In [7]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca
import matplotlib.pyplot as plt
```

In [9]:

```
data = pd.read_csv("C:/Users/KJCOEMR/Downloads/diabetes/diabetes.csv")
print(data.head())
```

```
   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

   Pedigree  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
```

In [13]:

```
data.shape
```

Out[13]:

```
(768, 9)
```

In [15]:

```
data.dtypes
```

Out[15]:

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

In [17]:

```
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   Pedigree       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 [21]:

```
data.describe()
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2

In [23]:

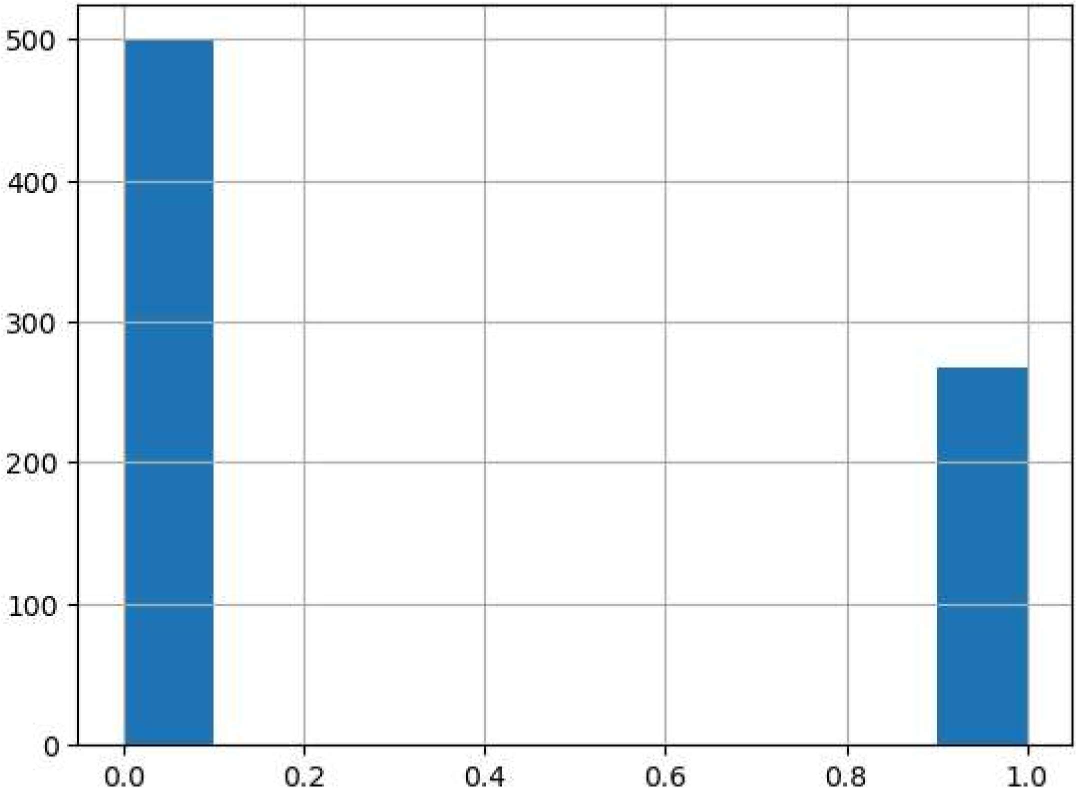
```
data.isna().sum()
```

Out[23]:

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

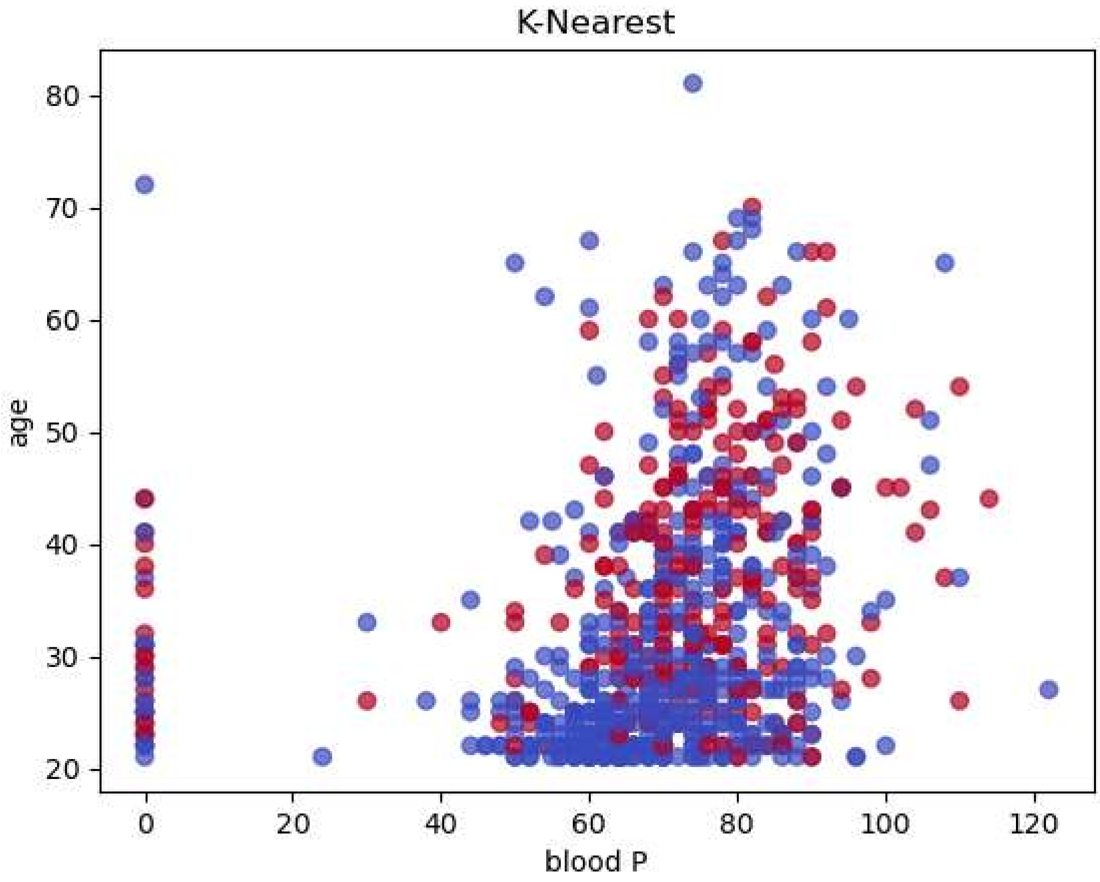
In [25]:

