In [26]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

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

In [6]:

```
data = pd.read_csv("diabetes.csv")
```

In [7]:

```
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
```

In [8]:

```
data.head()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [9]:

```
data.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [10]:

```
data.isnull().sum()
```

Out[10]:

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

In [11]:

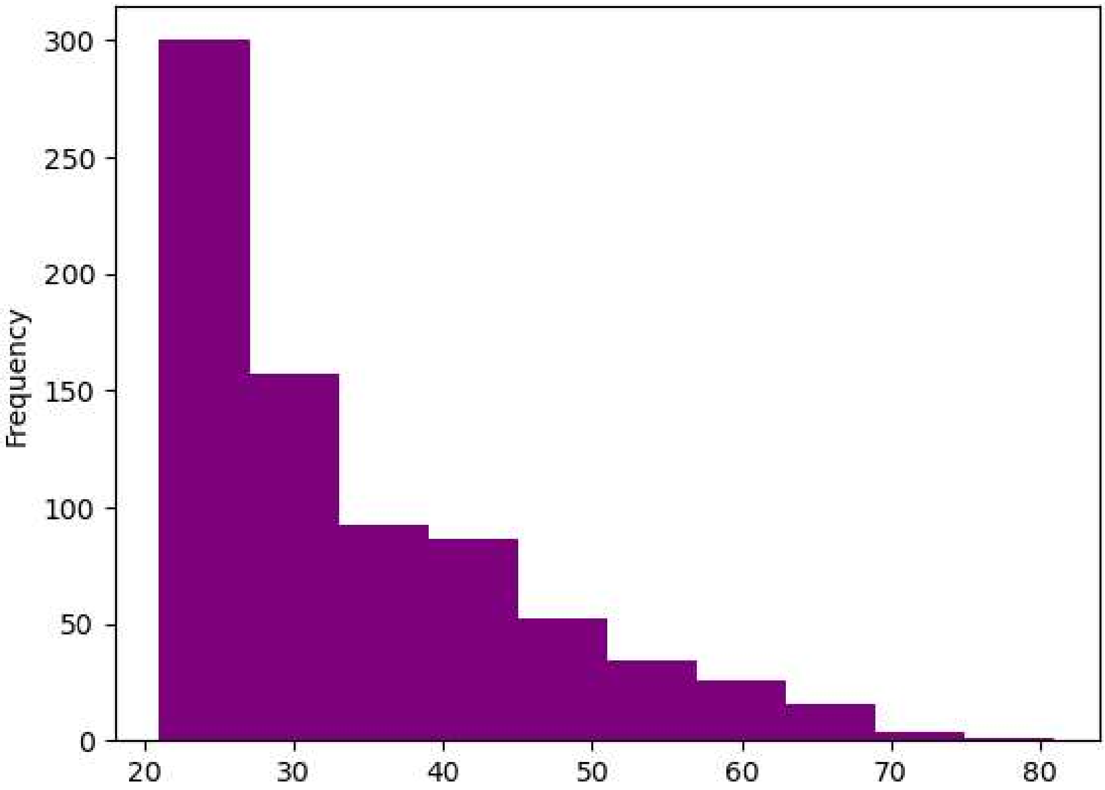
```
data.dtypes
```

Out[11]:

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

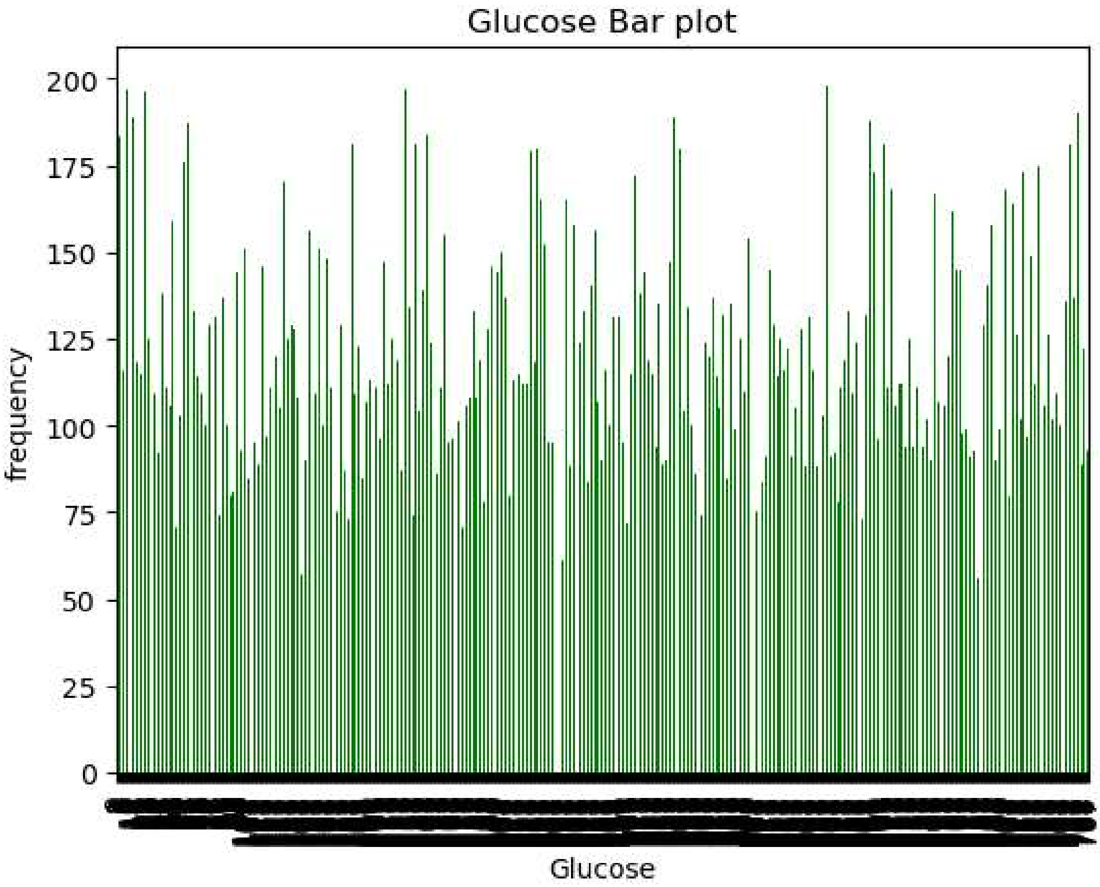
In [12]:

```
data.Age.plot(color="purple",kind="hist")
plt.show()
```



In [14]:

```
data.Glucose.plot(color="green",kind="bar")
plt.xlabel("Glucose")
plt.ylabel("frequency")
plt.title("Glucose Bar plot")
plt.show()
```



In [15]:

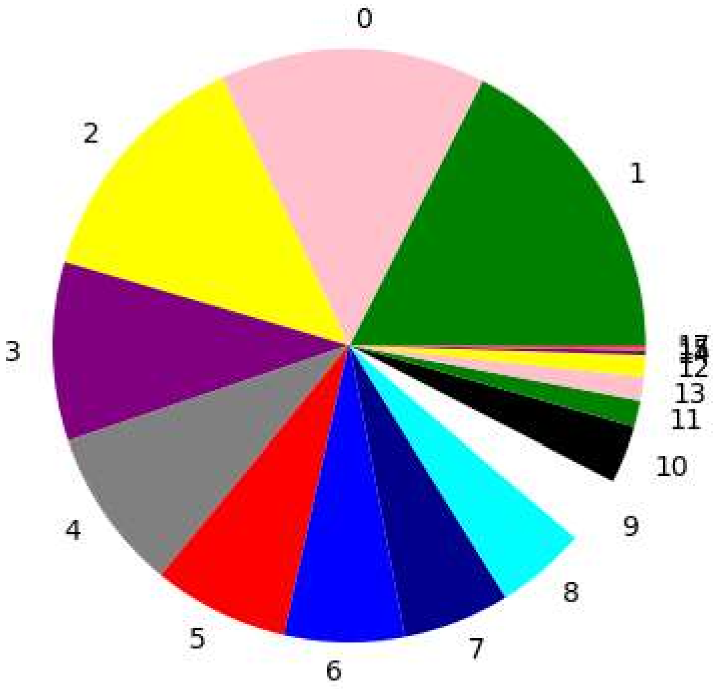
```
data.Pregnancies.value_counts()
```

Out[15]:

```
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: Pregnancies, dtype: int64
```

In [16]:

```
sizes=data.Pregnancies.value_counts().values
labels=data.Pregnancies.value_counts().index
colors=["green","pink","yellow","purple","grey","red","blue","darkblue","c
plt.pie(sizes,data=data,labels=labels,colors=colors)
plt.show()
```



In [17]:

```
corr_matrix = data.corr()
corr_matrix['Outcome'].sort_values(ascending=False)
```

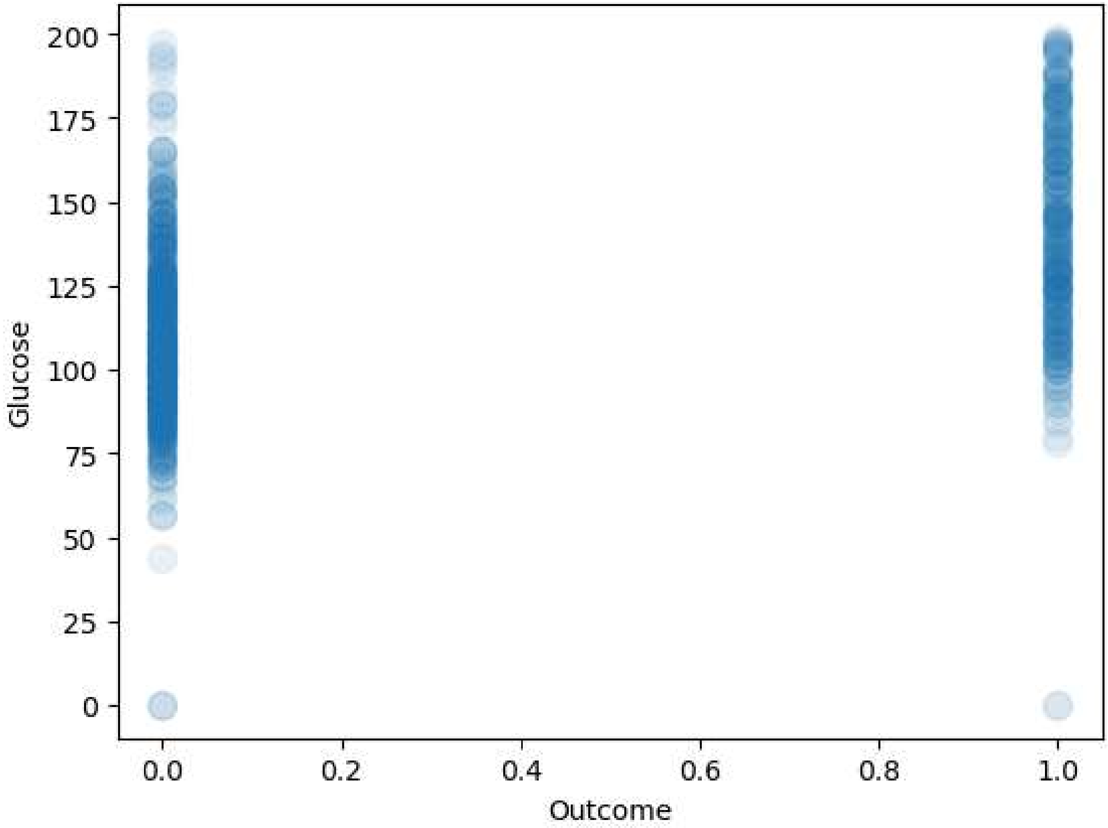
Out[17]:

```
Outcome                     1.000000
Glucose                     0.466581
BMI                         0.292695
Age                         0.238356
Pregnancies                 0.221898
DiabetesPedigreeFunction    0.173844
Insulin                     0.130548
SkinThickness               0.074752
BloodPressure               0.065068
Name: Outcome, dtype: float64
```

In [18]:

```
data.plot(kind = 'scatter', x = 'Outcome',y = 'Glucose',s = 100 ,alpha = 0

plt.show()
```



In [23]:

```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(data, test_size=0.1, random_state=42)


train_df_labels = train_df["Outcome"].copy()
train_df= train_df.drop("Outcome", axis=1)
```

In [24]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler                        # T

num_pipeline = Pipeline([('std_scaler', StandardScaler()), ])

train_prepared = num_pipeline.fit_transform(train_df)
```

In [27]:
```
#Logistic Regression
model = LogisticRegression()
model.fit(train_prepared, train_df_labels)
```

Out[27]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(model, train_prepared, train_df_labels, cv= 3, scoring='ac
```

Out[28]:
```
array([0.77922078, 0.77391304, 0.72173913])
```

In [29]:
```
prediction = model.predict(train_prepared)
print("LR Accuracy of Classifier: ", model.score(train_prepared, train_df_
```

```
LR Accuracy of Classifier:  0.7756874095513748
```

In [30]:
```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

poly_kernel_svm_clf = Pipeline([ ("scaler", StandardScaler()),
                                ("svm_clf", SVC(kernel="poly", degree=3, c
])

poly_kernel_svm_clf.fit(train_prepared, train_df_labels)
```

Out[30]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('svm_clf', SVC(C=5, coef0=1, kernel='poly'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
print("SVM Accuracy of Classifier: ", poly_kernel_svm_clf.score(train_prep
```

```
SVM Accuracy of Classifier:  0.8451519536903039
```

In [32]:
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(poly_kernel_svm_clf, train_prepared, trai
```

In [33]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(train_df_labels, y_train_pred)
```

Out[33]:
```
array([[375,  75],
       [109, 132]], dtype=int64)
```

In [34]:

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

print('Precision Score:',precision_score(train_df_labels, y_train_pred))
print('Recall Score:',recall_score(train_df_labels, y_train_pred))
print('F1 Score:',f1_score(train_df_labels, y_train_pred))
```

```
Precision Score: 0.6376811594202898
Recall Score: 0.5477178423236515
F1 Score: 0.5892857142857143
```

In [44]:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(train_df_labels, y_train_pred)
```

Out[44]:

```
0.6905255878284924
```

In [35]:

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(train_prepared, train_df_labels)

y_probas_forest = cross_val_predict(forest_clf, train_prepared, train_df_l
```

In [36]:

```
prediction = forest_clf.predict(train_prepared)
print("Random Forest Classifire Accuracy of Classifier: ", model.score(tra
```

```
Random Forest Classifire Accuracy of Classifier:  0.7756874095513748
```

In [37]:

```
cross_val_score(forest_clf, train_prepared, train_df_labels, cv= 3, scorin
```

Out[37]:

```
array([0.78787879, 0.79565217, 0.73043478])
```

In [40]:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(train_df_labels, y_train_pred)
```

In [41]:

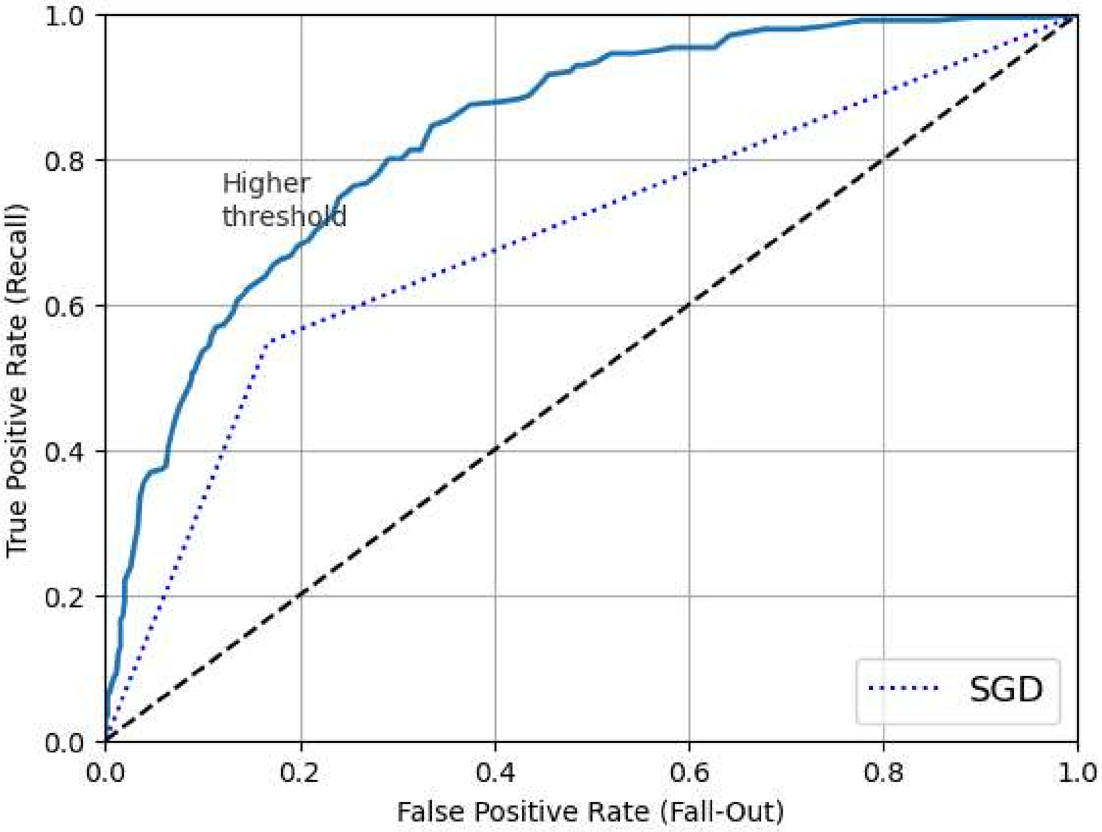
```
y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(train_df_labels, y_s
```

In [42]:

```
plt.plot(fpr_forest, tpr_forest, linewidth=2, label=None)

plt.plot(fpr, tpr, "b:", label="SGD")
plt.plot([0, 1], [0, 1], 'k--')
plt.text(0.12, 0.71, "Higher\nthreshold", color="#333333")
plt.xlabel('False Positive Rate (Fall-Out)')
plt.ylabel('True Positive Rate (Recall)')
plt.grid()
plt.axis([0, 1, 0, 1])
plt.legend(loc="lower right", fontsize=13)


plt.show()
```



In [45]:

```
roc_auc_score(train_df_labels, y_train_pred)
```

Out[45]:

```
0.6905255878284924
```

In [ ]: