Using an SVM for this data

- Prepare the dataset, select, analyze and remove any null values
- Split and train the dataset. scale the data
- Use a grid search to find suitable hyperparameters
- Fit data to SVM, run cross validation and calculate the accuracy

```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns #for statistics ploting
import matplotlib.pyplot as plt
import math
import sklearn
%matplotlib inline
```

Display and view the data, analyze the data, check for null contents, processing and cleaning the data

```
diabetes_data= pd.read_csv('diabetes.csv')
diabetes_data
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
0              6      148             72             35        0  33.6

1              1       85             66             29        0  26.6

2              8      183             64              0        0  23.3

3              1       89             66             23       94  28.1

4              0      137             40             35      168  43.1

..           ...      ...            ...            ...      ...   ...

763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4
```

```
     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
diabetes_data.head(10)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1

5            5      116             74              0        0  25.6

6            3       78             50             32       88  31.0

7           10      115              0              0        0  35.3

8            2      197             70             45      543  30.5

9            8      125             96              0        0   0.0


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
```

```
8                       0.158   53         1
9                       0.232   54         1
```

```
diabetes_data.tail(10)
```

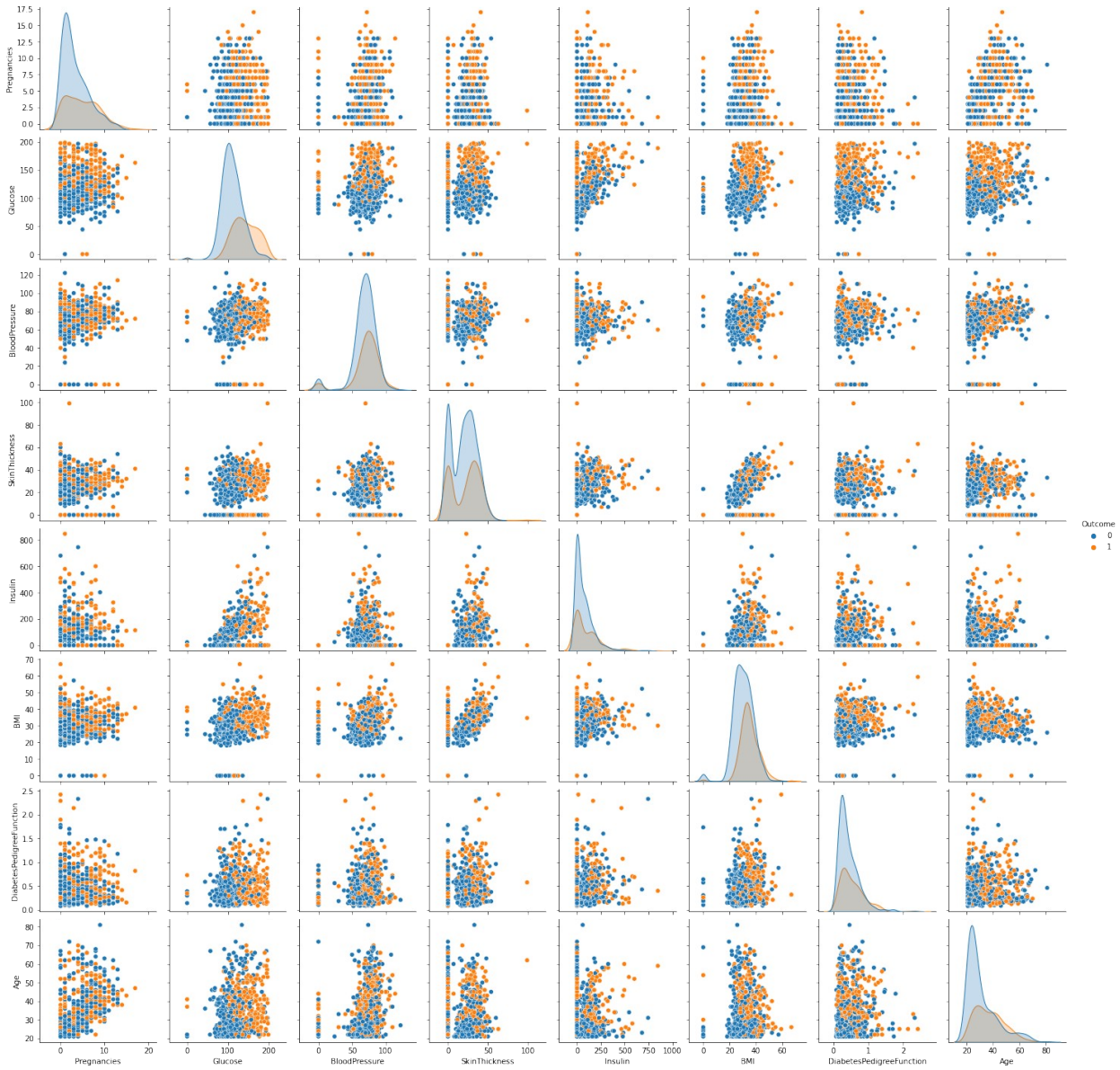
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
758            1      106             76              0        0  37.5

759            6      190             92              0        0  35.5

760            2       88             58             26       16  28.4

761            9      170             74             31        0  44.0

762            9       89             62              0        0  22.5

763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4


     DiabetesPedigreeFunction  Age  Outcome
758                     0.197   26        0
759                     0.278   66        1
760                     0.766   22        0
761                     0.403   43        1
762                     0.142   33        0
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

```
diabetes_data.index
```

```
RangeIndex(start=0, stop=768, step=1)
```

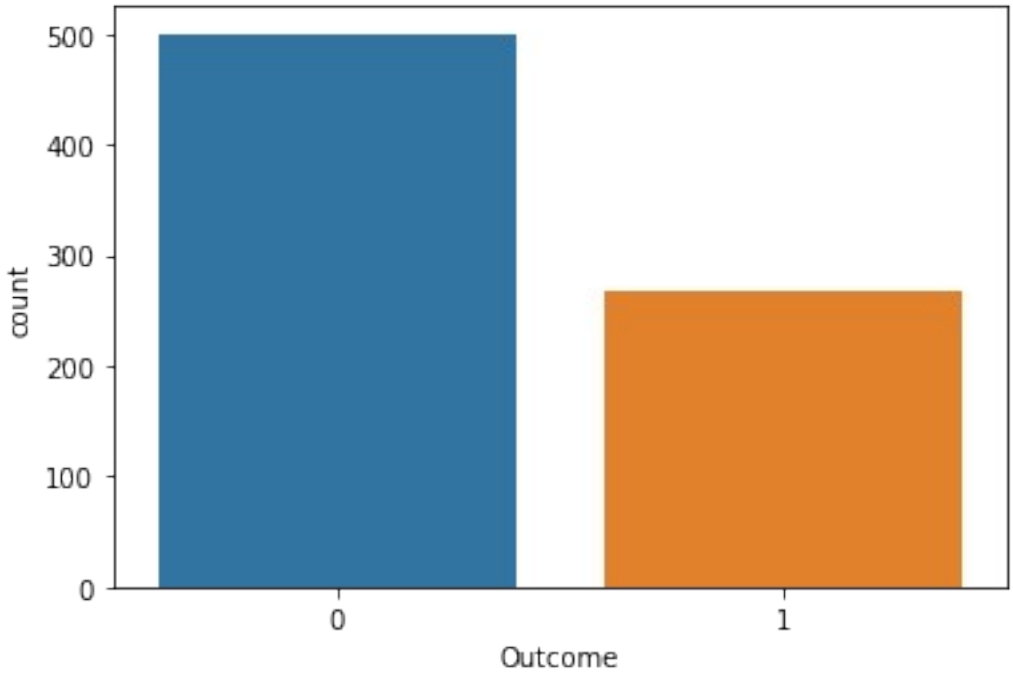
```
sns.pairplot(diabetes_data, hue='Outcome', vars=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness',
                                    'Insulin', 'BMI',
'DiabetesPedigreeFunction',
                                    'Age'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f516aa93220>
```



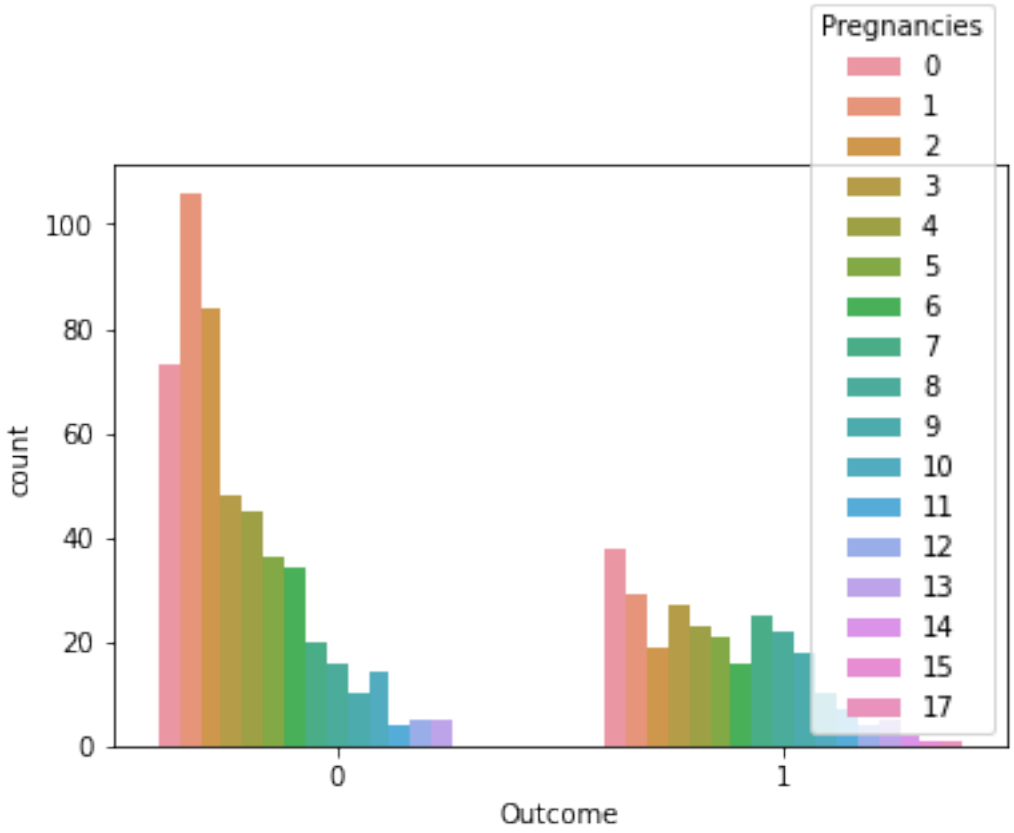

```
sns.countplot(x="Outcome", data=diabetes_data)
```

```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



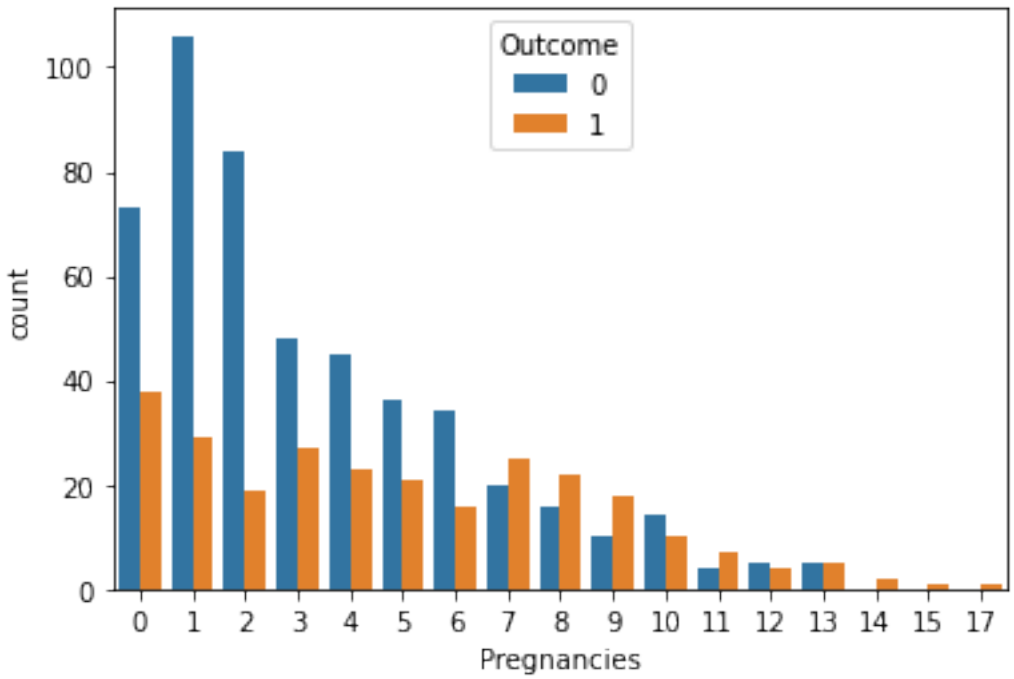

```
sns.countplot(x="Outcome",hue='Pregnancies', data=diabetes_data)
```

```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```




```
#sns.histplot(x="Outcome", y="Age", data=diabetes_data)
sns.countplot(x="Pregnancies",hue="Outcome", data=diabetes_data)
```

```
<AxesSubplot:xlabel='Pregnancies', ylabel='count'>
```

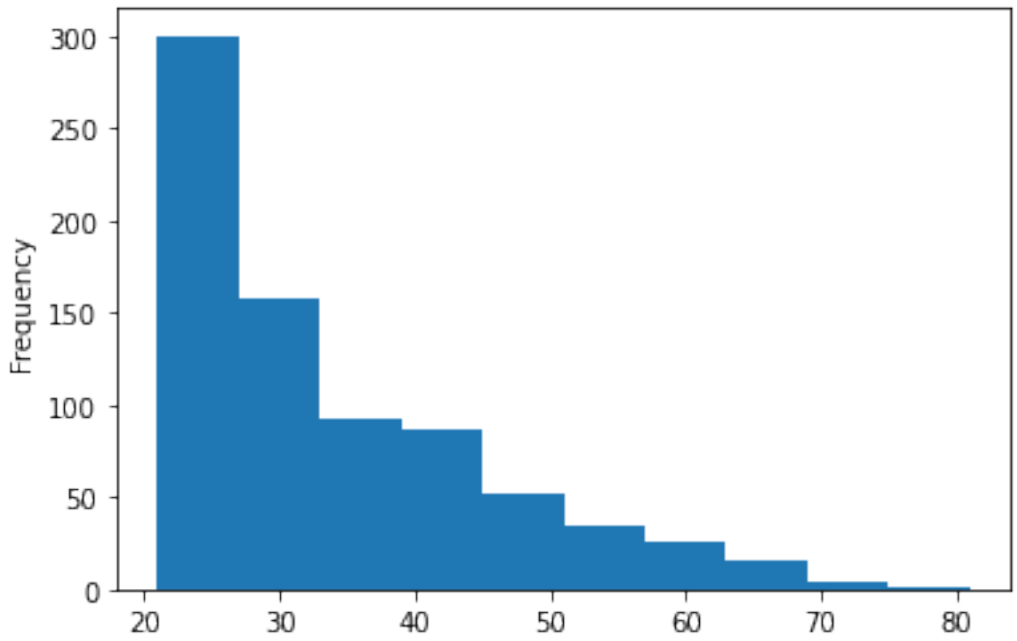



```
"""plt.hist(diabetes_data["Age"])

# Adding labels and title
plt.xlabel('Outcome')
plt.ylabel('Age')
plt.title('Age against outcome')

# Display the plot
plt.show()"""
```

```
'plt.hist(diabetes_data["Age"])\n \n# Adding labels and title\
nplt.xlabel(\'Outcome\')\nplt.ylabel(\'Age\')\nplt.title(\'Age against
outcome\')\n \n# Display the plot\nplt.show()'
```

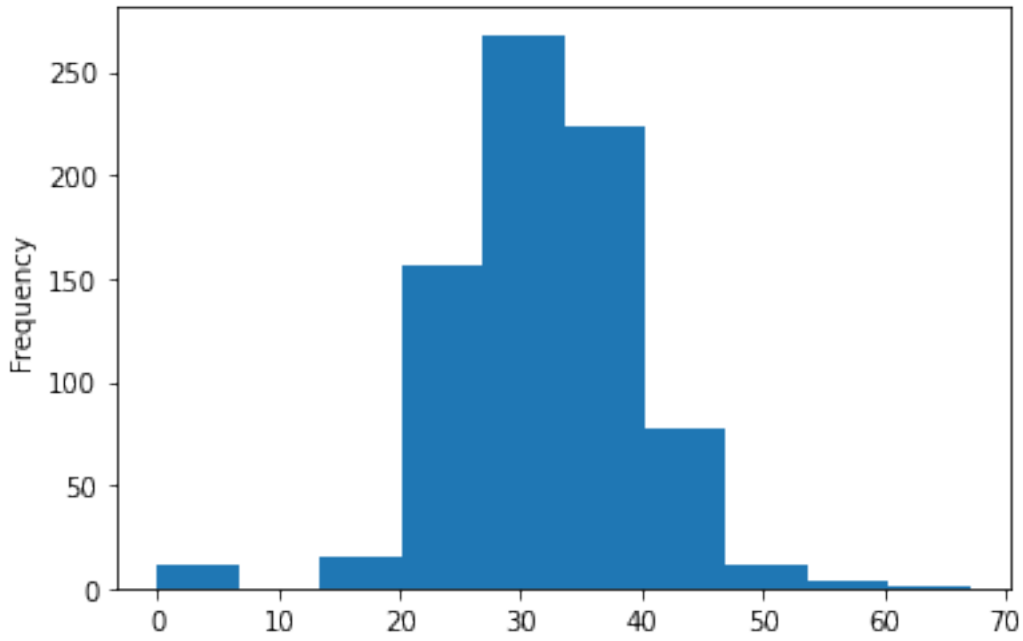
```
diabetes_data["Age"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



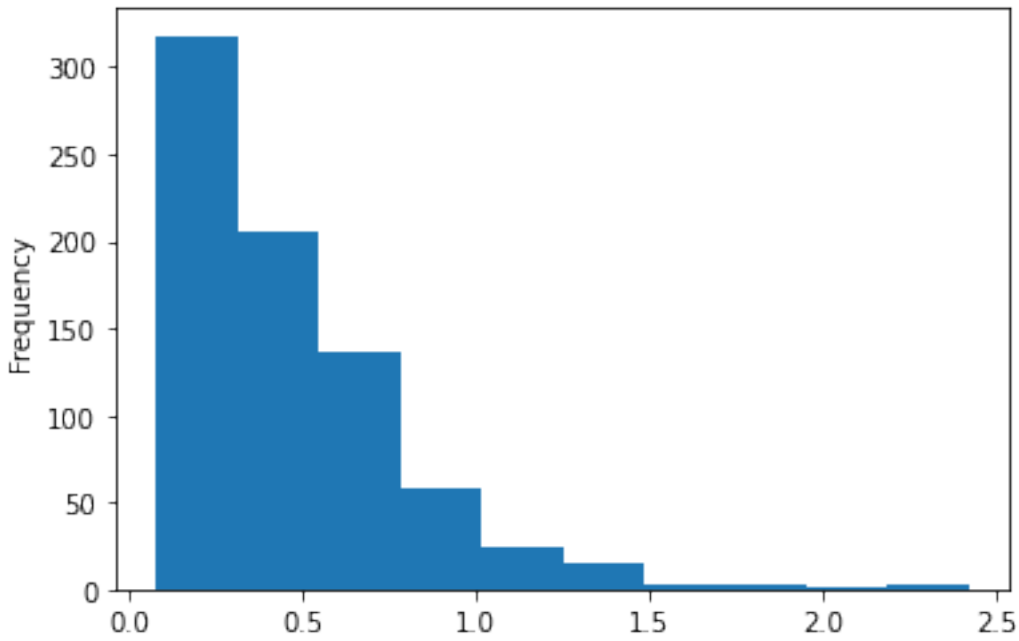

```
diabetes_data["BMI"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



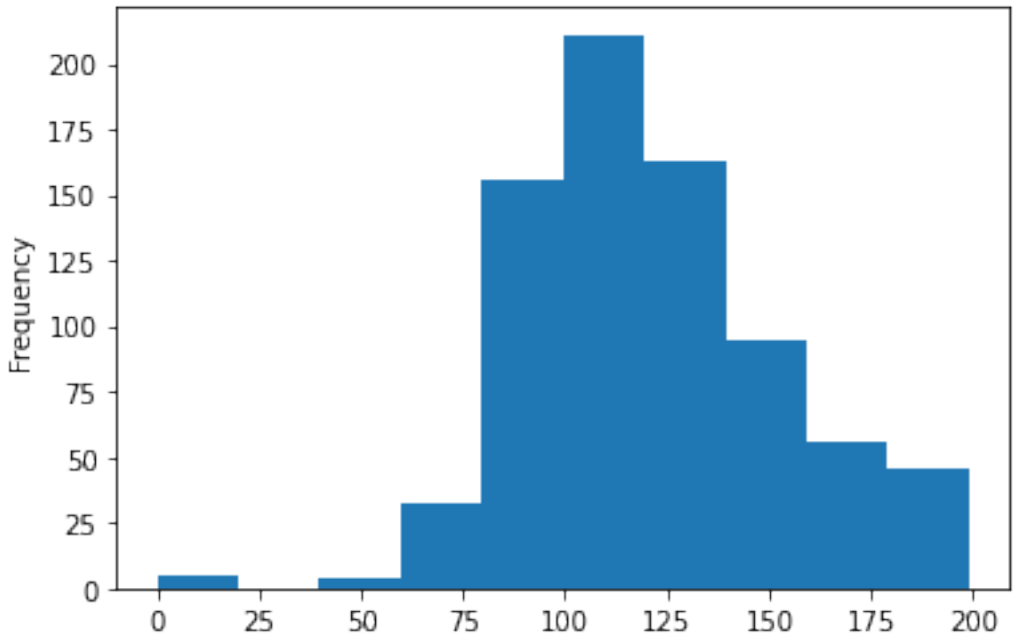

```
diabetes_data["DiabetesPedigreeFunction"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



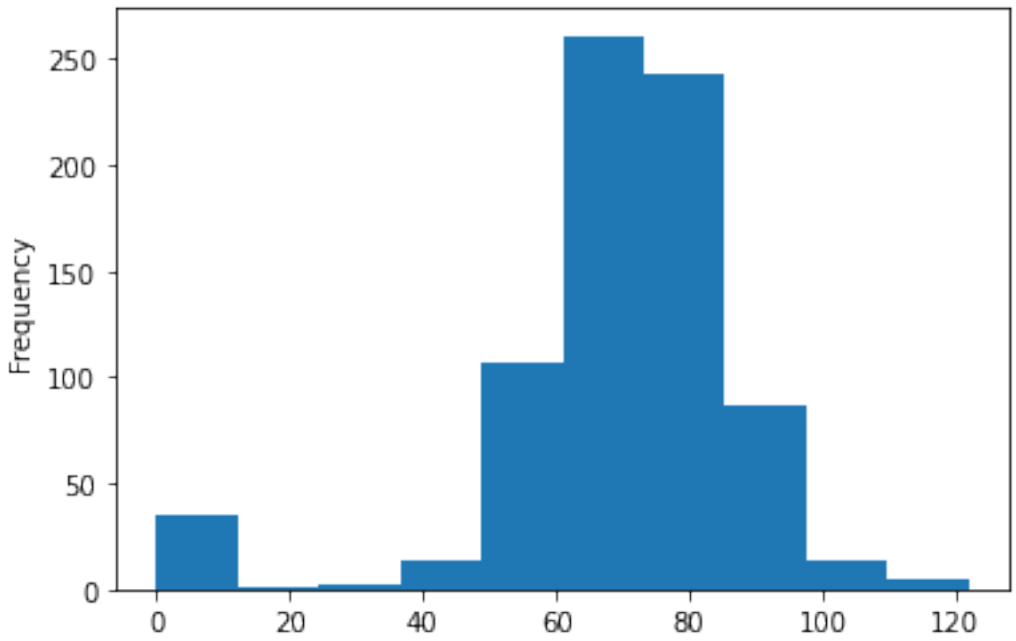

```
diabetes_data["Glucose"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



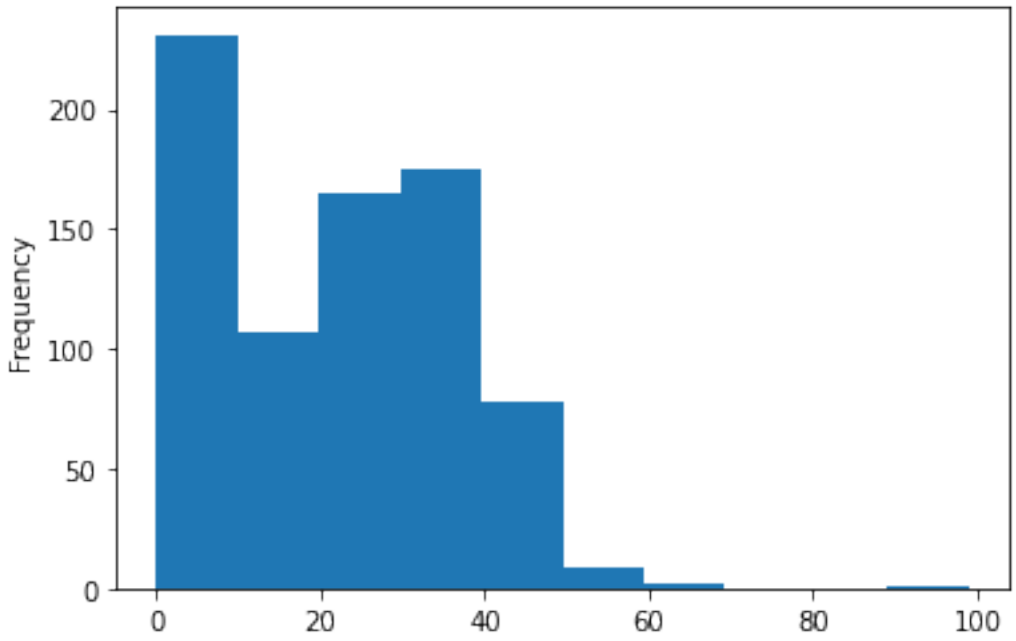

```
diabetes_data["BloodPressure"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



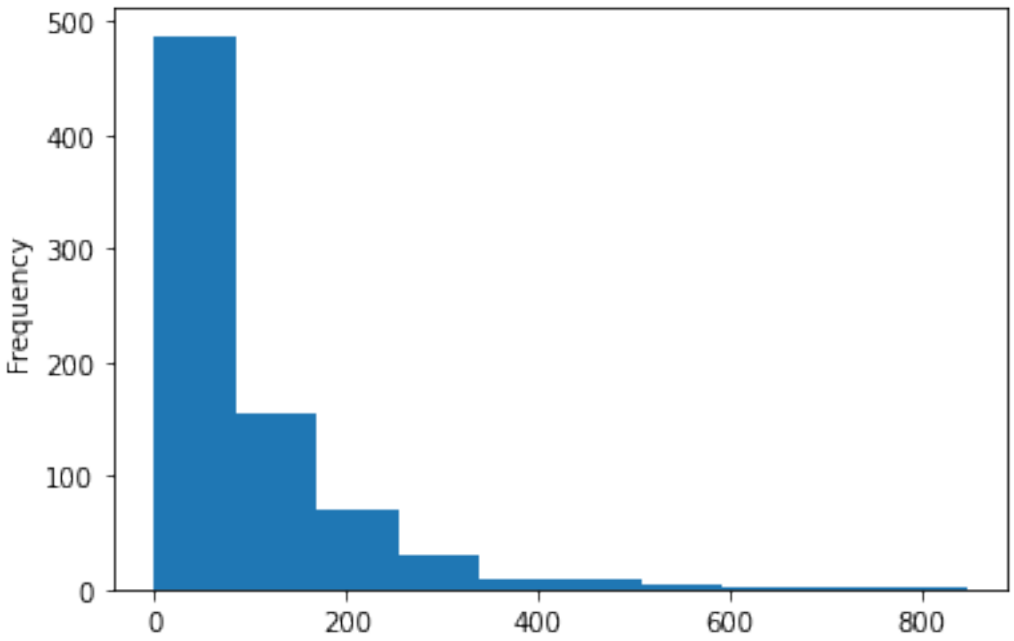

```
diabetes_data["SkinThickness"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```




```
diabetes_data["Insulin"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



Checking for null values

```
diabetes_data.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
```

No null values found, but zero values are present, checking for how many values are zero

```
diabetes_data.all()
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
```

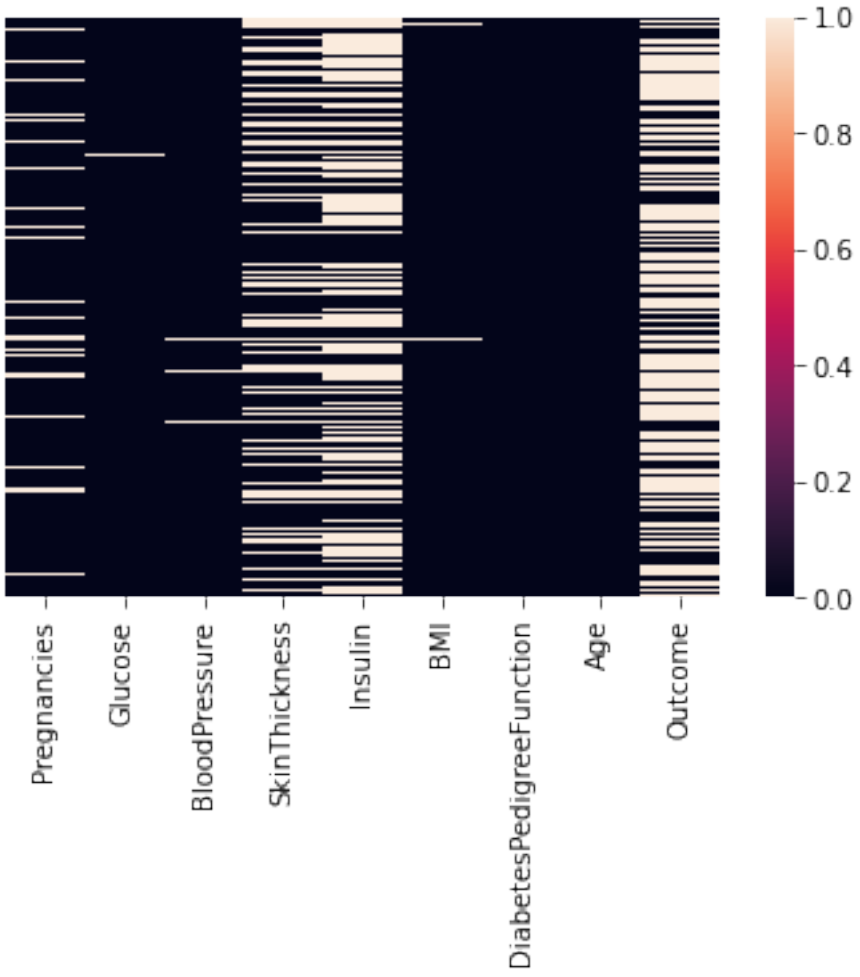
```
BMI                         False
DiabetesPedigreeFunction     True
Age                          True
Outcome                     False
dtype: bool
```

Cleaning the dataset, I found multiple zero values which could affect the acccuracy of the model

To deal with this problem, I dropped a column and replaced other zero values with the mean of the values in that column

```
sns.heatmap(diabetes_data == 0, yticklabels=False)
```

```
<AxesSubplot:>
```



Heat map shows the amount of zero values, due to how many zero values found in the thickness column i will be dropping it

I do not do the same for insulin because it is known to directly affect diabetes

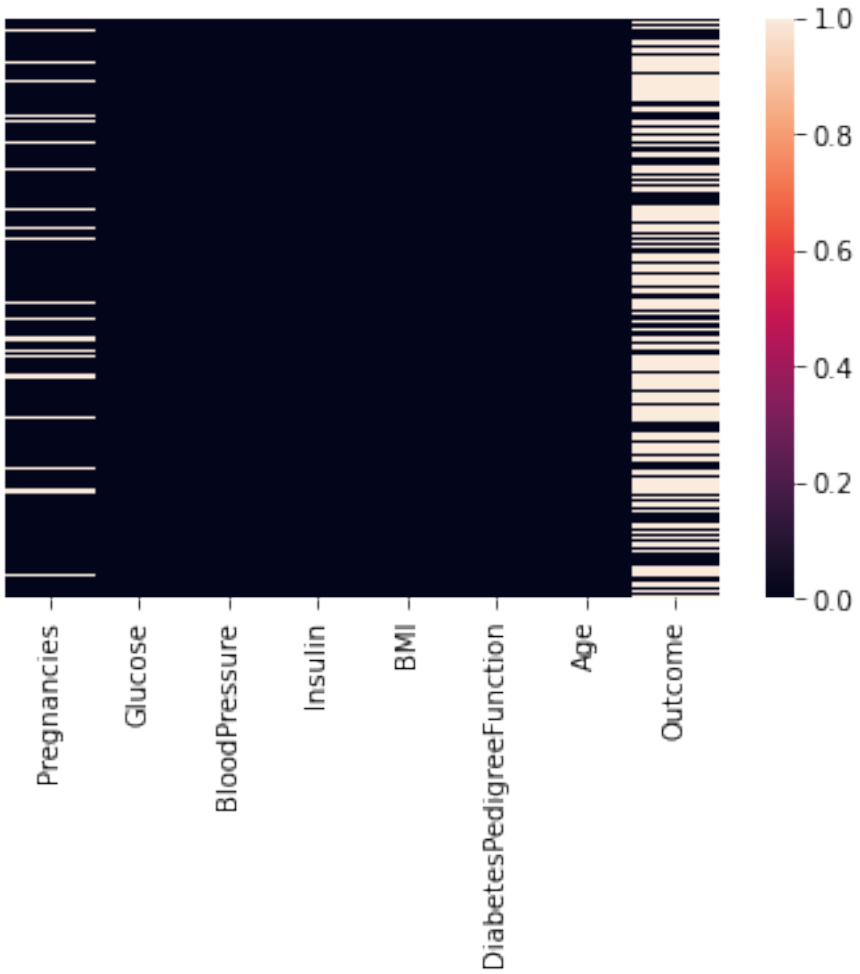
```
diabetes_data.drop('SkinThickness', axis=1, inplace=True)

diabetes_data.head(9)
```

```
   Pregnancies  Glucose  BloodPressure  Insulin   BMI  \
0            6      148             72        0  33.6
1            1       85             66        0  26.6
2            8      183             64        0  23.3
3            1       89             66       94  28.1
4            0      137             40      168  43.1
5            5      116             74        0  25.6
6            3       78             50       88  31.0
7           10      115              0        0  35.3
8            2      197             70      543  30.5

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
```

Below I calculate mean of the columns, I replace the zero values with the mean of their columns

```
avg_bmi = diabetes_data['BMI'].mean()

avg_glucose = diabetes_data['Glucose'].mean()

avg_bp = diabetes_data['BloodPressure'].mean()

avg_insulin = diabetes_data['Insulin'].mean()

diabetes_data['BMI'].replace(to_replace = 0, value = avg_bmi,
inplace=True)

diabetes_data['Glucose'].replace(to_replace = 0, value = avg_glucose,
inplace=True)

diabetes_data['BloodPressure'].replace(to_replace = 0, value = avg_bp,
inplace=True)

diabetes_data['Insulin'].replace(to_replace = 0, value = avg_insulin,
inplace=True)

sns.heatmap(diabetes_data == 0, yticklabels=False)
```

```
<AxesSubplot:>
```




```
diabetes_data.head(9)
```

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	\
0	6	148.0	72.000000	79.799479	33.6	
1	1	85.0	66.000000	79.799479	26.6	
2	8	183.0	64.000000	79.799479	23.3	
3	1	89.0	66.000000	94.000000	28.1	
4	0	137.0	40.000000	168.000000	43.1	
5	5	116.0	74.000000	79.799479	25.6	
6	3	78.0	50.000000	88.000000	31.0	
7	10	115.0	69.105469	79.799479	35.3	
8	2	197.0	70.000000	543.000000	30.5	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0

```
6                       0.248   26         1
7                       0.134   29         0
8                       0.158   53         1
```

Split te data into trainig and testing dataset

Splting the dataset

```
X=diabetes_data.drop(['Outcome'],axis=1)
y=diabetes_data['Outcome']

X[0:9]
```

```
   Pregnancies  Glucose  BloodPressure     Insulin   BMI  \
0            6    148.0      72.000000   79.799479  33.6
1            1     85.0      66.000000   79.799479  26.6
2            8    183.0      64.000000   79.799479  23.3
3            1     89.0      66.000000   94.000000  28.1
4            0    137.0      40.000000  168.000000  43.1
5            5    116.0      74.000000   79.799479  25.6
6            3     78.0      50.000000   88.000000  31.0
7           10    115.0      69.105469   79.799479  35.3
8            2    197.0      70.000000  543.000000  30.5

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
5                     0.201   30
6                     0.248   26
7                     0.134   29
8                     0.158   53
```

```
y[0:9]
```

```
0    1
1    0
2    1
3    0
4    1
5    0
6    1
7    0
8    1
Name: Outcome, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.3,random_state=0)

X_train.shape

(537, 7)

X_test.shape

(231, 7)

y_train.shape

(537,)

y_test.shape

(231,)
```

Scaling the data using a pipeline

```
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
pipe = make_pipeline(StandardScaler(), logmodel)
pipe.fit(X_train, y_train)  # apply the pipeline on training data

Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])

pipe.score(X_test, y_test) # apply the pipeline on the testing data

0.7705627705627706

#pred=pipe.predict(X_test) #make predictions on the testing data
```

Using a grid search to find the best hyperparameters and fitting it to the svm

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [0.01, 0.1, 1, 10, 100],
             #'gamma': ['scale', 'auto'],
             'kernel': ['linear','rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
```

```
#fitting the model for grid search
grid.fit(X_train, y_train)
```

```
"from sklearn.model_selection import GridSearchCV \nfrom sklearn.svm
import SVC\n# defining parameter range \nparam_grid = {'C': [0.1, 1,
10, 100, 1000],  \n              'gamma': [0.01, 0.1, 1, 10, 100],\n
#'gamma': ['scale', 'auto'], \n              'kernel':
['linear','rbf']}  \n  \ngrid = GridSearchCV(SVC(), param_grid, refit
= True, verbose = 3) \n  \n#fitting the model for grid search \
ngrid.fit(X_train, y_train) "
```

```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
'# print best parameter after tuning \nprint(grid.best_params_) \n  \
n# print how our model looks after hyper-parameter tuning \
nprint(grid.best_estimator_) '
```

Fit the dataset to an SVM using the hyperparameters

From the grid search the best hyperparameters include linear kernel, a C values of 1 and a gamma of 0.01

```
from sklearn.svm import SVC # Support Vector Classifier
models = SVC(kernel='linear', gamma =0.01 ,C=1)
models.fit(X_train, y_train)
```

```
SVC(C=1, gamma=0.01, kernel='linear')
```

Function to generate a classificatin report and confusion matrix for both the training anmd testing dataset

I generate both reports to compare the results i get from both the testing amd training datasets

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
def print_result(models, X_train, y_train, X_test, y_test,
train=True):
    if train:
        prediction = pipe.predict(X_train)
        models_report = pd.DataFrame(classification_report(y_train,
prediction, output_dict=True))
        print("Train Result:\
n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, prediction) *
100:.2f}%")
        print("_____________________________________________________")
```

```
        print(f"CLASSIFICATION REPORT:\n{models_report}")
        print("_____________________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train,
prediction)}\n")

    elif train==False:
        prediction = pipe.predict(X_test)
        models_report = pd.DataFrame(classification_report(y_test,
prediction, output_dict=True))
        print("Test Result:\
n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, prediction) *
100:.2f}%")
        print("_____________________________________________________")
        print(f"CLASSIFICATION REPORT:\n{models_report}")
        print("_____________________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test,
prediction)}\n")
```

Fit the dataset to the SVC using a linear classifier

Running a cross validatiuon on the dataset

```
from sklearn.model_selection import cross_val_score

# do cross validation on the training data
scores = cross_val_score(pipe, X_train, y_train, scoring = 'accuracy',
cv = 5)

# print the scores across 10 folds
print('Accuracies: ', scores)

# print the average score
print('Average accuracy: ', np.mean(scores))

# do cross validation on the test data
scores_test = cross_val_score(pipe, X_test, y_test, scoring =
'accuracy', cv = 5)

print('Test accuracies: ', scores_test)

print('Test average accuracy: ', np.mean(scores_test))
```

```
Accuracies:  [0.80555556 0.76851852 0.76635514 0.71962617 0.77570093]
Average accuracy:  0.7671512634129456
Test accuracies:  [0.87234043 0.76086957 0.73913043 0.80434783
0.69565217]
Test average accuracy:  0.774468085106383
```

```
from sklearn.model_selection import cross_val_score

# do cross validation on the training data
scores = cross_val_score(pipe, X_train, y_train, scoring =
'precision', cv = 5)

# print the scores across 10 folds
print('Precisions: ', scores)

# print the average score
print('Average precison: ', np.mean(scores))

# do cross validation on the test data
scores_test = cross_val_score(pipe, X_test, y_test, scoring =
'precision', cv = 5)

print('Test precisions: ', scores_test)

print('Test average precision: ', np.mean(scores_test))
```

```
Precisions:  [0.8        0.69444444 0.73333333 0.63636364 0.75      ]
Average precison:  0.7228282828282828
Test precisions:  [0.8        0.64285714 0.61538462 0.8        0.5
]
Test average precision:  0.6716483516483518
```

Running the function

```
print_result(models, X_train, y_train, X_test, y_test, train=True)
print_result(models, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 77.65%

________________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.794195    0.734177  0.776536    0.764186      0.772513
recall       0.877551    0.597938  0.776536    0.737745      0.776536
f1-score     0.833795    0.659091  0.776536    0.746443      0.770680
support    343.000000  194.000000  0.776536  537.000000    537.000000
________________________________________________
Confusion Matrix:
 [[301  42]
 [ 78 116]]

Test Result:
================================================
Accuracy Score: 77.06%

________________________________________________
```

```
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.795455   0.690909  0.770563    0.743182      0.761964
recall       0.891720   0.513514  0.770563    0.702617      0.770563
f1-score     0.840841   0.589147  0.770563    0.714994      0.760212
support    157.000000  74.000000  0.770563  231.000000    231.000000
_____________________________________________________
Confusion Matrix:
 [[140  17]
 [ 36  38]]
```

Using an ensembles on the same dataset

- Analyze the data set, clean and remove any null or zero values
- Split the data into training and testing dataset
- Use a grid search to find suitatble hyperparameters
- Fit data to model, run a cross validation and calculate accuraccy

```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns #for statistics ploting
import matplotlib.pyplot as plt
import math
import sklearn
%matplotlib inline
```

Displaying the dataset and analysing it

```
diabetes_data = pd.read_csv('diabetes.csv')

diabetes_data.head(9)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1
```

```
5              5      116             74              0        0  25.6

6              3       78             50             32       88  31.0

7             10      115              0              0        0  35.3

8              2      197             70             45      543  30.5


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
```

```
diabetes_data.tail(9)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
759            6      190             92              0        0  35.5

760            2       88             58             26       16  28.4

761            9      170             74             31        0  44.0

762            9       89             62              0        0  22.5

763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4


     DiabetesPedigreeFunction  Age  Outcome
759                     0.278   66        1
760                     0.766   22        0
761                     0.403   43        1
762                     0.142   33        0
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
```

```
766                      0.349   47         1
767                      0.315   23         0
```

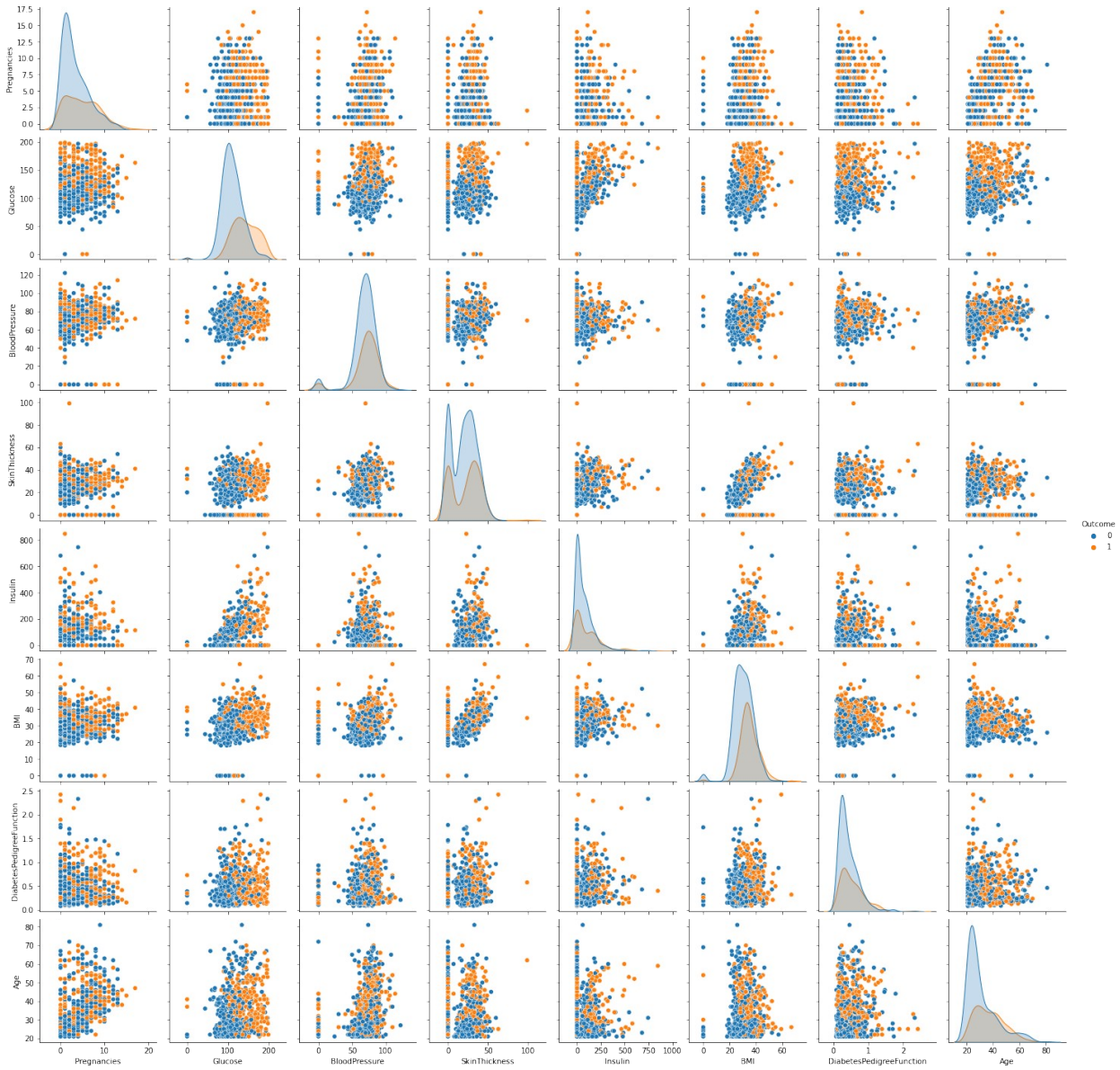
```
diabetes_data.index
```

```
RangeIndex(start=0, stop=768, step=1)
```

Using various plots to see the data i am woorking with

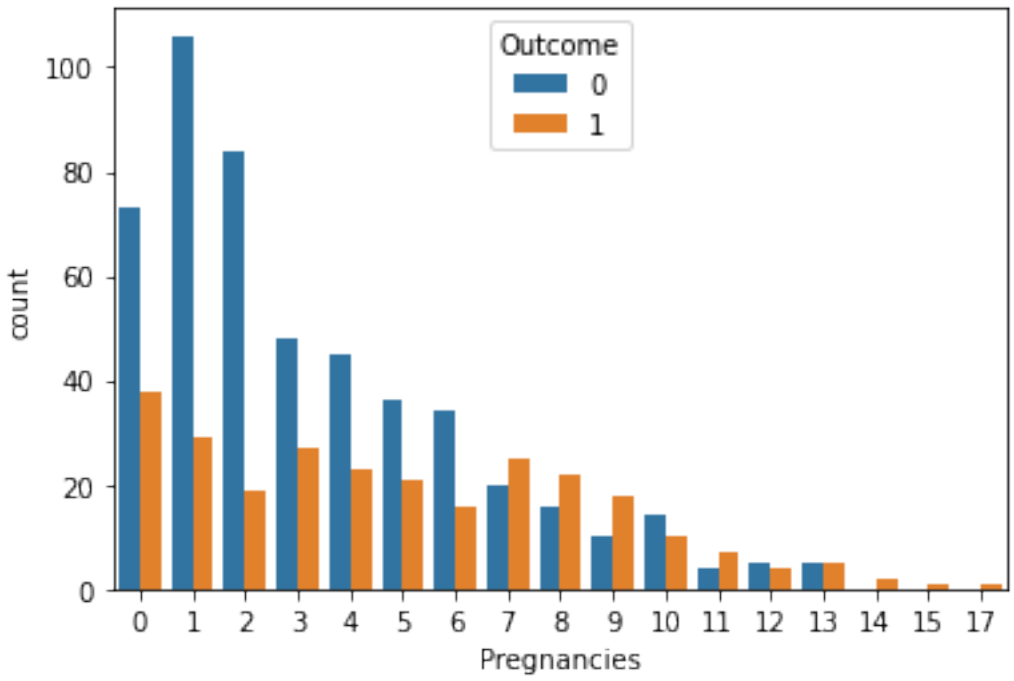
```
sns.pairplot(diabetes_data, hue='Outcome', vars=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness',
                                    'Insulin', 'BMI',
'DiabetesPedigreeFunction',
                                    'Age'])
```

```
<seaborn.axisgrid.PairGrid at 0x7f51600f4f40>
```



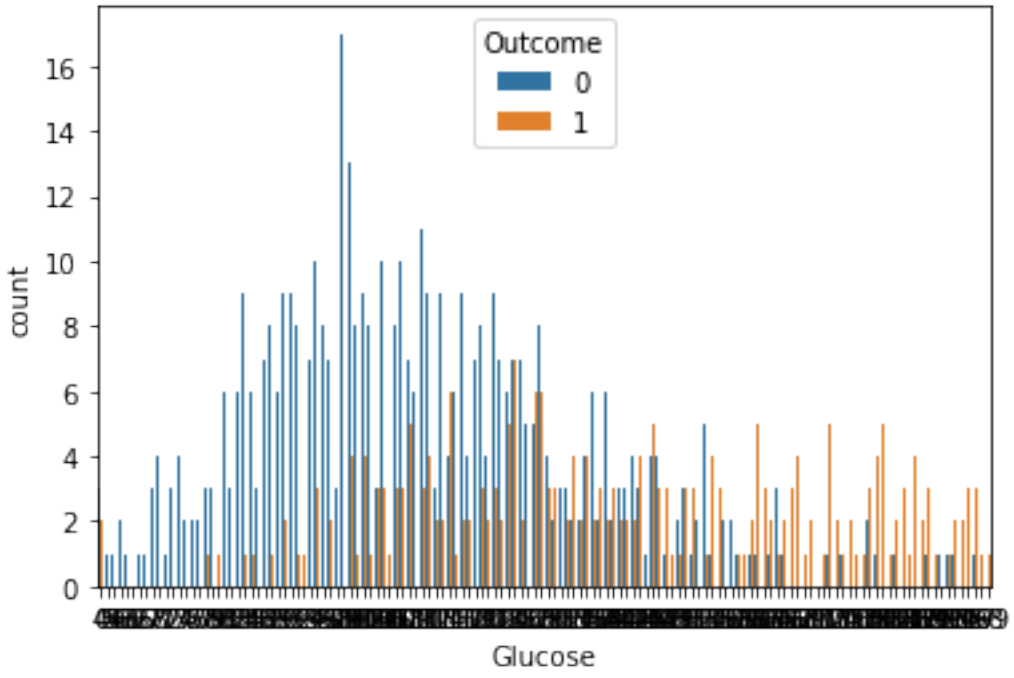

```
sns.countplot(x="Pregnancies", hue="Outcome", data=diabetes_data)
```

```
<AxesSubplot:xlabel='Pregnancies', ylabel='count'>
```



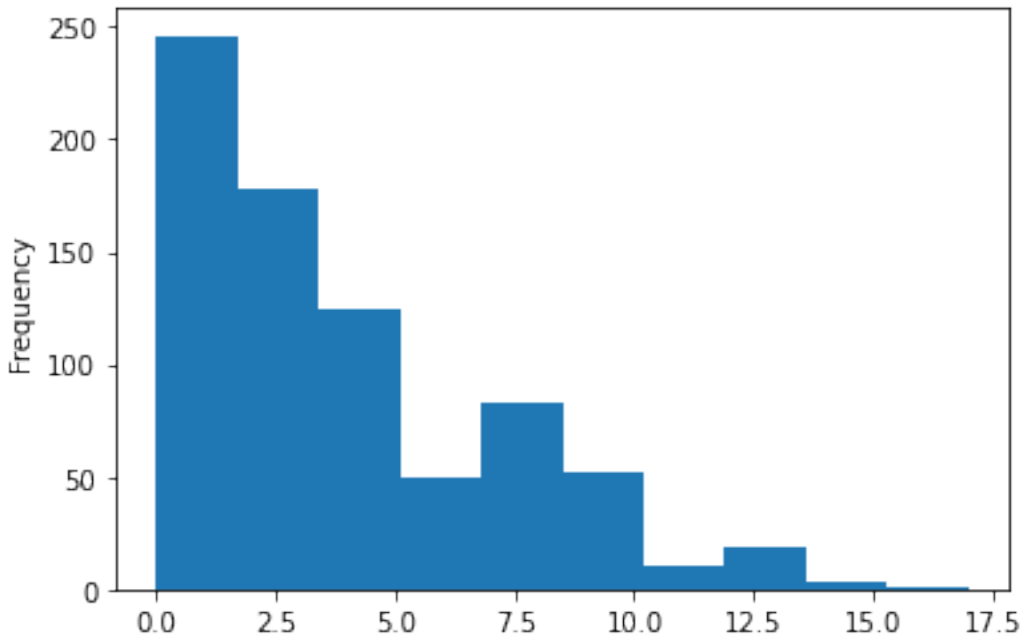

```
sns.countplot(x="Glucose", hue="Outcome", data=diabetes_data)
```

```
<AxesSubplot:xlabel='Glucose', ylabel='count'>
```



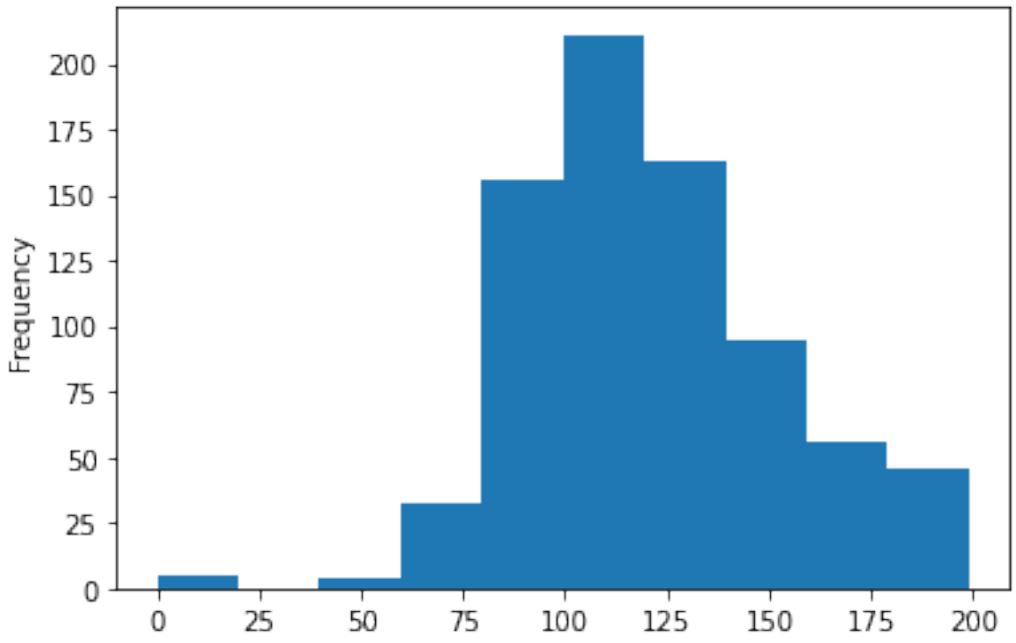

```
diabetes_data["Pregnancies"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



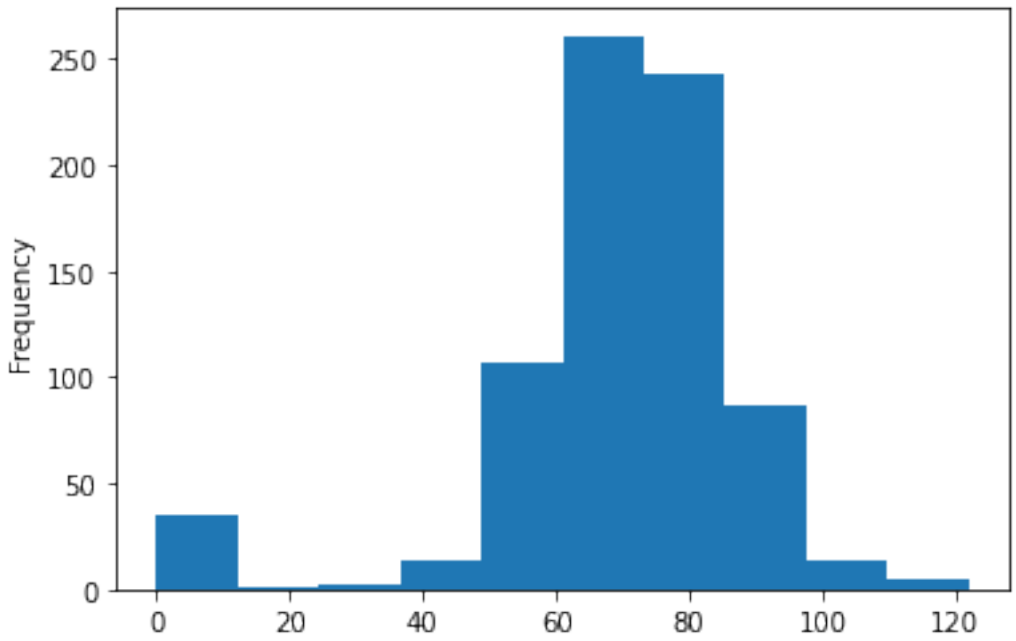

```
diabetes_data["Glucose"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



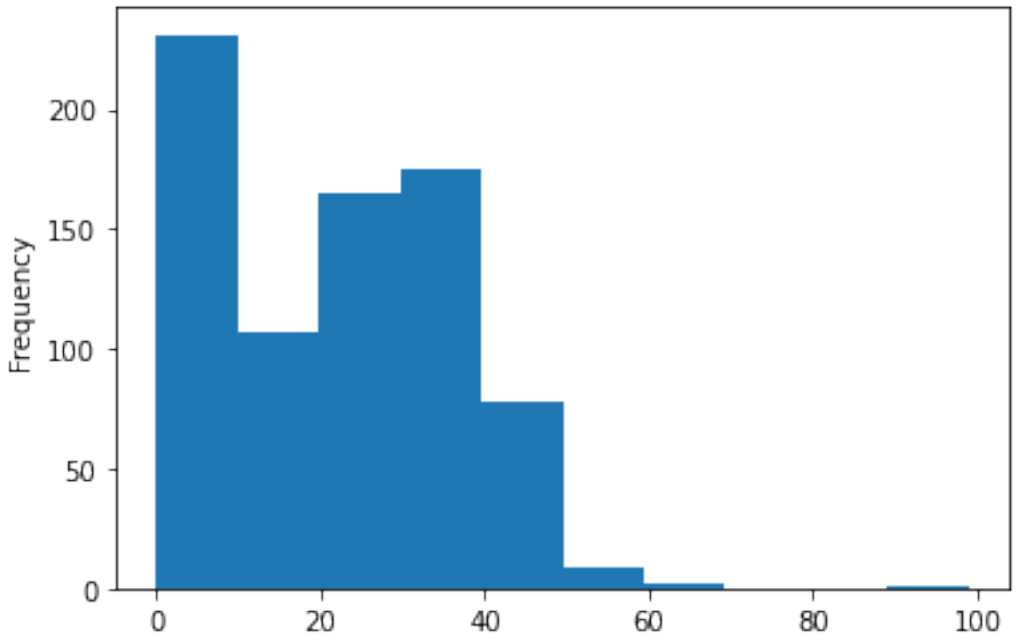

```
diabetes_data["BloodPressure"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



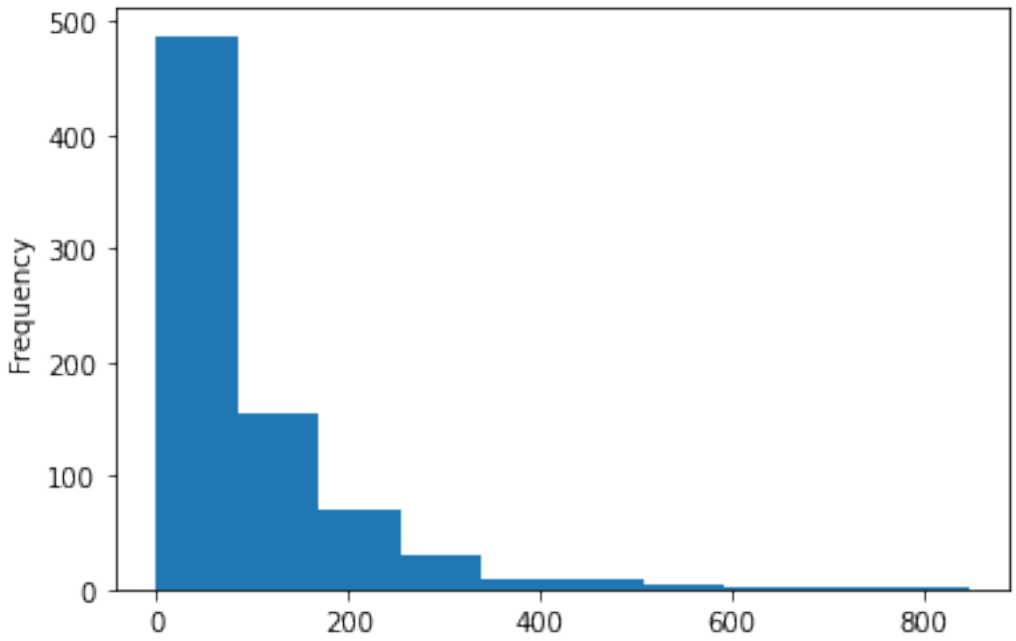

```
diabetes_data["SkinThickness"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



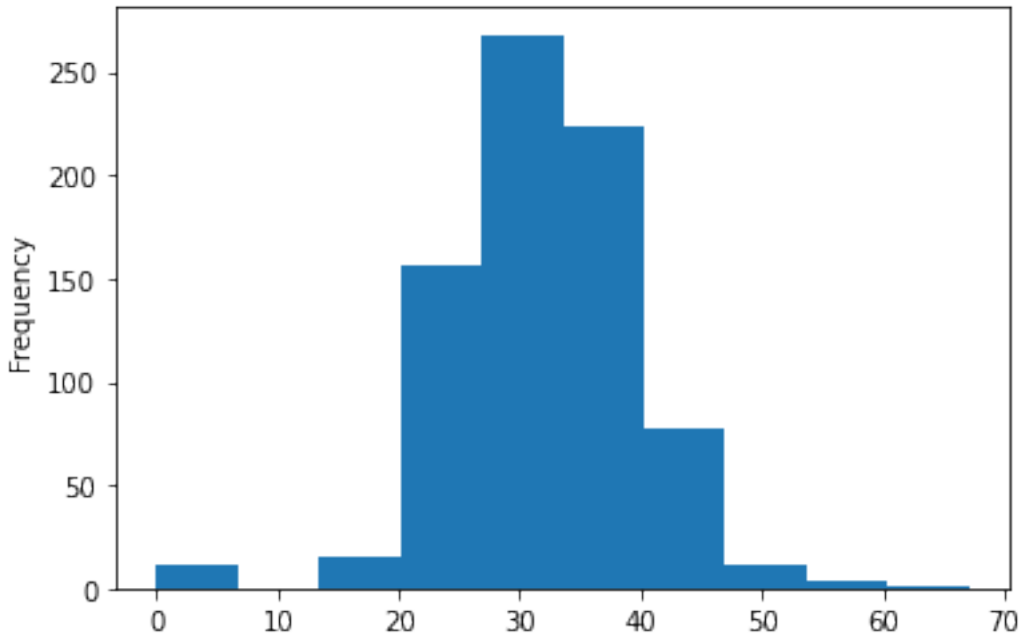

```
diabetes_data["Insulin"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



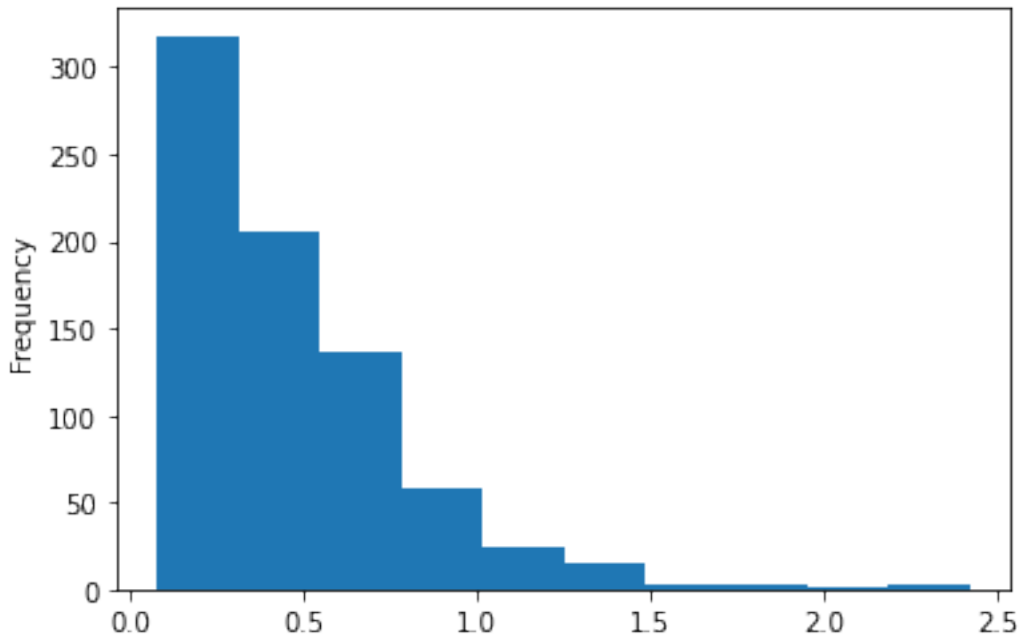

```
diabetes_data["BMI"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



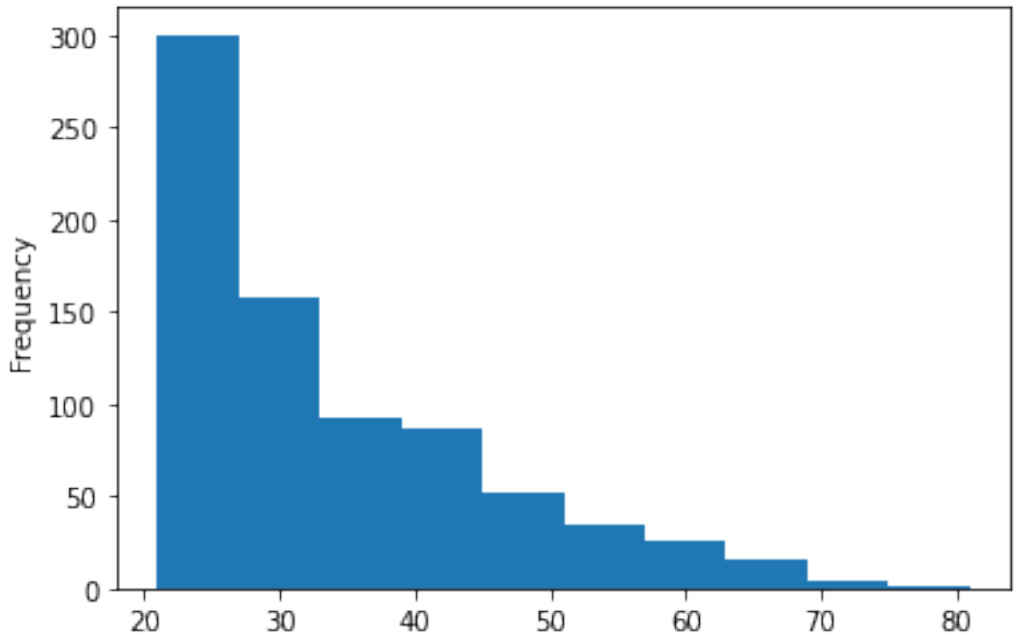

```
diabetes_data["DiabetesPedigreeFunction"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```




```
diabetes_data["Age"].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



Checking for null values in the dataset

```
diabetes_data.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
```

Processing the data and cleaning the data

```
diabetes_data.isnull()
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0          False    False          False          False    False
False
1          False    False          False          False    False
False
2          False    False          False          False    False
False
3          False    False          False          False    False
False
4          False    False          False          False    False
False
..           ...      ...            ...            ...      ...    ..
.
763        False    False          False          False    False
False
764        False    False          False          False    False
False
765        False    False          False          False    False
False
766        False    False          False          False    False
False
767        False    False          False          False    False
False

     DiabetesPedigreeFunction    Age  Outcome
0                       False  False    False
1                       False  False    False
```

```
2                       False  False    False
3                       False  False    False
4                       False  False    False
..                        ...    ...      ...
763                     False  False    False
764                     False  False    False
765                     False  False    False
766                     False  False    False
767                     False  False    False

[768 rows x 9 columns]
```

No null values found

```
diabetes_data.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

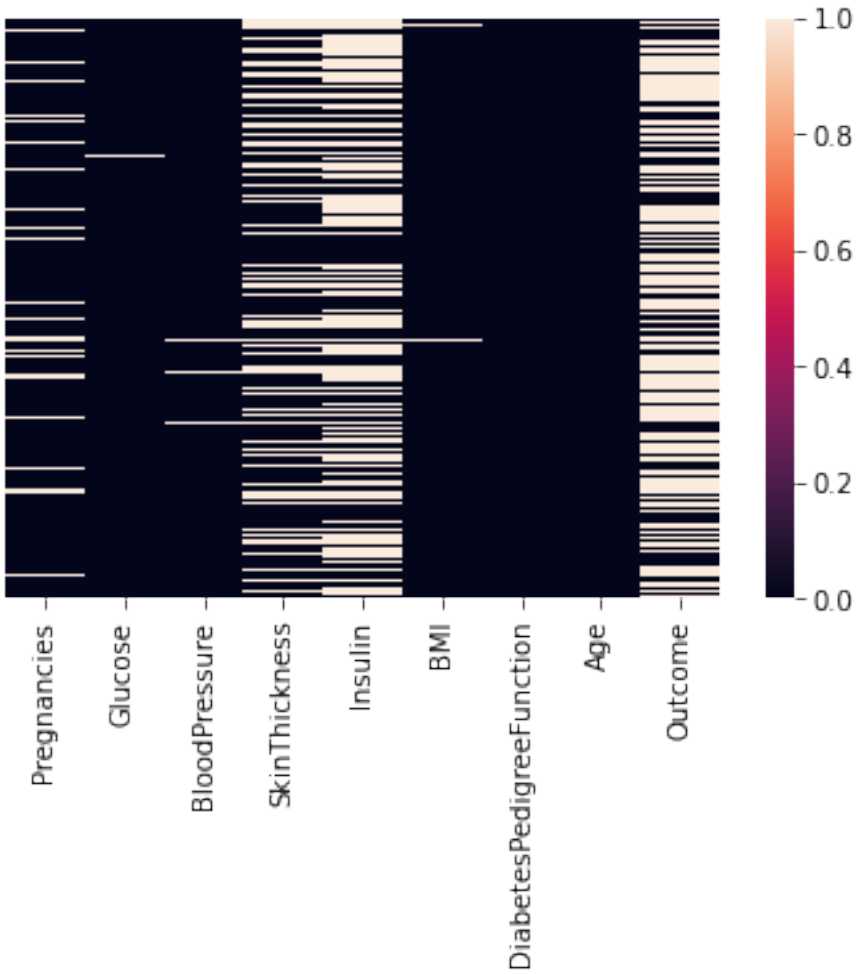
Zero values found present in the dataset

```
diabetes_data.all()
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction     True
Age                          True
Outcome                     False
dtype: bool
```

```
sns.heatmap(diabetes_data == 0, yticklabels=False)
```

```
<AxesSubplot:>
```



Found alot of zero values in skinthickness and insulin, decided to drop the skinthickness due to the amount of zero values

Did not drop insulin because it directly affects diabetes

```
diabetes_data.drop('SkinThickness', axis=1, inplace=True)

diabetes_data.head(5)
```

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	\
0	6	148	72	0	33.6	
1	1	85	66	0	26.6	
2	8	183	64	0	23.3	
3	1	89	66	94	28.1	
4	0	137	40	168	43.1	

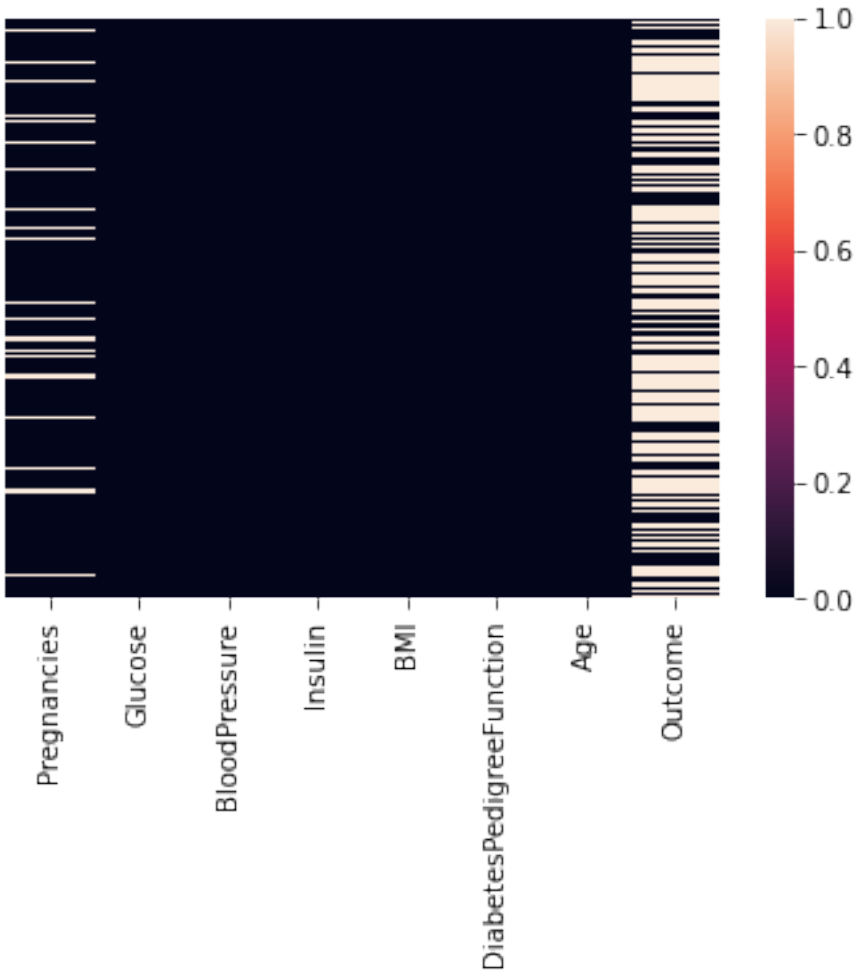
	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1

```
3                       0.167   21         0
4                       2.288   33         1
```

Calculate the mean of the columns and replace the zero values the mean value

```
avg_bmi = diabetes_data['BMI'].mean()

avg_glucose = diabetes_data['Glucose'].mean()

avg_insulin = diabetes_data['Insulin'].mean()

avg_bp = diabetes_data['BloodPressure'].mean()

diabetes_data['BMI'].replace(to_replace = 0, value = avg_bmi,
inplace=True)

diabetes_data['Glucose'].replace(to_replace = 0, value = avg_glucose,
inplace=True)

diabetes_data['Insulin'].replace(to_replace = 0, value = avg_insulin,
inplace=True)

diabetes_data['BloodPressure'].replace(to_replace = 0, value = avg_bp,
inplace=True)

sns.heatmap(diabetes_data == 0, yticklabels=False)
```

```
<AxesSubplot:>
```




```
diabetes_data.head(9)
```

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	\
0	6	148.0	72.000000	79.799479	33.6	
1	1	85.0	66.000000	79.799479	26.6	
2	8	183.0	64.000000	79.799479	23.3	
3	1	89.0	66.000000	94.000000	28.1	
4	0	137.0	40.000000	168.000000	43.1	
5	5	116.0	74.000000	79.799479	25.6	
6	3	78.0	50.000000	88.000000	31.0	
7	10	115.0	69.105469	79.799479	35.3	
8	2	197.0	70.000000	543.000000	30.5	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0

```
6                       0.248   26        1
7                       0.134   29        0
8                       0.158   53        1
```

```
diabetes_data.all()
```

```
Pregnancies                 False
Glucose                      True
BloodPressure                True
Insulin                      True
BMI                          True
DiabetesPedigreeFunction     True
Age                          True
Outcome                     False
dtype: bool
```

Splitting the data into training and testing datasets

```
X=diabetes_data.drop(['Outcome'],axis=1)
y=diabetes_data['Outcome']
```

```
X[0:9]
```

```
   Pregnancies  Glucose  BloodPressure     Insulin   BMI  \
0            6    148.0      72.000000   79.799479  33.6
1            1     85.0      66.000000   79.799479  26.6
2            8    183.0      64.000000   79.799479  23.3
3            1     89.0      66.000000   94.000000  28.1
4            0    137.0      40.000000  168.000000  43.1
5            5    116.0      74.000000   79.799479  25.6
6            3     78.0      50.000000   88.000000  31.0
7           10    115.0      69.105469   79.799479  35.3
8            2    197.0      70.000000  543.000000  30.5

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
5                     0.201   30
6                     0.248   26
7                     0.134   29
8                     0.158   53
```

```
y[0:9]
```

```
0    1
1    0
2    1
```

```
3    0
4    1
5    0
6    1
7    0
8    1
Name: Outcome, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.3,random_state=1)

from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
```

Scaling the dataset using a pipeline

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pipe = make_pipeline(StandardScaler(), logmodel)
pipe.fit(X_train, y_train)  # apply the pipeline on training data

Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])

pipe.score(X_test, y_test) # apply the pipeline on the testing data

0.7792207792207793

#preds =pipe.predict(X_test)
```

Grid search to find the best hyperparameters for the model

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
# Create the parameter grid based on the results of random search
param_grid = {
    'max_features': ['sqrt', 'log2'],
    'n_estimators': [10,100, 200, 300, 1000]}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 5, verbose = 3)
grid.fit(X_train, y_train)

"from sklearn.model_selection import GridSearchCV\nfrom
sklearn.ensemble import RandomForestRegressor\nfrom sklearn.tree
```

```
import DecisionTreeClassifier\n# Create the parameter grid based on
the results of random search \nparam_grid = {\n    #'bootstrap':
[True],\n    #'max_depth': [80, 90, 100, 110],\n    'max_features':
['sqrt', 'log2'],\n    #'min_samples_leaf': [3, 4, 5],\n
#'min_samples_split': [8, 10, 12],\n    'n_estimators': [10,100, 200,
300, 1000],\n    \n}\n# Create a based model\nrf =
RandomForestRegressor()\n# Instantiate the grid search model\ngrid =
GridSearchCV(estimator = rf, param_grid = param_grid, \n
cv = 5, verbose = 3)\ngrid.fit(X_train, y_train) "

# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)

'# print best parameter after tuning \nprint(grid.best_params_) \n  \
n# print how our model looks after hyper-parameter tuning \
nprint(grid.best_estimator_) '
```

Fitting the dataset to the random forest model

The hyperparameters were chosen using a grid search

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, max_features =
'log2', random_state=0)
model.fit(X_train, y_train)

RandomForestClassifier(max_features='log2', random_state=0)

#len(preds)
```

Function to generate the accuracy, classification report and confusion matrix of the testing and training dataset

Geenrating reports for both testing and traing to compare the results

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

def print_score(model, X_train, y_train, X_test, y_test, train=True):
    if train:
        preds = pipe.predict(X_train)
        model_report = pd.DataFrame(classification_report(y_train,
preds, output_dict=True))
        print("Train Result:\
n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, preds) *
100:.2f}%")
```

```
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{model_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train,
preds)}\n")

    elif train==False:
        preds = pipe.predict(X_test)
        model_report = pd.DataFrame(classification_report(y_test,
preds, output_dict=True))
        print("Test Result:\
n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, preds) *
100:.2f}%")
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{model_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test,
preds)}\n")
```

Running a cross validation on the dataset

```
from sklearn.model_selection import cross_val_score

# do cross validation on the training data
scores = cross_val_score(pipe, X_train, y_train, scoring = 'accuracy',
cv = 5)

# print the scores across 10 folds
print('Accuracies: ', scores)

# print the average score
print('Average accuracy: ', np.mean(scores))

# do cross validation on the test data
scores_test = cross_val_score(pipe, X_test, y_test, scoring =
'accuracy', cv = 5)

print('Test accuracies: ', scores_test)

print('Test average accuracy: ', np.mean(scores_test))
```

```
Accuracies:  [0.72222222 0.80555556 0.82242991 0.72897196 0.77570093]
Average accuracy:  0.770976116303219
Test accuracies:  [0.76595745 0.73913043 0.7826087  0.76086957
0.80434783]
Test average accuracy:  0.7705827937095282
```

```
from sklearn.model_selection import cross_val_score
```

```
# do cross validation on the training data
scores = cross_val_score(pipe, X_train, y_train, scoring =
'precision', cv = 5)

# print the scores across 10 folds
print('Precisions: ', scores)

# print the average score
print('Average precison: ', np.mean(scores))

# do cross validation on the test data
scores_test = cross_val_score(pipe, X_test, y_test, scoring =
'precision', cv = 5)

print('Test precisions: ', scores_test)

print('Test average precision: ', np.mean(scores_test))
```

```
Precisions:  [0.60606061 0.80769231 0.78125    0.62962963 0.73076923]
Average precison:  0.7110803548303549
Test precisions:  [0.71428571 0.66666667 0.76923077 0.6875     0.9
]
Test average precision:  0.74753663003663
```

Printing out the result of the function

```
print_score(model, X_train, y_train, X_test, y_test, train=True)
print_score(model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 77.65%

_______________________________________________
CLASSIFICATION REPORT:
                     0           1  accuracy   macro avg  weighted avg
precision     0.796954    0.720280  0.776536    0.758617      0.770825
recall        0.887006    0.562842  0.776536    0.724924      0.776536
f1-score      0.839572    0.631902  0.776536    0.735737      0.768802
support     354.000000  183.000000  0.776536  537.000000    537.000000

_______________________________________________
Confusion Matrix:
 [[314  40]
 [ 80 103]]

Test Result:
================================================
Accuracy Score: 77.92%

_______________________________________________
CLASSIFICATION REPORT:
                     0           1  accuracy   macro avg  weighted avg
```

```
precision    0.784431   0.765625  0.779221    0.775028      0.777511
recall       0.897260   0.576471  0.779221    0.736865      0.779221
f1-score     0.837061   0.657718  0.779221    0.747389      0.771069
support    146.000000  85.000000  0.779221  231.000000    231.000000
_________________________________________________
Confusion Matrix:
 [[131  15]
 [ 36  49]]
```

Comparison of both models

For both models I used the same method, the only difference being the models, I did this make a proper comparison between the two models

I chose random forest classifier because the algorithm applies bagging

For the SVM model I noticed that the accuracy score of the training data is higher than the accuracy of the tesing data

For the Ensemble model, the accuracy score of the tesing dataset is higher than the accuracy score of the training dataset

Running a gridsearch on the parametsers for the SVM took a longer time than the ensemble

The training time for a SVM is significantly longer compared to the random forest claasifier

Based on the accracy scores from both models, random forest classifier produces a better accuracy compared to the SVM

The acuracy scores are better for both the testing and training dataset using the random forest classifier