PGP - Data Science Capstone

Project 2 - PGP Healthcare

Report and Source Code

Importing libraries

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

Importing the dataset

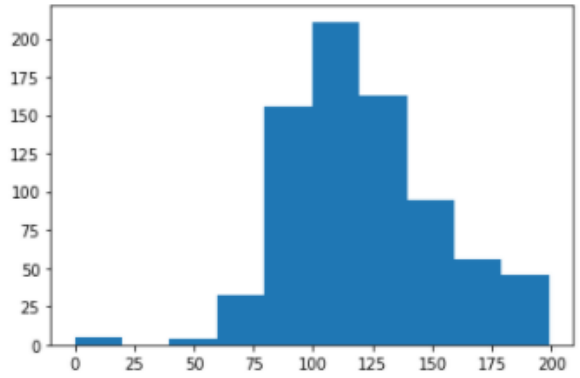
```
data = pd.read_csv("health care diabetes.csv")
data.head()
```

Week 1

```
data.isnull().any()
data.info()
```

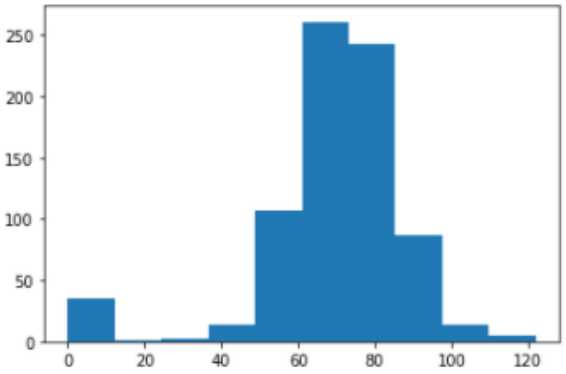
Performing Descriptive Analysis and Visualization

```
data['Glucose'].value_counts().head(7)
plt.hist(data['Glucose'])
```



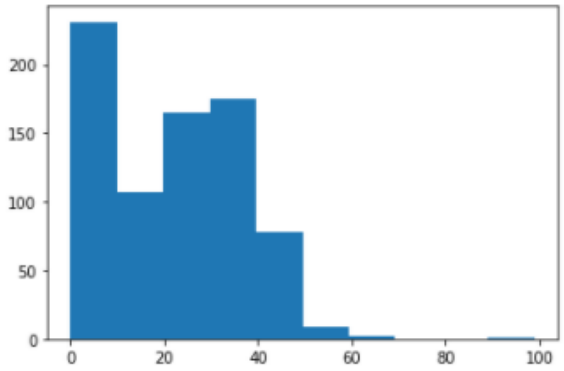
data['BloodPressure'].value_counts().head(7)

plt.hist(data['BloodPressure'])



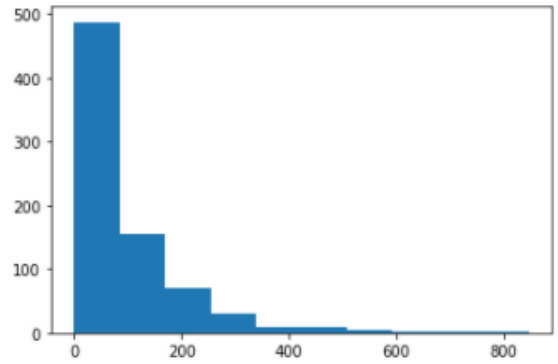
data['SkinThickness'].value_counts().head(7)

plt.hist(data['SkinThickness'])



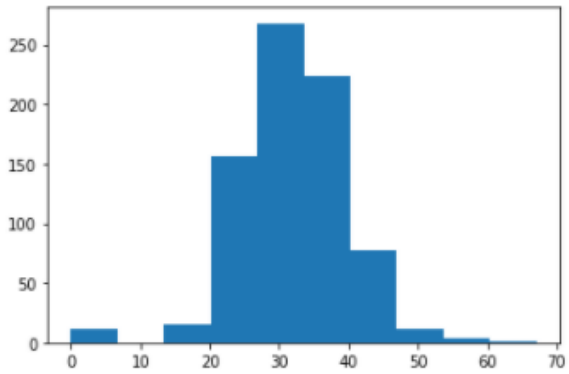
data['Insulin'].value_counts().head(7)

plt.hist(data['Insulin'])



```
data['BMI'].value_counts().head(7)
```

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

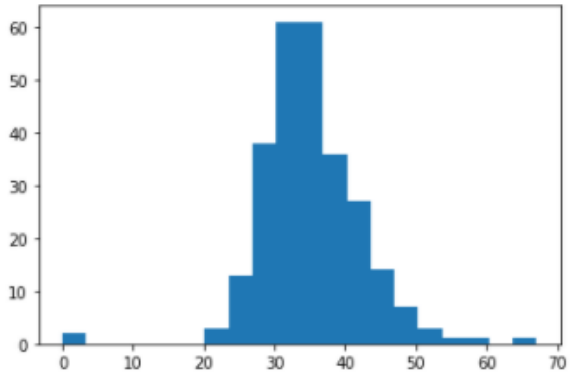



```
Positive = data[data['Outcome']==1]
```

```
Positive.head(5)
```

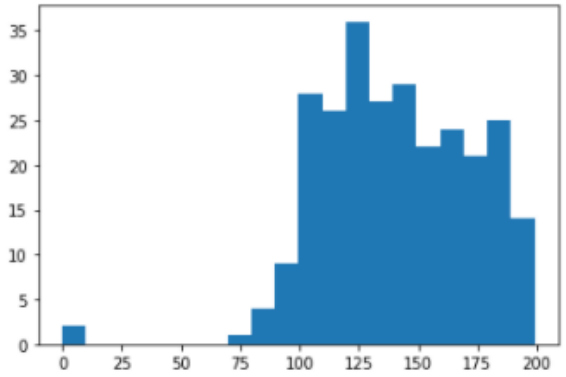
```
plt.hist(Positive['BMI'],histtype='stepfilled',bins=20)
```

```
Positive['BMI'].value_counts().head(7)
```



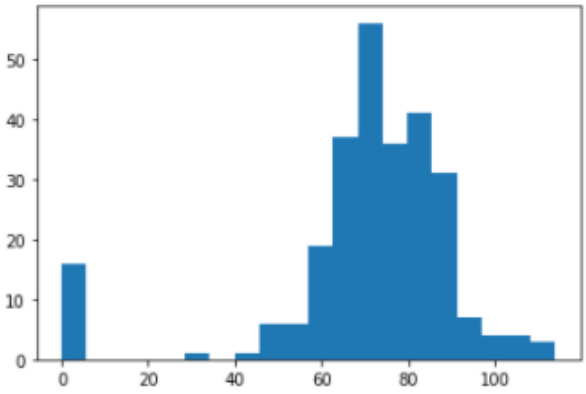

```
plt.hist(Positive['Glucose'],histtype='stepfilled',bins=20)
```

```
Positive['Glucose'].value_counts().head(7)
```



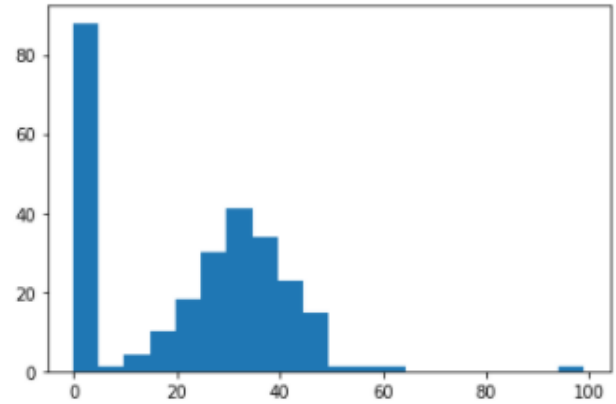
plt.hist(Positive['BloodPressure'],histtype='stepfilled',bins=20)

Positive['BloodPressure'].value_counts().head(7)



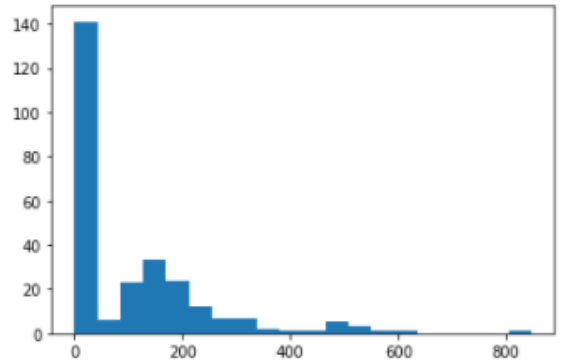
plt.hist(Positive['SkinThickness'],histtype='stepfilled',bins=20)

Positive['SkinThickness'].value_counts().head(7)



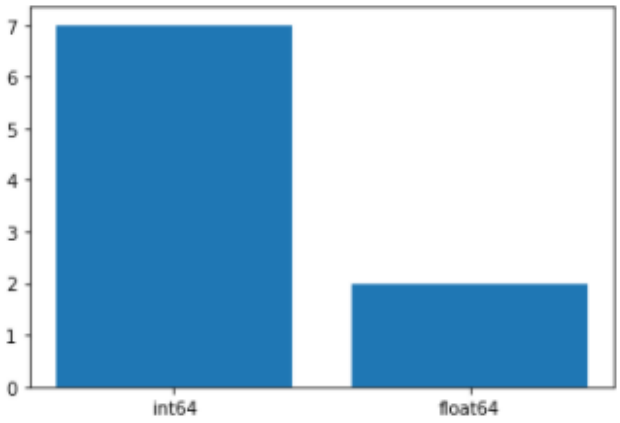
plt.hist(Positive['Insulin'],histtype='stepfilled',bins=20)

Positive['Insulin'].value_counts().head(7)



Creating a count (frequency) plot describing the data types and the count of variables

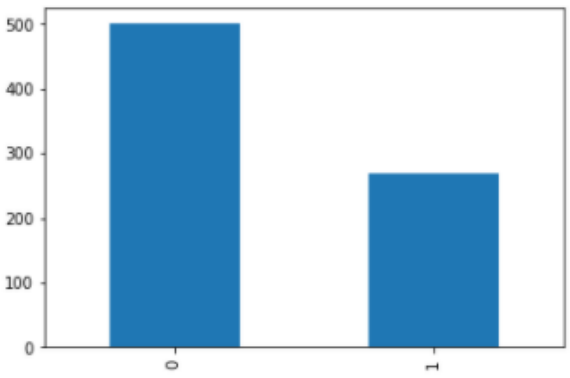
```
data.describe().transpose()


datatype_dict = dict()
for k in data.columns:
    datatype = data.dtypes[k].name
    if datatype not in datatype_dict:
        datatype_dict[datatype] = 0
    datatype_dict[datatype] += 1
plt.bar(datatype_dict.keys(), datatype_dict.values())
```



Week 2

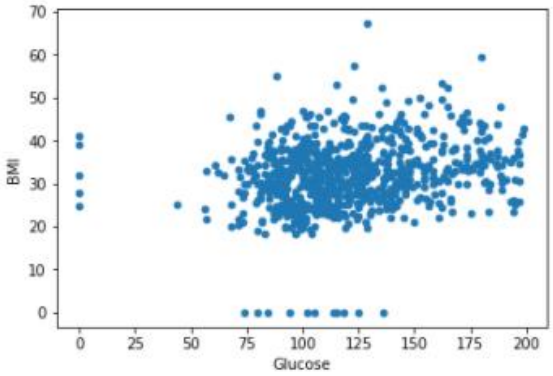
Checking the balance of the data by plotting the count of outcomes by their value

```
fig, ax = plt.subplots()
data['Outcome'].value_counts().plot(ax=ax, kind='bar')
```

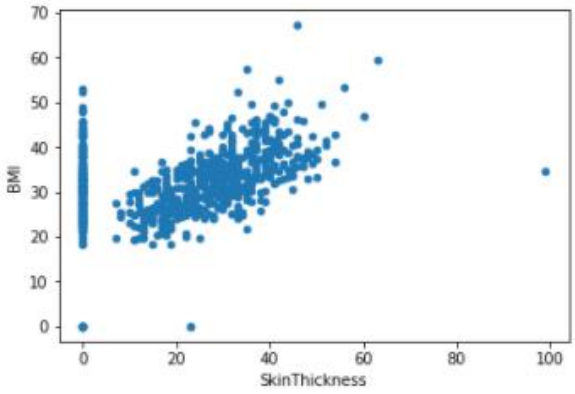


Scatter Plots

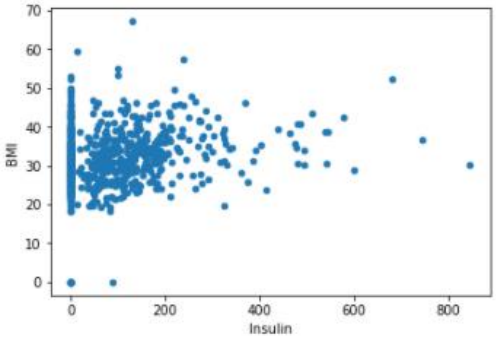
data.plot.scatter("Glucose", "BMI")



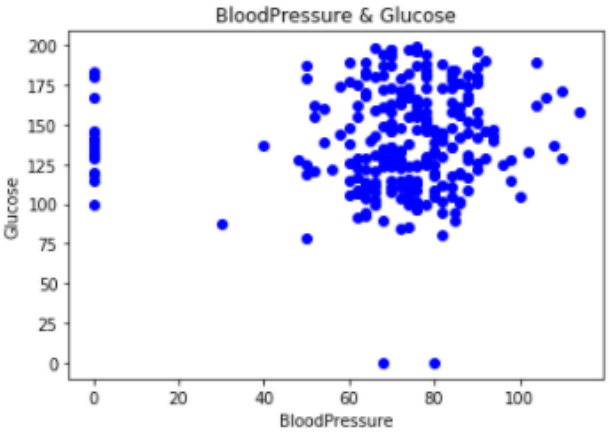
data.plot.scatter("SkinThickness", "BMI")



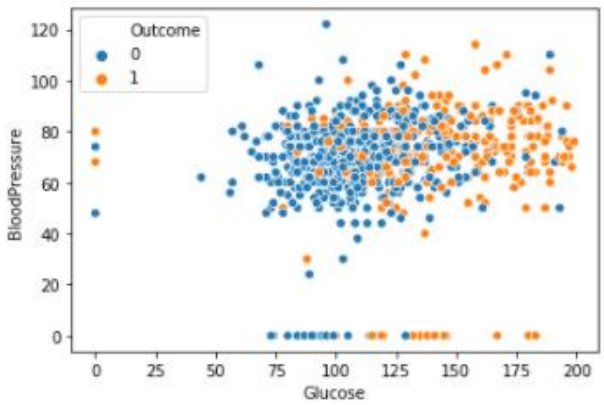
data.plot.scatter("Insulin", "BMI")



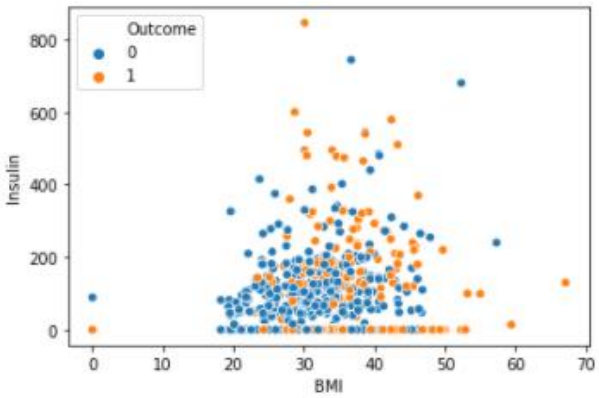
```
BloodPressure = Positive['BloodPressure']

Glucose = Positive['Glucose']

SkinThickness = Positive['SkinThickness']

Insulin = Positive['Insulin']

BMI = Positive['BMI']


plt.scatter(BloodPressure, Glucose, color=['b'])

plt.xlabel('BloodPressure')

plt.ylabel('Glucose')

plt.title('BloodPressure & Glucose')

plt.show()
```



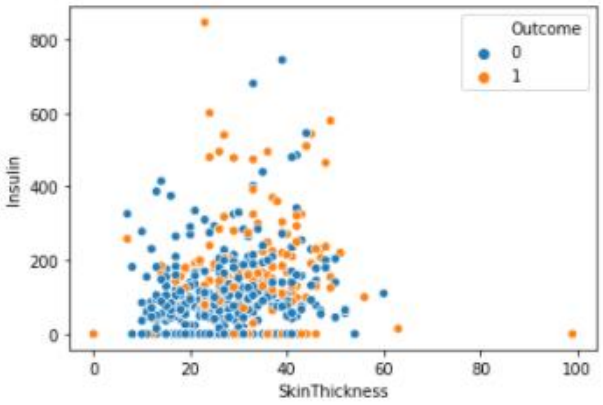

```
g = sns.scatterplot(x= "Glucose", y= "BloodPressure", hue="Outcome", data=data);
```



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



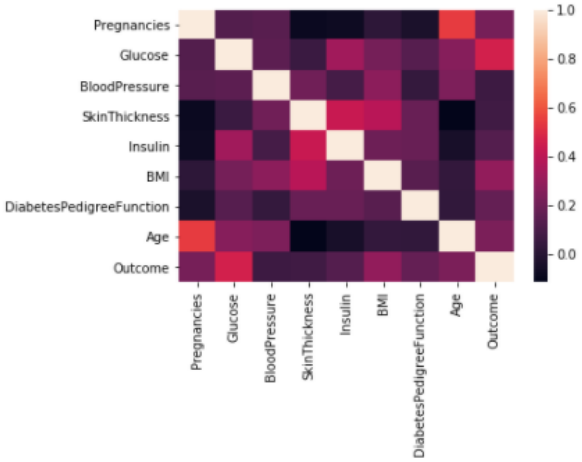
S = sns.scatterplot(x= "SkinThickness" ,y= "Insulin", hue="Outcome", data=data);



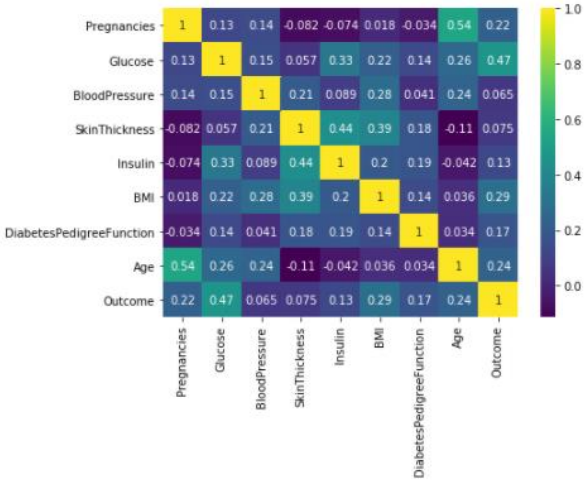
Correlation Matrix and Heat Maps

data.corr()

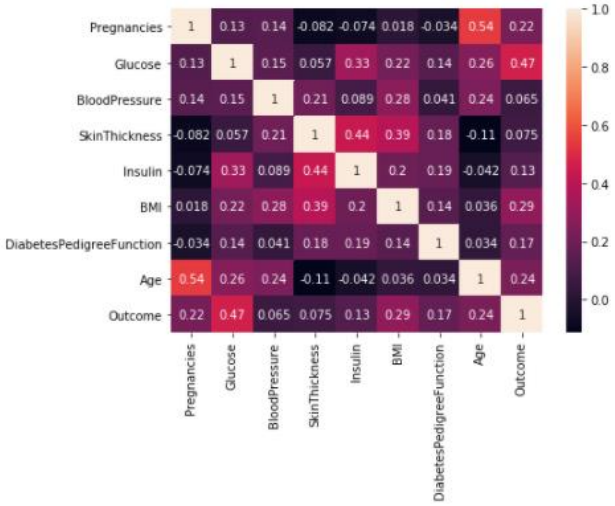
sns.heatmap(data.corr())



```
plt.subplots(figsize=(7,5))
sns.heatmap(data.corr(),annot=True,cmap='viridis')
```




```
plt.subplots(figsize=(7,5))
sns.heatmap(data.corr(),annot=True)
```



Week 3 and 4

Logistic Regression and Model Building

```
data.head(5)
#Train Test Split
features = data.iloc[:,[0,1,2,3,4,5,6,7]].values
label = data.iloc[:,8].values
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features, label, test_size=0.2,  random_state =10)
```

Creating Model

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(X_train,y_train)

print(model.score(X_train,y_train))
```

0.7719869706840391

```
print(model.score(X_test,y_test))
```

0.7662337662337663

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(label,model.predict(features))

cm
```

```
from sklearn.metrics import classification_report

print(classification_report(label,model.predict(features)))
```

	precision	recall	f1-score	support
0	0.79	0.89	0.84	500
1	0.73	0.54	0.62	268
accuracy			0.77	768
macro avg	0.76	0.72	0.73	768
weighted avg	0.77	0.77	0.76	768

Preparing ROC Curve (Receiver Operating Characteristics Curve)

```
from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score
```

Predict Probabilities

```
probs = model.predict_proba(features)
```

Keeping Probabilities for the Positive Outcome only

```
probs = probs[:, 1]
```

Calculating AUC

```
auc = roc_auc_score(label, probs)

print('AUC: %.3f' % auc)
```

Calculate ROC Curve

```
fpr, tpr, thresholds = roc_curve(label, probs)
```

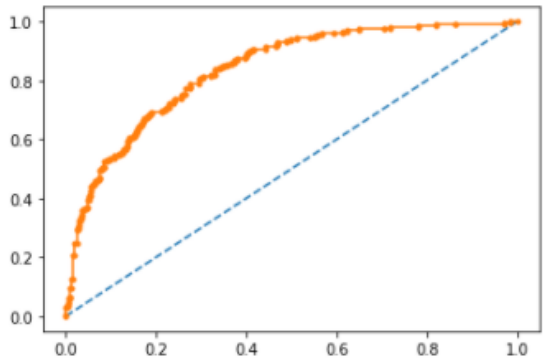
Plotting No skill

```
plt.plot([0, 1], [0, 1], linestyle='--')
```

Plotting the ROC curve for the Model

```
plt.plot(fpr, tpr, marker='.')
```

AUC: 0.837



Applying Decission Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

model3 = DecisionTreeClassifier(max_depth=5)

model3.fit(X_train,y_train)
```

```
model3.score(X_train,y_train)
```

0.8289902280130294

```
model3.score(X_test,y_test)
```

0.7597402597402597

Applying Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier(n_estimators=11)
model4.fit(X_train,y_train)

model4.score(X_train,y_train)
0.995114006514658

model4.score(X_test,y_test)
0.7402597402597403
```

Support Vector Classifier

```
from sklearn.svm import SVC
model5 = SVC(kernel='rbf',
        gamma='auto')
model5.fit(X_train,y_train)


model5.score(X_train,y_train)

1.0


model5.score(X_test,y_test)

0.6168831168831169
```

Applying KNN

```
from sklearn.neighbors import KNeighborsClassifier

model2 = KNeighborsClassifier(n_neighbors=7,

                    metric='minkowski',

                    p = 2)

model2.fit(X_train,y_train)
```

Preparing ROC Curve (Receiver Operating Characteristics Curve)

```
from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score
```

Predicting Probabilities

```
probs = model2.predict_proba(features)
```

Keeping Probabilities for the Positive Outcome only

probs = probs[:, 1]

Calculating AUC

auc = roc_auc_score(label, probs)

print('AUC: %.3f' % auc)

AUC: 0.836

Calculating ROC Curve

fpr, tpr, thresholds = roc_curve(label, probs)

print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr,thresholds))

Plotting No Skill

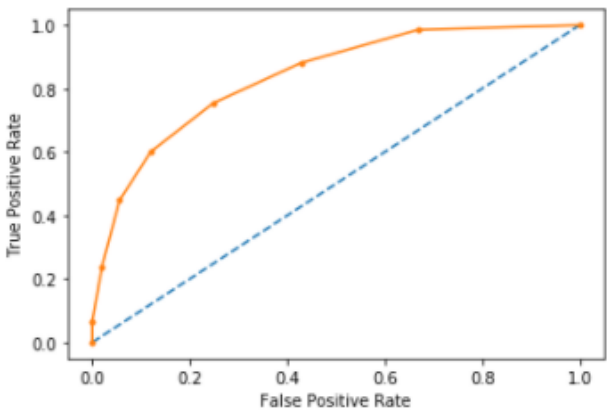
plt.plot([0, 1], [0, 1], linestyle='--')

Plotting the ROC curve for the Model

plt.plot(fpr, tpr, marker='.')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")



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
```

Predicting Probabilities

```
probs = model.predict_proba(features)
```

Keeping Probabilities for the Positive Outcome only

```
probs = probs[:, 1]
```

Predicting Class Values

```
yhat = model.predict(features)
```

Calculating Precision Recall Curve

```
precision, recall, thresholds = precision_recall_curve(label, probs)
```

Calculating F1 Score

```
f1 = f1_score(label, yhat)
```

Calculating Precision Recall AUC

```
auc = auc(recall, precision)
```

Calculating Average Precision Score

```
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
```

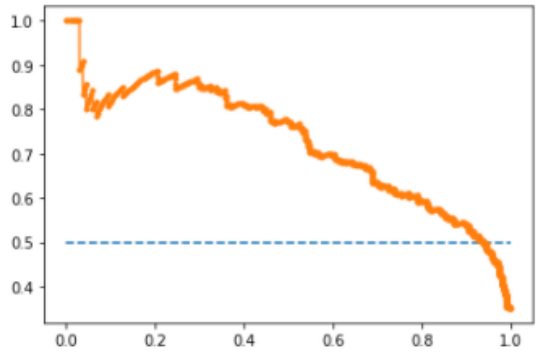
Plotting No Skill

```
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
```

Plotting the Precision Recall Curve for the Model

plt.plot(recall, precision, marker='.')

f1=0.624 auc=0.726 ap=0.727



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

Predicting Probabilities

probs = model2.predict_proba(features)

Keeping Probabilities for the Positive Outcome only

probs = probs[:, 1]

Predicting Class Values

yhat = model2.predict(features)

Calculating Precision Recall Curve

precision, recall, thresholds = precision_recall_curve(label, probs)

Calculating F1 Score

f1 = f1_score(label, yhat)

Calculating Precision Recall AUC

auc = auc(recall, precision)

Calculating Average Precision Score

ap = average_precision_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

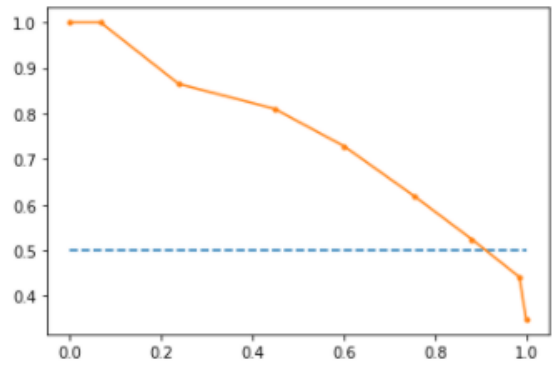
Plotting No Skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

Plotting the Precision Recall Curve for the Model

plt.plot(recall, precision, marker='.')

f1=0.658 auc=0.752 ap=0.709



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

Predicting Probabilities

probs = model3.predict_proba(features)

Keeping Probabilities for the Positive Outcome only

probs = probs[:, 1]

Predicting Class Values

yhat = model3.predict(features)

Calculating Precision Recall Curve

precision, recall, thresholds = precision_recall_curve(label, probs)

Calculating F1 Score

f1 = f1_score(label, yhat)

Calculating Precision Recall AUC

auc = auc(recall, precision)

Calculating Average Precision Score

ap = average_precision_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

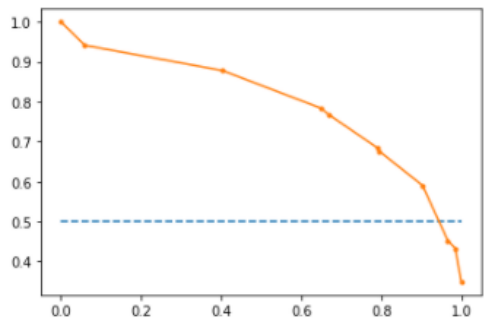
Plotting No Skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

Plotting the Precision Recall Curve for the Model

plt.plot(recall, precision, marker='.')

f1=0.710 auc=0.797 ap=0.758



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
```

Predicting Probabilities

```
probs = model4.predict_proba(features)
```

Keeping Probabilities for the Positive Outcome only

```
probs = probs[:, 1]
```

Predicting Class Values

```
yhat = model4.predict(features)
```

Calculating Precision Recall Curve

```
precision, recall, thresholds = precision_recall_curve(label, probs)
```

Calculating F1 Score

```
f1 = f1_score(label, yhat)
```

Calculating Precision Recall AUC

```
auc = auc(recall, precision)
```

Calculating Average Precision Score

```
ap = average_precision_score(label, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
```

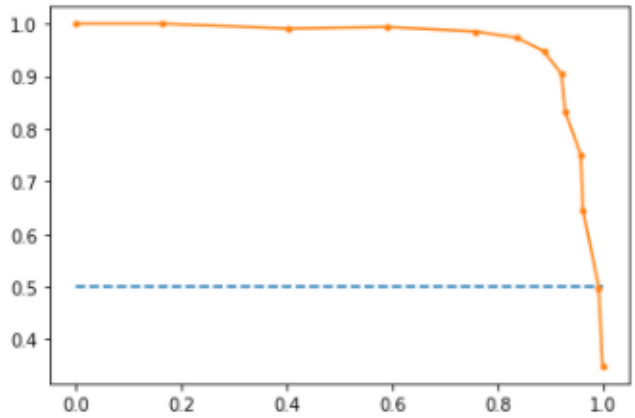
Plotting No Skill

```
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
```

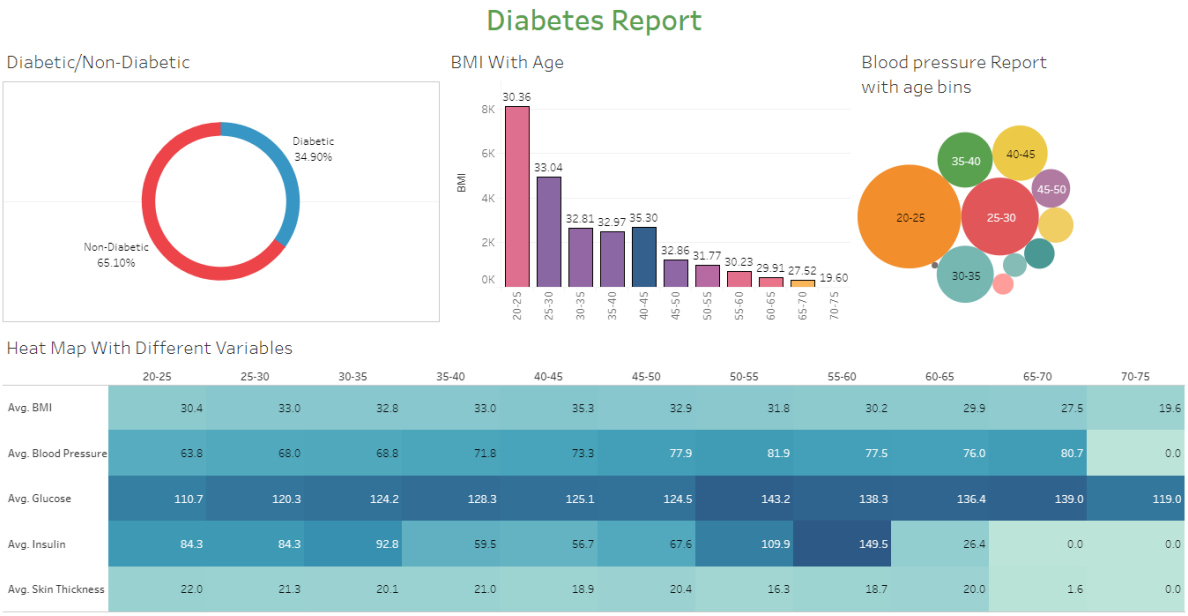
Plotting the Precision Recall Curve for the Model

plt.plot(recall, precision, marker='.')

f1=0.917 auc=0.964 ap=0.956



Tableau



Link: https://public.tableau.com/profile/apoorva.lakshman7065#!/vizhome/PGP-HealthcareCapstoneProjectApoorva/DiabetesReport?publish=yes