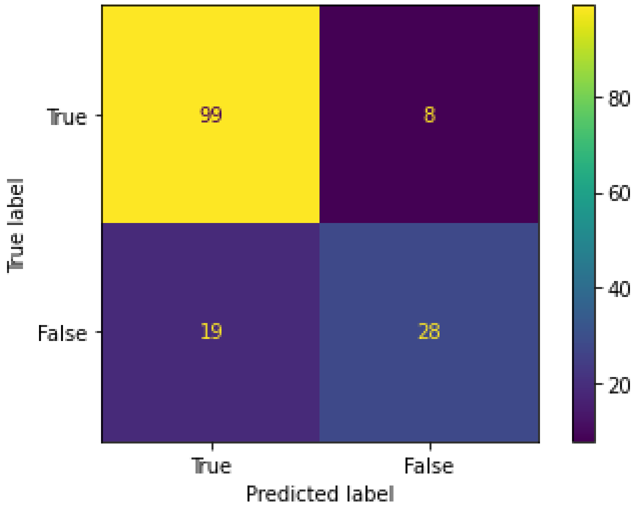
## Confusion Matrix For Logistic Regression:

In [208…

```
from sklearn import metrics
confusion_matrix_LR = metrics.confusion_matrix(y_test,y_pred1_LR)
confusion_matrix_LR = metrics.ConfusionMatrixDisplay(confusion_matrix_LR, display_labels=[True,False])
confusion_matrix_LR.plot()
```

Out[208]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14cd6662c10>
```



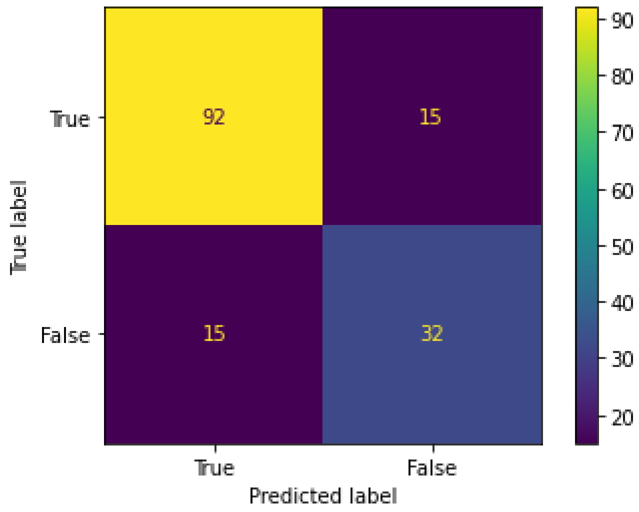

## Confusion Matrix For Random Forest:

In [209…

```
confusion_matrix_RF = metrics.confusion_matrix(y_test,y_pred2_RF)
confusion_matrix_RF = metrics.ConfusionMatrixDisplay(confusion_matrix_RF, display_labels=[True,False])
confusion_matrix_RF.plot()
```

Out[209]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14cd64ecb80>
```



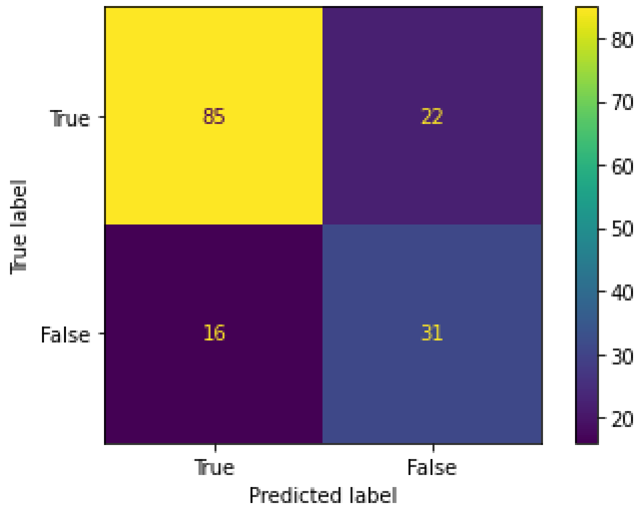

# Confusion Matrix For Decision Tree :

In [210…

```
confusion_matrix_DT = metrics.confusion_matrix(y_test,y_pred3_DT)
confusion_matrix_DT = metrics.ConfusionMatrixDisplay(confusion_matrix_DT, display_labels=[True,False])
confusion_matrix_DT.plot()
```

Out[210]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14cd6462820>
```



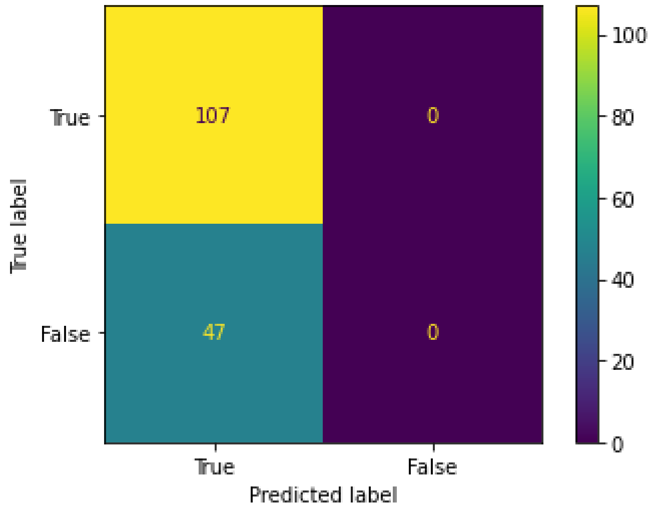
## Confusion Matrix For SVM:

In [211…

```
confusion_matrix_SVM = metrics.confusion_matrix(y_test,y_pred4_SVM)
confusion_matrix_SVM = metrics.ConfusionMatrixDisplay(confusion_matrix_SVM, display_labels=[True,False])
confusion_matrix_SVM.plot()
```

Out[211]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14cd6859eb0>
```



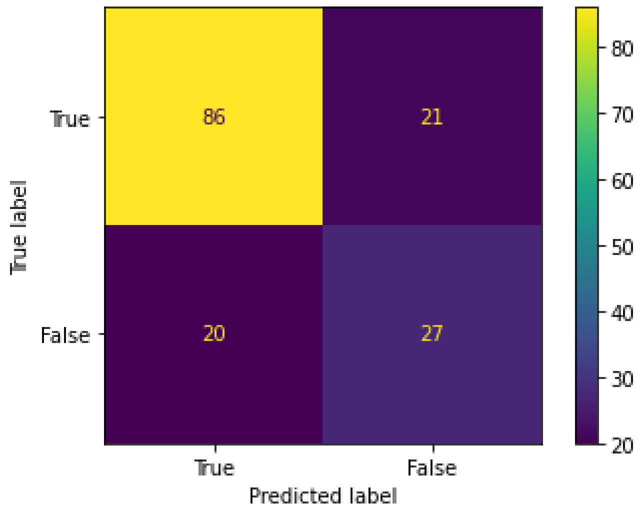

## Confusion Matrix For KNN:

In [212…

```
confusion_matrix_KNN = metrics.confusion_matrix(y_test,y_pred5_KNN)
confusion_matrix_KNN = metrics.ConfusionMatrixDisplay(confusion_matrix_KNN, display_labels=[True,False])
confusion_matrix_KNN.plot()
```

Out[212]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14cd68f2df0>
```




## Evaluation for Logistic Regression:

In [213…

```
from sklearn.metrics import accuracy_score
print('Accuracy=',accuracy_score(y_test,y_pred1_LR))

from sklearn.metrics import precision_score
print('Precision',precision_score(y_test,y_pred1_LR))

from sklearn.metrics import recall_score
print('Recall_score=',recall_score(y_test,y_pred1_LR))

from sklearn.metrics import f1_score
print('F1_Score=',f1_score(y_test,y_pred1_LR))
```

```
Accuracy= 0.8246753246753247
Precision 0.7777777777777778
Recall_score= 0.5957446808510638
F1_Score= 0.674698795180723
```

## Evaluation for Random Forest:

In [214…

```
from sklearn.metrics import accuracy_score
print('Accuracy=',accuracy_score(y_test,y_pred2_RF))
```

```
from sklearn.metrics import precision_score
print('Precision',precision_score(y_test,y_pred2_RF))

from sklearn.metrics import recall_score
print('Recall_score=',recall_score(y_test,y_pred2_RF))

from sklearn.metrics import f1_score
print('F1_Score=',f1_score(y_test,y_pred2_RF))
```

```
Accuracy= 0.8051948051948052
Precision 0.6808510638297872
Recall_score= 0.6808510638297872
F1_Score= 0.6808510638297872
```

# Evaluation for Decision Tree:

In [215…

```
from sklearn.metrics import accuracy_score
print('Accuracy=',accuracy_score(y_test,y_pred3_DT))

from sklearn.metrics import precision_score
print('Precision',precision_score(y_test,y_pred3_DT))

from sklearn.metrics import recall_score
print('Recall_score=',recall_score(y_test,y_pred3_DT))

from sklearn.metrics import f1_score
print('F1_Score=',f1_score(y_test,y_pred3_DT))
```

```
Accuracy= 0.7532467532467533
Precision 0.5849056603773585
Recall_score= 0.6595744680851063
F1_Score= 0.62
```

# Evaluation for SVM:

In [216…

```
from sklearn.metrics import accuracy_score
print('Accuracy=',accuracy_score(y_test,y_pred4_SVM))

from sklearn.metrics import precision_score
print('Precision',precision_score(y_test,y_pred4_SVM))

from sklearn.metrics import recall_score
```

```
print('Recall_score=',recall_score(y_test,y_pred4_SVM))

from sklearn.metrics import f1_score
print('F1_Score=',f1_score(y_test,y_pred4_SVM))
```

```
Accuracy= 0.6948051948051948
Precision 0.0
Recall_score= 0.0
F1_Score= 0.0
```

## Evaluation for KNN:

In [217…

```
from sklearn.metrics import accuracy_score
print('Accuracy=',accuracy_score(y_test,y_pred5_KNN))

from sklearn.metrics import precision_score
print('Precision',precision_score(y_test,y_pred5_KNN))

from sklearn.metrics import recall_score
print('Recall_score=',recall_score(y_test,y_pred5_KNN))

from sklearn.metrics import f1_score
print('F1_Score=',f1_score(y_test,y_pred5_KNN))
```

```
Accuracy= 0.7337662337662337
Precision 0.5625
Recall_score= 0.574468085106383
F1_Score= 0.5684210526315789
```

# 2. Compare various models with the results from KNN algorithm:

In [218…

```
print('Accuracy_LR=',accuracy_score(y_test,y_pred1_LR))
print('Accuracy_RF=',accuracy_score(y_test,y_pred2_RF))
print('Accuracy_DT=',accuracy_score(y_test,y_pred3_DT))
print('Accuracy_SVM=',accuracy_score(y_test,y_pred4_SVM))
print('Accuracy_KNN=',accuracy_score(y_test,y_pred5_KNN))
```

```
Accuracy_LR= 0.8246753246753247
Accuracy_RF= 0.8051948051948052
Accuracy_DT= 0.7532467532467533
Accuracy_SVM= 0.6948051948051948
Accuracy_KNN= 0.7337662337662337
```

Note:In my learnings KNN algorithm gives average accuracy compare to remaining models

In [219…
```
print('Precision_LR=',precision_score(y_test,y_pred1_LR))
print('Precision_RF=',precision_score(y_test,y_pred2_RF))
print('Precision_DT=',precision_score(y_test,y_pred3_DT))
print('Precision_SVM=',precision_score(y_test,y_pred4_SVM))
print('Precision_KNN=',precision_score(y_test,y_pred5_KNN))
```

```
Precision_LR= 0.7777777777777778
Precision_RF= 0.6808510638297872
Precision_DT= 0.5849056603773585
Precision_SVM= 0.0
Precision_KNN= 0.5625
```

Note:In my learnings KNN algorithm gives low precison compare to remaining models

In [220…
```
print('Recall_Score_LR=',recall_score(y_test,y_pred1_LR))
print('Recall_Score_RF=',recall_score(y_test,y_pred2_RF))
print('Recall_Score_DT=',recall_score(y_test,y_pred3_DT))
print('Recall_Score_SVM=',recall_score(y_test,y_pred4_SVM))
print('Recall_Score_KNN=',recall_score(y_test,y_pred5_KNN))
```

```
Recall_Score_LR= 0.5957446808510638
Recall_Score_RF= 0.6808510638297872
Recall_Score_DT= 0.6595744680851063
Recall_Score_SVM= 0.0
Recall_Score_KNN= 0.574468085106383
```

In [221…
```
print('F1_Score_LR=',f1_score(y_test,y_pred1_LR))
print('F1_Score_RF=',f1_score(y_test,y_pred2_RF))
print('F1_Score_DT=',f1_score(y_test,y_pred3_DT))
print('F1_Score_SVM=',f1_score(y_test,y_pred4_SVM))
print('F1_Score_KNN=',f1_score(y_test,y_pred5_KNN))
```

```
F1_Score_LR= 0.674698795180723
F1_Score_RF= 0.6808510638297872
F1_Score_DT= 0.62
F1_Score_SVM= 0.0
F1_Score_KNN= 0.5684210526315789
```

# Week 4-Data Modeling

# 1. Create a classification report by analyzing sensitivity, specificity, AUC (RO

## Classification Reports:

In [222…

```
from sklearn.metrics import classification_report
print('---Classification Report For LR'   )

print(classification_report(y,model1.predict(x)))
print('-'*125)
print('---Classification Report For RF'   )
print(classification_report(y,model2.predict(x)))
print('-'*125)
print('---Classification Report For DT'   )
print(classification_report(y,model3.predict(x)))
print('-'*125)
print('---Classification Report For SVM'   )
print(classification_report(y,model4.predict(x)))
print('-'*125)
print('---Classification Report For KNN'   )
print(classification_report(y,model5.predict(x)))
```

```
---Classification Report For LR
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       500
           1       0.75      0.55      0.64       268

    accuracy                           0.78       768
   macro avg       0.77      0.73      0.74       768
weighted avg       0.78      0.78      0.77       768

-----------------------------------------------------------------------------------------------------------------------------
---Classification Report For RF
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       500
           1       0.93      0.93      0.93       268

    accuracy                           0.95       768
   macro avg       0.95      0.95      0.95       768
weighted avg       0.95      0.95      0.95       768

-----------------------------------------------------------------------------------------------------------------------------
---Classification Report For DT
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       500
           1       0.92      0.94      0.93       268

    accuracy                           0.95       768
   macro avg       0.94      0.95      0.95       768
weighted avg       0.95      0.95      0.95       768

-----------------------------------------------------------------------------------------------------------------------------
---Classification Report For SVM
              precision    recall  f1-score   support

           0       0.91      1.00      0.96       500
           1       1.00      0.82      0.90       268

    accuracy                           0.94       768
   macro avg       0.96      0.91      0.93       768
weighted avg       0.94      0.94      0.94       768

-----------------------------------------------------------------------------------------------------------------------------
```

```
---Classification Report For KNN
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       500
           1       0.70      0.62      0.66       268

    accuracy                           0.77       768
   macro avg       0.75      0.74      0.74       768
weighted avg       0.77      0.77      0.77       768
```

# Create AUC(ROC Curves) Receiver Operating Characteristics Curve:

In [223…

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

# ROC Curve for LR:

In [224…

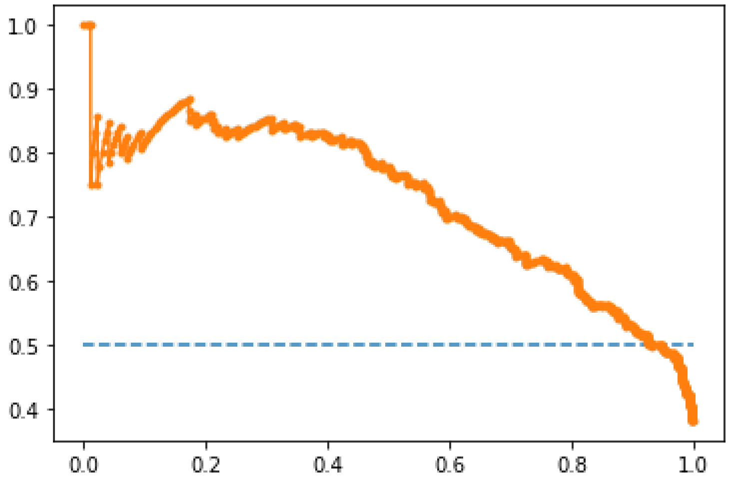
```
#Precision Recall Curve for Logistic Regression

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
pred1 = model1.predict_proba(x)
# keep probabilities for the positive outcome only
pred1 = pred1[:, 1]
# predict class values
y_pred1_LR = model1.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y, pred1)
# calculate F1 score
f1 = f1_score(y_test,y_pred1_LR)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y, pred1)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
```

```
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.675 auc=0.727 ap=0.728
```

Out[224]:

```
[<matplotlib.lines.Line2D at 0x14cd69d27c0>]
```




## ROC Curve for Random Forest:

In [225…

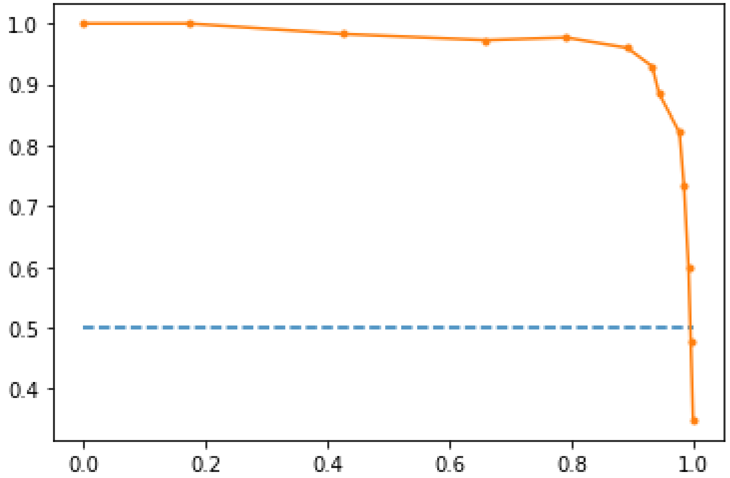
```
#Precision Recall Curve for Random Forest

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
pred2 = model2.predict_proba(x)
# keep probabilities for the positive outcome only
pred2 = pred2[:, 1]
# predict class values
y_pred2_RF= model2.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y, pred2)
# calculate F1 score
f1 = f1_score(y_test,y_pred2_RF)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
```

```
ap = average_precision_score(y, pred2)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.681 auc=0.970 ap=0.963
```

Out[225]:

```
[<matplotlib.lines.Line2D at 0x14cd6a3d580>]
```




# ROC Curve for Decision Tree:

In [226…

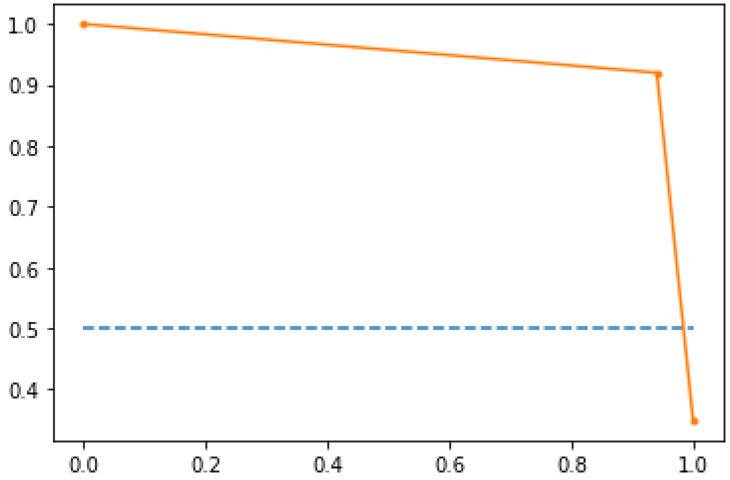
```
#Precision Recall Curve for Decission Tree Classifier

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
pred3 = model3.predict_proba(x)
# keep probabilities for the positive outcome only
pred3 = pred3[:, 1]
# predict class values
y_pred3_DT= model3.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y, pred3)
# calculate F1 score
```

```
f1 = f1_score(y_test,y_pred3_DT)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y, pred3)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.620 auc=0.940 ap=0.886
```

Out[226]:

```
[<matplotlib.lines.Line2D at 0x14cd7a76520>]
```




# ROC Curve for KNN:

In [227…

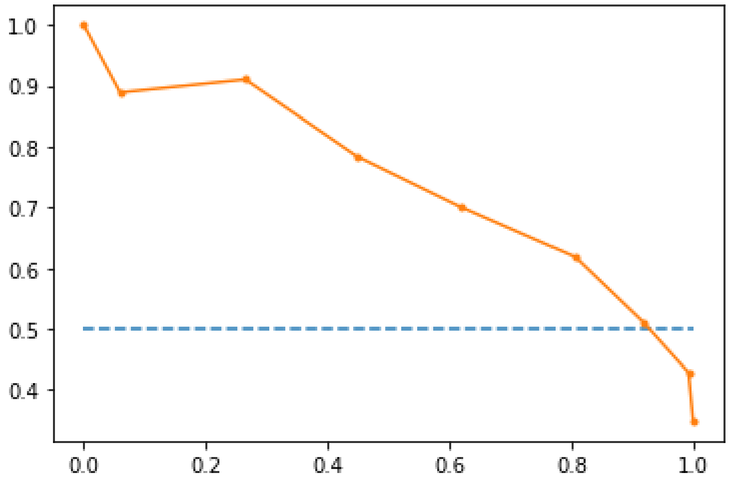
```
#Precision Recall Curve for KNN

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
pred5 = model5.predict_proba(x)
# keep probabilities for the positive outcome only
pred5 = pred5[:, 1]
# predict class values
```

```python
y_pred5_KNN= model5.predict(x_test)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y, pred5)
# calculate F1 score
f1 = f1_score(y_test,y_pred5_KNN)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y, pred5)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

```
f1=0.568 auc=0.748 ap=0.711
```

Out[227]:

```
[<matplotlib.lines.Line2D at 0x14cd7adb5b0>]
```




# Submitted by:

# P Maniraj Kumar