```
data['Outcome'].hist()
plt.show()
```



In [27]:

```
plt.scatter(data['BloodPressure'],data['Age'], c=data['Outcome'], cmap='coolwarm',a
plt.title("K-Nearest")
plt.xlabel("blood P")
plt.ylabel("age")
plt.show()
```



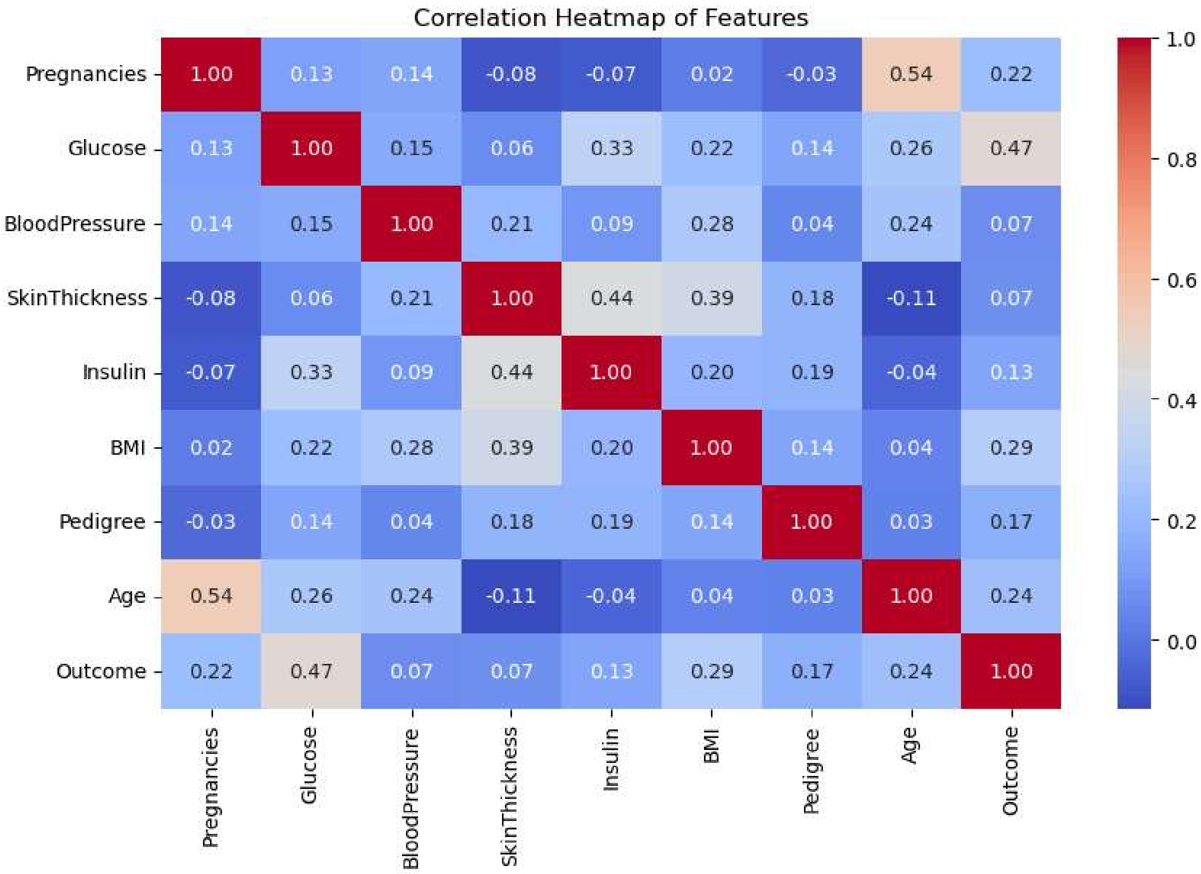
In [31]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# calculate correlation matrix
corr_matrix = data.corr()

# plot heatmap
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation Heatmap of Features")
plt.show()
```



In [41]:

```
# Set the Independent Parameters(x) and the Target Variable(y)
X = data.drop("Outcome",axis=1)
Y = data["Outcome"]
```

In [43]:

```
from sklearn.model_selection import train_test_split
```

In [45]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta
```

In [47]:

```
# Import KNN
from sklearn.neighbors import KNeighborsClassifier
```

In [53]:

```
# Iterative/Elbow Method to find the Optimal K
accuracy_values = []
from sklearn import metrics
for i in range(1,20):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train,Y_train)
    Y_pred = model.predict(X_test)
    accuracy = metrics.accuracy_score(Y_test,Y_pred)
    accuracy_values.append(accuracy)
```

In [55]:

```
# Select the Optimal K based on the Optimal Accuracy Score
optimal_k = -1
optimal_accuracy = -1

for i in list(zip(range(1,20),accuracy_values)):
    if i[1]>optimal_accuracy:
        optimal_accuracy = i[1]
        optimal_k = i[0]
```

In [57]:

```
# Train the Model on Optimal K
knn = KNeighborsClassifier(n_neighbors = optimal_k)
```

In [59]:

```
# Fit the Model
knn.fit(X_train,Y_train)
```

Out[59]:

```
▾      KNeighborsClassifier      ⓘ ?

KNeighborsClassifier(n_neighbors=15)
```

In [63]:

```
Y_pred = knn.predict(X_test)
```

In [67]:

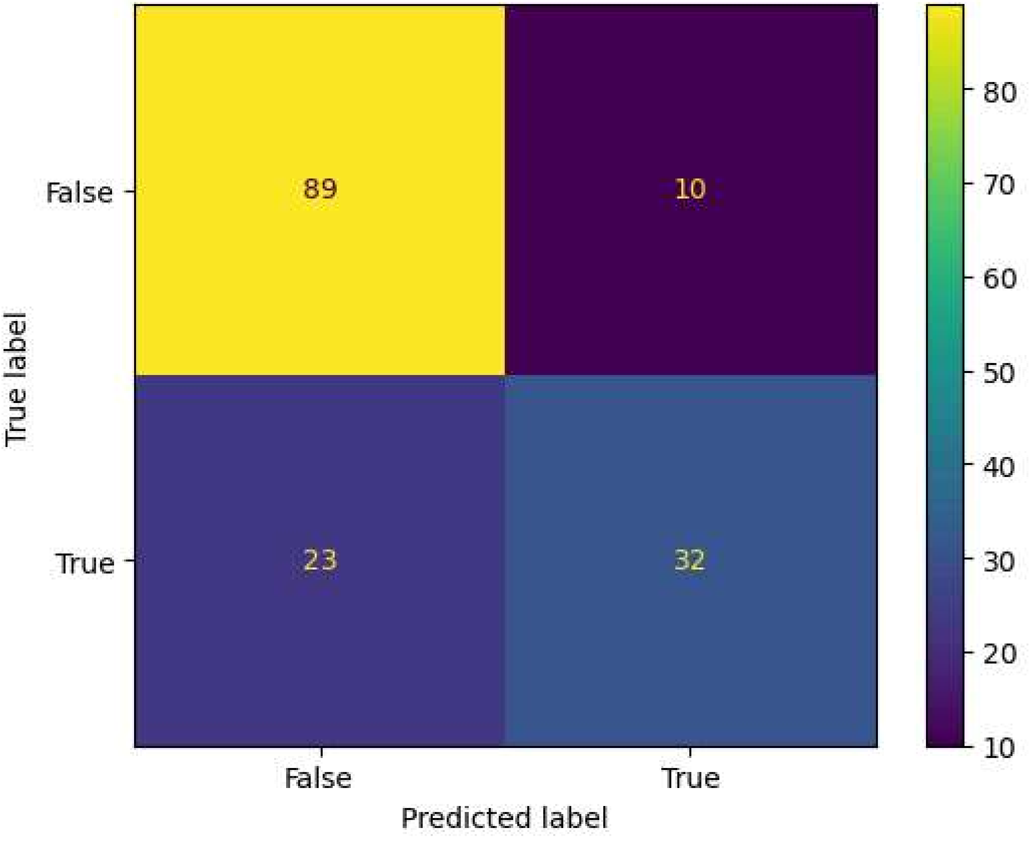
```
# Generate the Confusion Matrix
confusion_matrix = metrics.confusion_matrix(Y_test,Y_pred)
```

In [69]:

```
# Plot the Confusion Matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, disp
cm_display.plot()
```

Out[69]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17927b2bc80>
```



In [73]:

```
print(metrics.classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        99
           1       0.76      0.58      0.66        55

    accuracy                           0.79       154
   macro avg       0.78      0.74      0.75       154
weighted avg       0.78      0.79      0.78       154
```

In [79]:

```
TN = confusion_matrix[0][0]
FP = confusion_matrix[0][1]
FN = confusion_matrix[1][0]
TP = confusion_matrix[1][1]

# Recall
recall = TP / (TP + FN)

# Precision
precision = TP / (TP + FP)

# F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

# Accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)
```

In [81]:

```
print(f"Recall: {recall:.2f}")
print(f"Precision: {precision:.2f}")
print(f"F1 Score: {f1_score:.2f}")
print(f"Accuracy: {accuracy:.2f}")
```

```
Recall: 0.58
Precision: 0.76
F1 Score: 0.66
Accuracy: 0.79
```

In [85]:

```
from sklearn import metrics
mae = metrics.mean_absolute_error(Y_test, Y_pred)
print(f"Mean Absolute Error: {mae:.2f}")
```

```
Mean Absolute Error: 0.21
```

In [ ]: