- Misbah Sabir
- Batch IV
- Machine Learning

Assigment 1

Appying Linear Regression, Sipport Vector Machine and K-means Algorithm on Diabetes Dataset

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LinearRegression
```

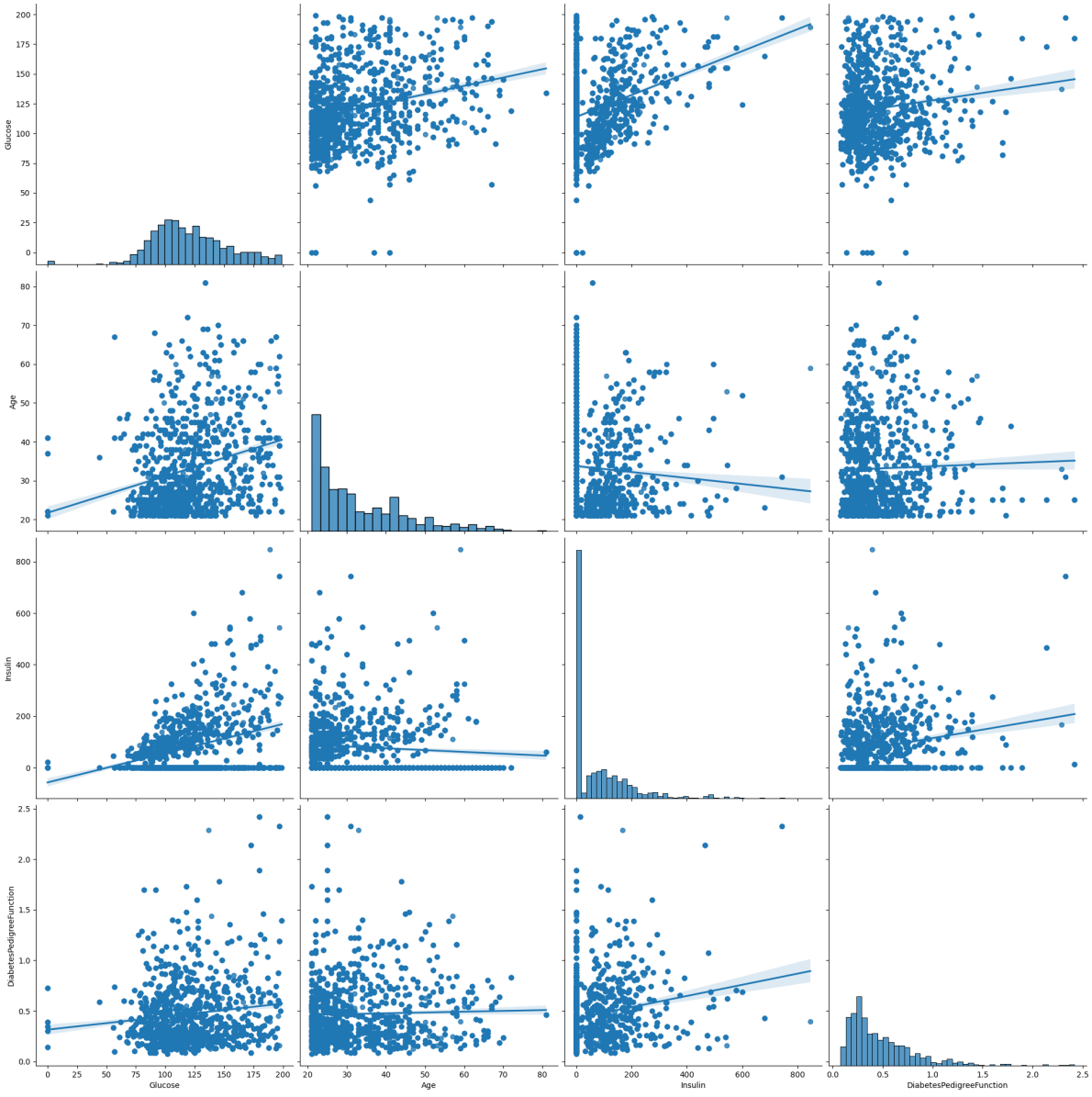
In [2]:
```
from scipy.stats import linregress
```

In [3]:
```
df = pd.read_csv("Diabetes.csv")
```

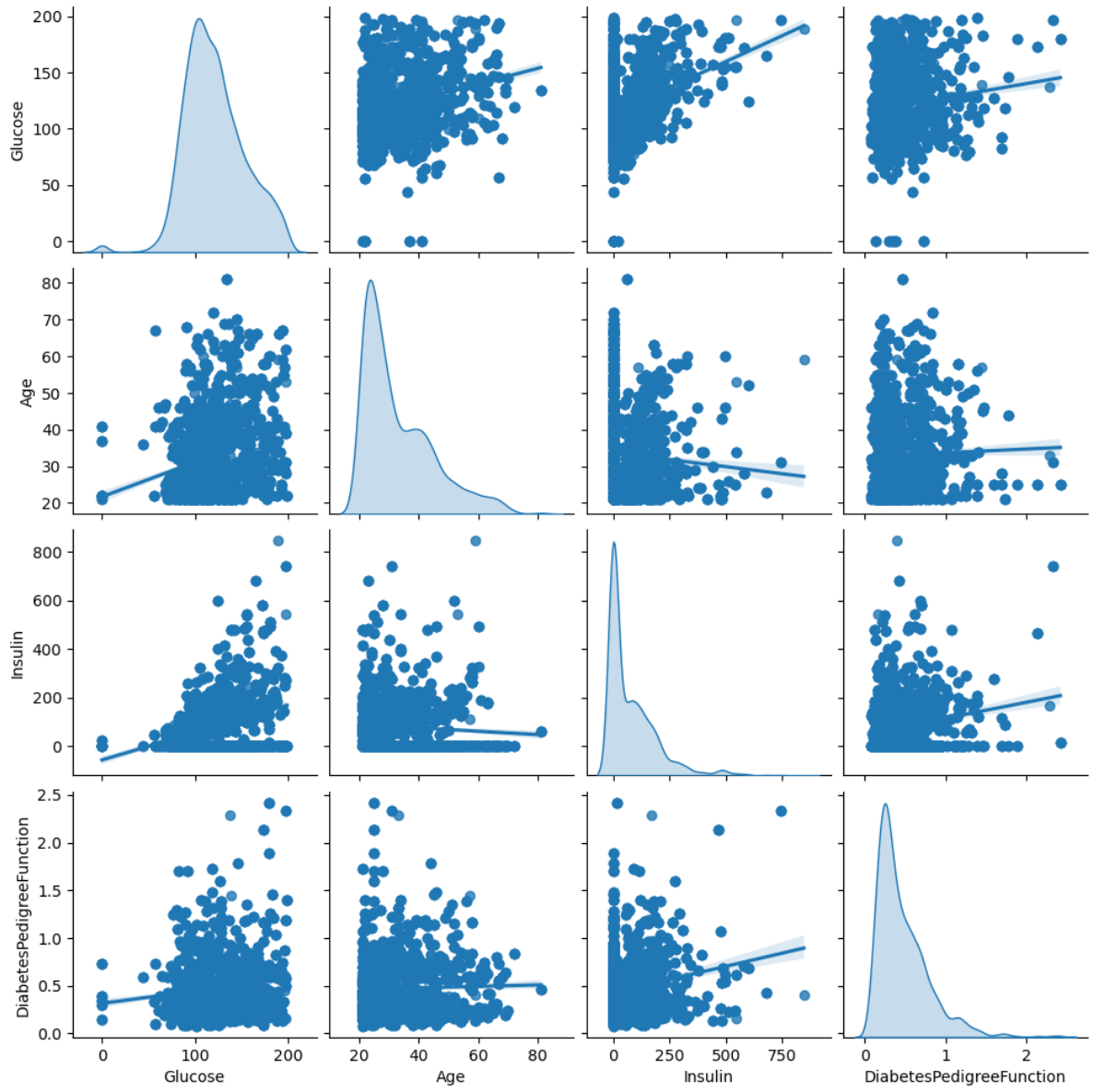
In [4]:
```
df.columns
```

Out[4]:
```
Index(['Id', 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
       'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

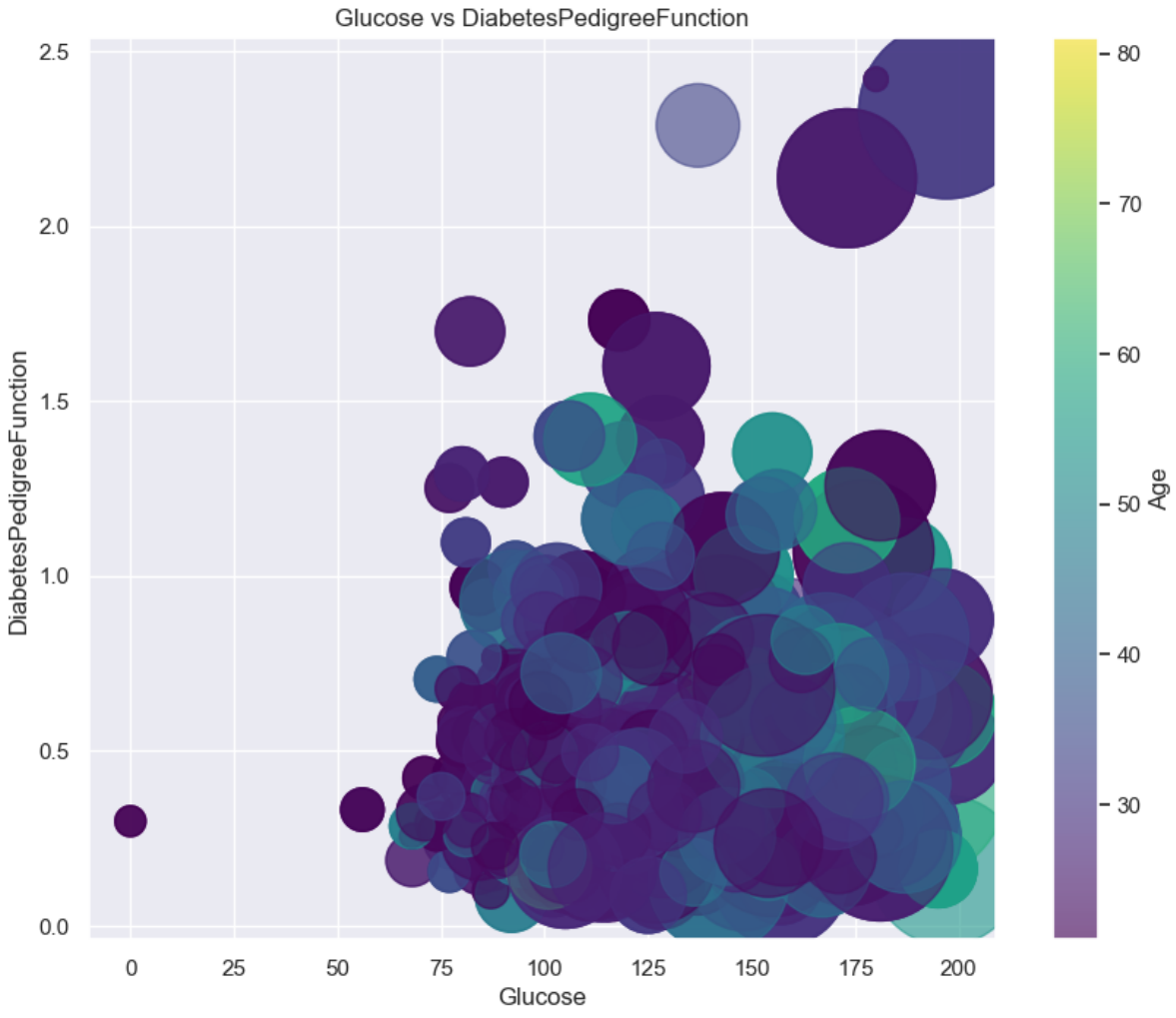
In [5]:
```
# Fit the multiple regression model
X = df[['Glucose', 'Age', 'Insulin']]
y = df['DiabetesPedigreeFunction']
model = LinearRegression()
model.fit(X, y)

# Create a scatter plot matrix with regression lines
data = pd.concat([X, y], axis=1)
sns.pairplot(data, kind='reg', height=5)
plt.show()
```



In [6]:
```
# Create a DataFrame for easy plotting
data = df[['Glucose', 'Age', 'Insulin', 'DiabetesPedigreeFunction']]

# Create a pair plot with multiple regression lines
sns.pairplot(data=data, kind='reg', diag_kind='kde')
plt.show()
```



In [7]:
```
# Assuming df contains your data with columns: Glucose, Age, Insulin, DiabetesP
sns.set(style="darkgrid")

# Create a scatter plot with color representing Age and size representing Insu
plt.figure(figsize=(10, 8))
scatter = plt.scatter(df['Glucose'], df['DiabetesPedigreeFunction'],
                      c=df['Age'], cmap='viridis',
                      s=df['Insulin']*10, alpha=0.6)  # Multiplying Insulin by

# Add colorbar and title
cbar = plt.colorbar(scatter)
cbar.set_label('Age')
plt.title('Glucose vs DiabetesPedigreeFunction')
plt.xlabel('Glucose')
plt.ylabel('DiabetesPedigreeFunction')
plt.show()
```



Linear Regression

- Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variabl

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [9]:
```
# create independent and dependent variables as features and target variable (X
X = df[['Glucose', 'Age', 'Insulin']]
y = df['DiabetesPedigreeFunction']

model = LinearRegression().fit(X, y)
```

In [10]:
```
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.10160861098798962
```

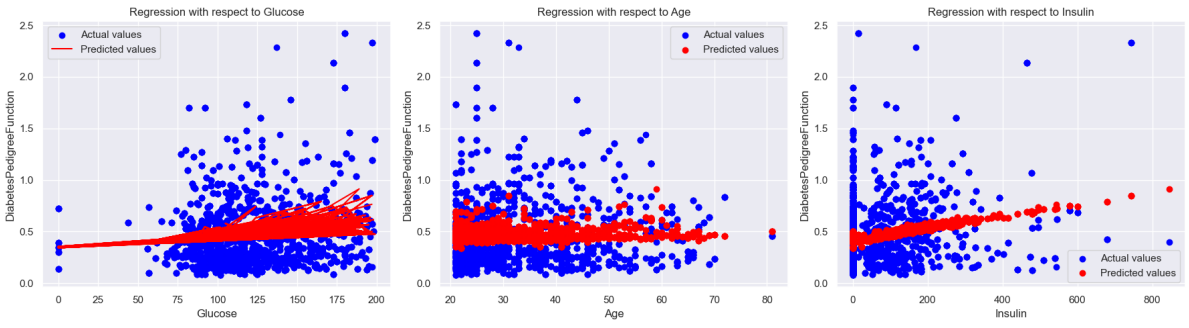
In [11]:
```
from sklearn.metrics import r2_score

rsquare = r2_score(y, y_pred)
print("Mean Squared:", rsquare)
```

```
Mean Squared: 0.04162595095685462
```

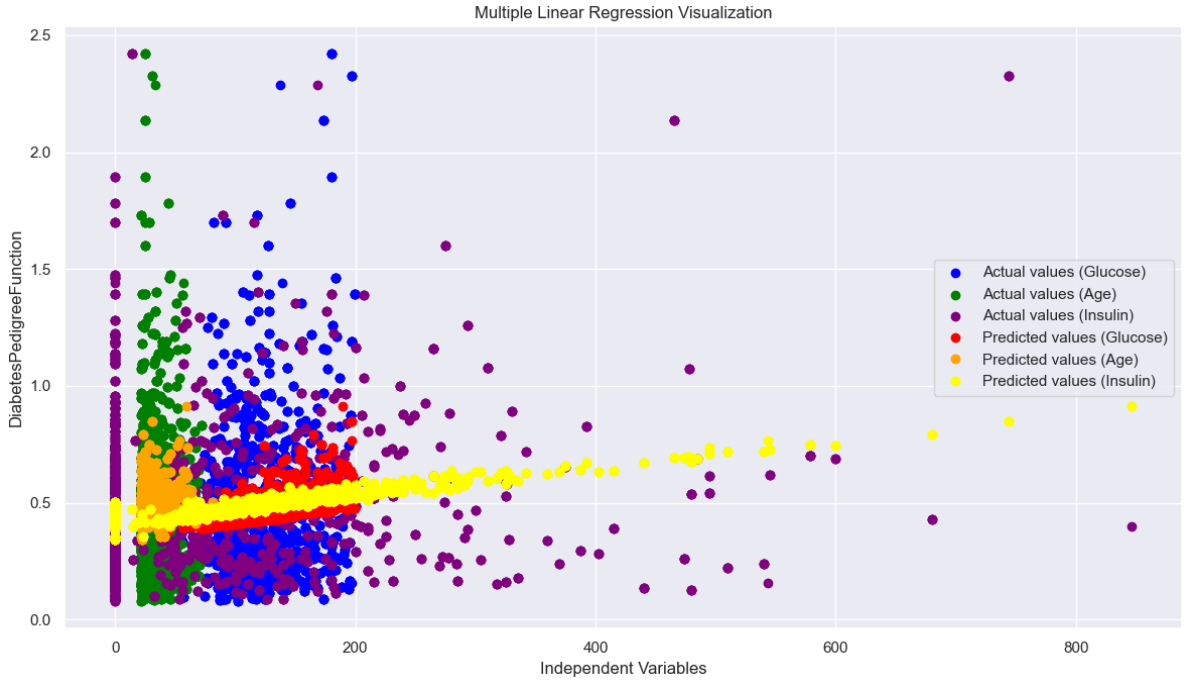
In [12]:
```
# Print the coefficients and intercept of the linear regression model
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [0.00066728 0.00067034 0.00049603]
Intercept: 0.32842791971727214
```

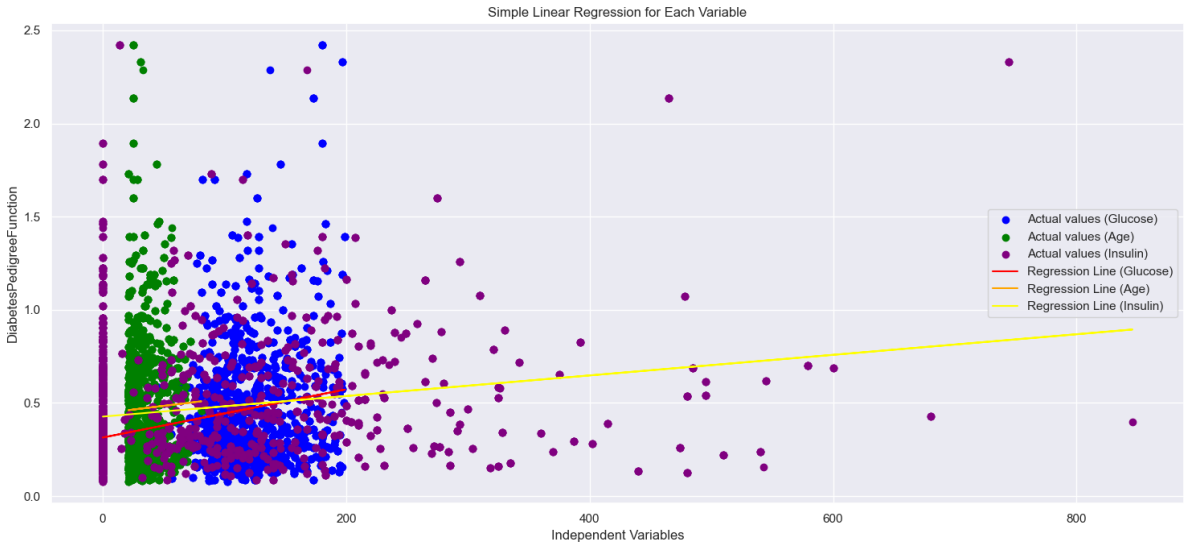
In [13]:
```
plt.figure(figsize=(18, 5))  # Adjusted the figure size to accommodate three su

# Plot for Glucose
plt.subplot(1, 3, 1)  # Changed to 1, 3, 1 for 1x3 configuration
plt.scatter(df['Glucose'], df['DiabetesPedigreeFunction'], color='blue', label=
plt.plot(df['Glucose'], y_pred, color='red', label='Predicted values')
plt.xlabel('Glucose')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Regression with respect to Glucose')
plt.legend()

# Plot for Age
plt.subplot(1, 3, 2)  # Changed to 1, 3, 2 for 1x3 configuration
plt.scatter(df['Age'], df['DiabetesPedigreeFunction'], color='blue', label='Act
plt.scatter(df['Age'], y_pred, color='red', label='Predicted values')
plt.xlabel('Age')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Regression with respect to Age')
plt.legend()

# Plot for Insulin
plt.subplot(1, 3, 3)  # Changed to 1, 3, 3 for 1x3 configuration
plt.scatter(df['Insulin'], df['DiabetesPedigreeFunction'], color='blue', label=
plt.scatter(df['Insulin'], y_pred, color='red', label='Predicted values')
plt.xlabel('Insulin')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Regression with respect to Insulin')
plt.legend()

plt.tight_layout()
plt.show()
```



In [14]:
```
plt.figure(figsize=(12, 7))

# Plotting actual values
plt.scatter(df['Glucose'], df['DiabetesPedigreeFunction'], color='blue', label=
plt.scatter(df['Age'], df['DiabetesPedigreeFunction'], color='green', label='A
plt.scatter(df['Insulin'], df['DiabetesPedigreeFunction'], color='purple', lab

# Plotting predicted values
plt.plot(df['Glucose'], y_pred, color='red', marker='o', linestyle='', label='
plt.plot(df['Age'], y_pred, color='orange', marker='o', linestyle='', label='P
plt.plot(df['Insulin'], y_pred, color='yellow', marker='o', linestyle='', labe

plt.xlabel('Independent Variables')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Multiple Linear Regression Visualization')
plt.legend()

plt.tight_layout()
plt.show()
```



In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Create the linear regression model instance
lm = LinearRegression()

# For Glucose
lm.fit(df[['Glucose']], df['DiabetesPedigreeFunction'])
glucose_pred = lm.predict(df[['Glucose']])

# For Age
lm.fit(df[['Age']], df['DiabetesPedigreeFunction'])
age_pred = lm.predict(df[['Age']])

# For Insulin
lm.fit(df[['Insulin']], df['DiabetesPedigreeFunction'])
insulin_pred = lm.predict(df[['Insulin']])

# Plotting
plt.figure(figsize=(15, 7))

# Actual Data
plt.scatter(df['Glucose'], df['DiabetesPedigreeFunction'], color='blue', label=
plt.scatter(df['Age'], df['DiabetesPedigreeFunction'], color='green', label='A
plt.scatter(df['Insulin'], df['DiabetesPedigreeFunction'], color='purple', lab

# Regression Lines
plt.plot(df['Glucose'], glucose_pred, color='red', label='Regression Line (Glu
plt.plot(df['Age'], age_pred, color='orange', label='Regression Line (Age)')
plt.plot(df['Insulin'], insulin_pred, color='yellow', label='Regression Line (

plt.xlabel('Independent Variables')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Simple Linear Regression for Each Variable')
plt.legend()

plt.tight_layout()
plt.show()
```



Support Vector Machine

- support vector machines are supervised learning models with associated learning algorithms that analyze data for * classification and regression analysis four types of kernal supported by the SVM
- linear
- polynomial
- Radial basis function (RBF) (defaulted)
- sigmoid
- Gamma: it is the kernal coeffcient, which is a parameter that determines the width of the kernel function
- C: C is a regularization parameter that controls the trade-off between achieving a good fit to the training data and a simple decision boundary.

In [16]:
```
df
```

Out[16]:

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	1	6	148	72	35	0	33.6	
1	2	1	85	66	29	0	26.6	
2	3	8	183	64	0	0	23.3	
3	4	1	89	66	23	94	28.1	
4	5	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	...	
2763	2764	2	75	64	24	55	29.7	
2764	2765	8	179	72	42	130	32.7	
2765	2766	6	85	78	0	0	31.2	
2766	2767	0	129	110	46	130	67.1	
2767	2768	2	81	72	15	76	30.1	

2768 rows × 10 columns

In [17]:
```
# now set the values for independent and dependent in to features and label var
X =np.array(df.drop('Outcome',axis=1)) # for features we drop the outcome whic
y = df['Outcome'] #target
X[0:5]
```

Out[17]:
```
array([[1.000e+00, 6.000e+00, 1.480e+02, 7.200e+01, 3.500e+01, 0.000e+00,
        3.360e+01, 6.270e-01, 5.000e+01],
       [2.000e+00, 1.000e+00, 8.500e+01, 6.600e+01, 2.900e+01, 0.000e+00,
        2.660e+01, 3.510e-01, 3.100e+01],
       [3.000e+00, 8.000e+00, 1.830e+02, 6.400e+01, 0.000e+00, 0.000e+00,
        2.330e+01, 6.720e-01, 3.200e+01],
       [4.000e+00, 1.000e+00, 8.900e+01, 6.600e+01, 2.300e+01, 9.400e+01,
        2.810e+01, 1.670e-01, 2.100e+01],
       [5.000e+00, 0.000e+00, 1.370e+02, 4.000e+01, 3.500e+01, 1.680e+02,
        4.310e+01, 2.288e+00, 3.300e+01]])
```

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.svm import SVC
```

In [19]:
```
#now split the data in to Taining and testing data
# 0.3 shows that 30% data is for testing whereas 70% data is for training
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_stat

#1937*9 rows
X_train.shape

#1937*1
y_train.shape #(total training transactions)

#831*9 rows
X_test.shape

#831, 1
y_test.shape #(total testing Transactions)
```

Out[19]:
```
(831,)
```

In [20]:
```
# create a model for SVM
svm_model = SVC(kernel='linear',gamma='auto',C=2)
```

In [21]:
```
svm_model.fit(X_train,y_train)
```

Out[21]:
```
SVC(C=2, gamma='auto', kernel='linear')
```
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 [22]:
```
y_pred= svm_model.predict(X_test)
```

In [23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       562
           1       0.70      0.55      0.61       269

    accuracy                           0.78       831
   macro avg       0.75      0.72      0.73       831
weighted avg       0.77      0.78      0.77       831
```

In [24]:
```
#now check the accuracy
from sklearn.metrics import accuracy_score, precision_score, recall_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.776173285198556
```

In [25]:
```
# Calculate precision and recall
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

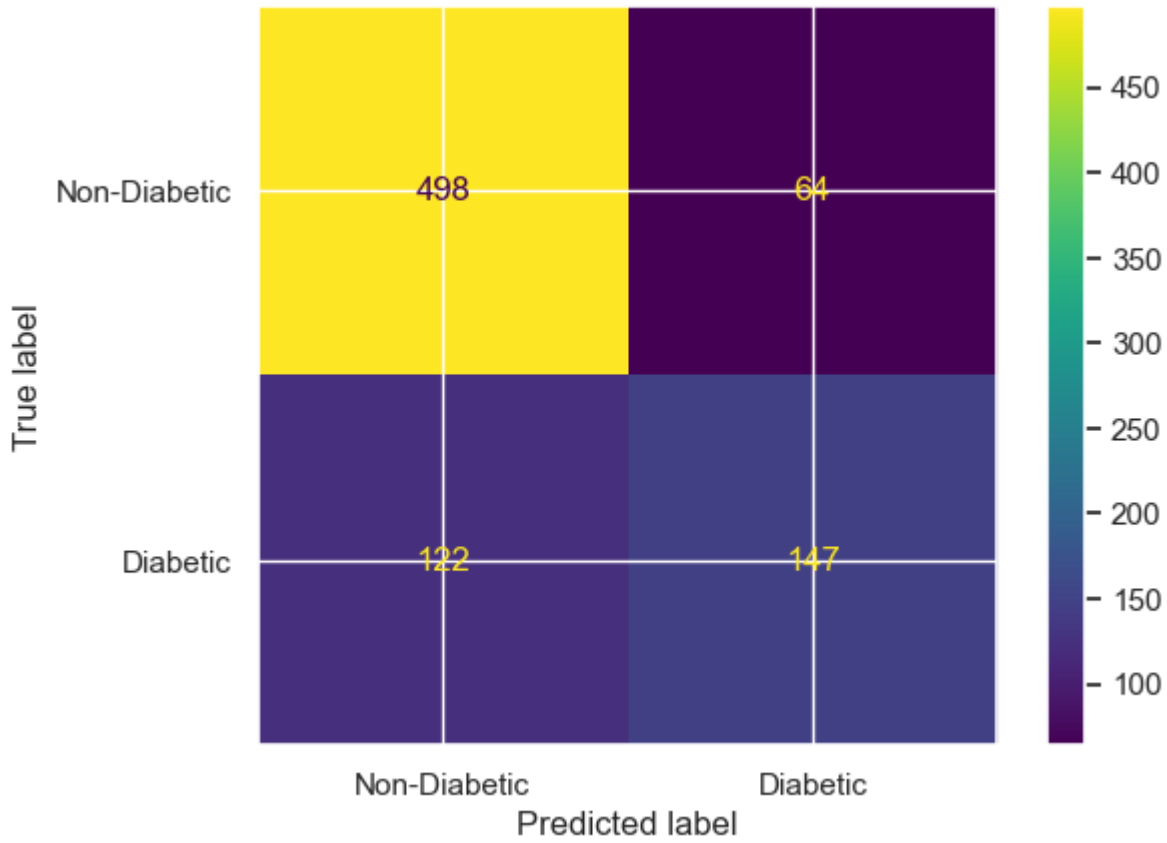
```
Precision: 0.6966824644549763
Recall: 0.5464684014869888
```

In [26]:
```
# Create a confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay, roc_curve

confusion = confusion_matrix(y_test, y_pred)
confusion
```

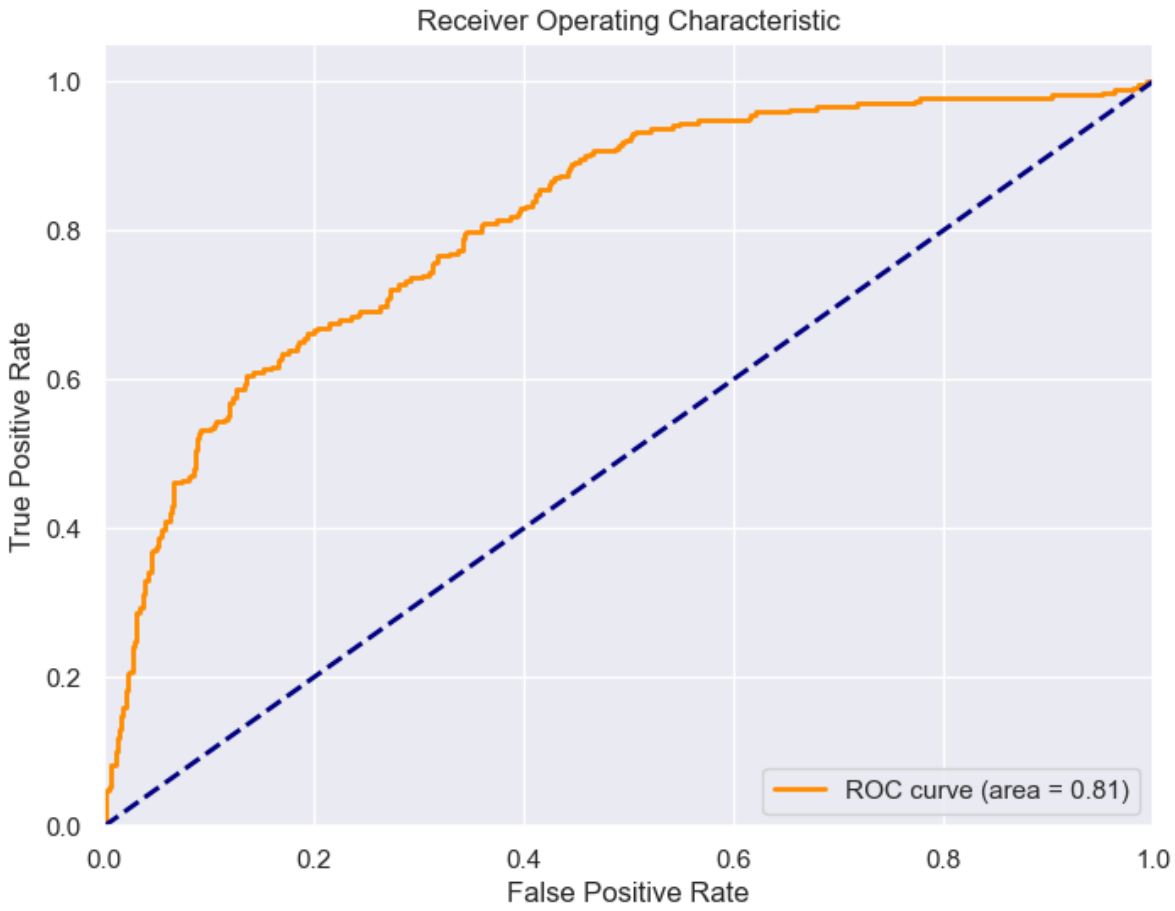
Out[26]:
```
array([[498,  64],
       [122, 147]], dtype=int64)
```

In [27]:
```
#plot the confusion matrix
matrix=ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels = ['N
matrix.plot()
```

Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24592ec72
b0>
```



In [28]:
```
#Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, svm_model.decision_function(X_test))
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



K-means Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

In [ ]:
```
#import the library for k-means
from sklearn.cluster import KMeans
```

In [57]:
```
df
```

Out[57]:

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	1	6	148	72	35	0	33.6	
1	2	1	85	66	29	0	26.6	
2	3	8	183	64	0	0	23.3	
3	4	1	89	66	23	94	28.1	
4	5	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	...	
2763	2764	2	75	64	24	55	29.7	
2764	2765	8	179	72	42	130	32.7	
2765	2766	6	85	78	0	0	31.2	
2766	2767	0	129	110	46	130	67.1	
2767	2768	2	81	72	15	76	30.1	

2768 rows × 11 columns

In [76]:
```
df = df.drop(0)
```

In [77]:
```
df.iloc[ :,0:1]
```

Out[77]:

	Pregnancies
1	1
2	8
3	1
4	0
5	5
...	...
2763	2
2764	8
2765	6
2766	0
2767	2

2767 rows × 1 columns

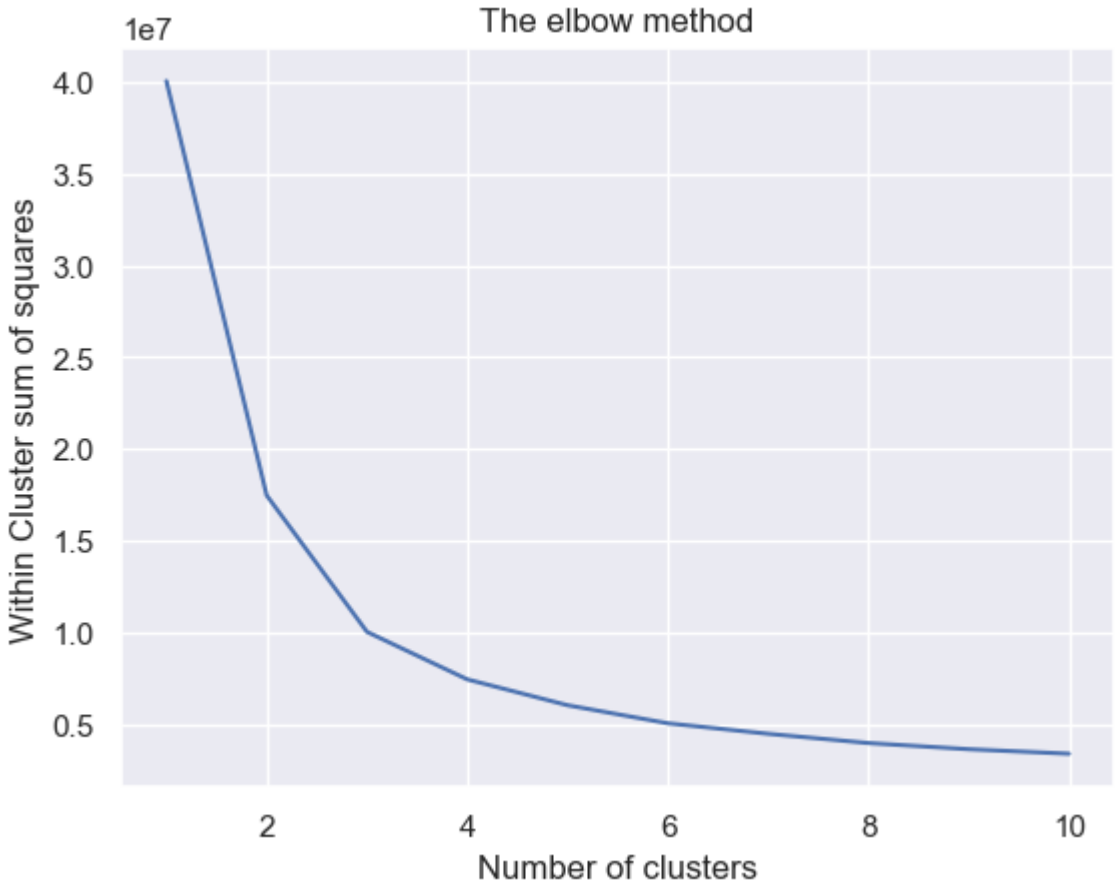
In [78]:
```
x = df.iloc[:, [0, 1, 2, 3,4,5,6,7]].values
x
```

Out[78]:
```
array([[1.00e+00, 8.50e+01, 6.60e+01, ..., 2.66e+01, 3.51e-01, 3.10e+01],
       [8.00e+00, 1.83e+02, 6.40e+01, ..., 2.33e+01, 6.72e-01, 3.20e+01],
       [1.00e+00, 8.90e+01, 6.60e+01, ..., 2.81e+01, 1.67e-01, 2.10e+01],
       ...,
       [6.00e+00, 8.50e+01, 7.80e+01, ..., 3.12e+01, 3.82e-01, 4.20e+01],
       [0.00e+00, 1.29e+02, 1.10e+02, ..., 6.71e+01, 3.19e-01, 2.60e+01],
       [2.00e+00, 8.10e+01, 7.20e+01, ..., 3.01e+01, 5.47e-01, 2.50e+01]])
```

In [79]:
```
y=df.iloc[:,8:9].values
y
```

Out[79]:
```
array([[0],
       [1],
       [0],
       ...,
       [0],
       [1],
       [0]], dtype=int64)
```

In [81]:
```
#Now we will implement the elbow method to find the optimum number of clusters
results = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init
    kmeans.fit(x)
    results.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), results)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Within er sum of squaresClust') #within cluster sum of squares
plt.show()
```



In [82]:
```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 300, n_init = 1
y_kmeans = kmeans.fit_predict(x)
print(y_kmeans)
```

```
[2 2 0 ... 2 0 0]
```

In [95]:
```
df
```

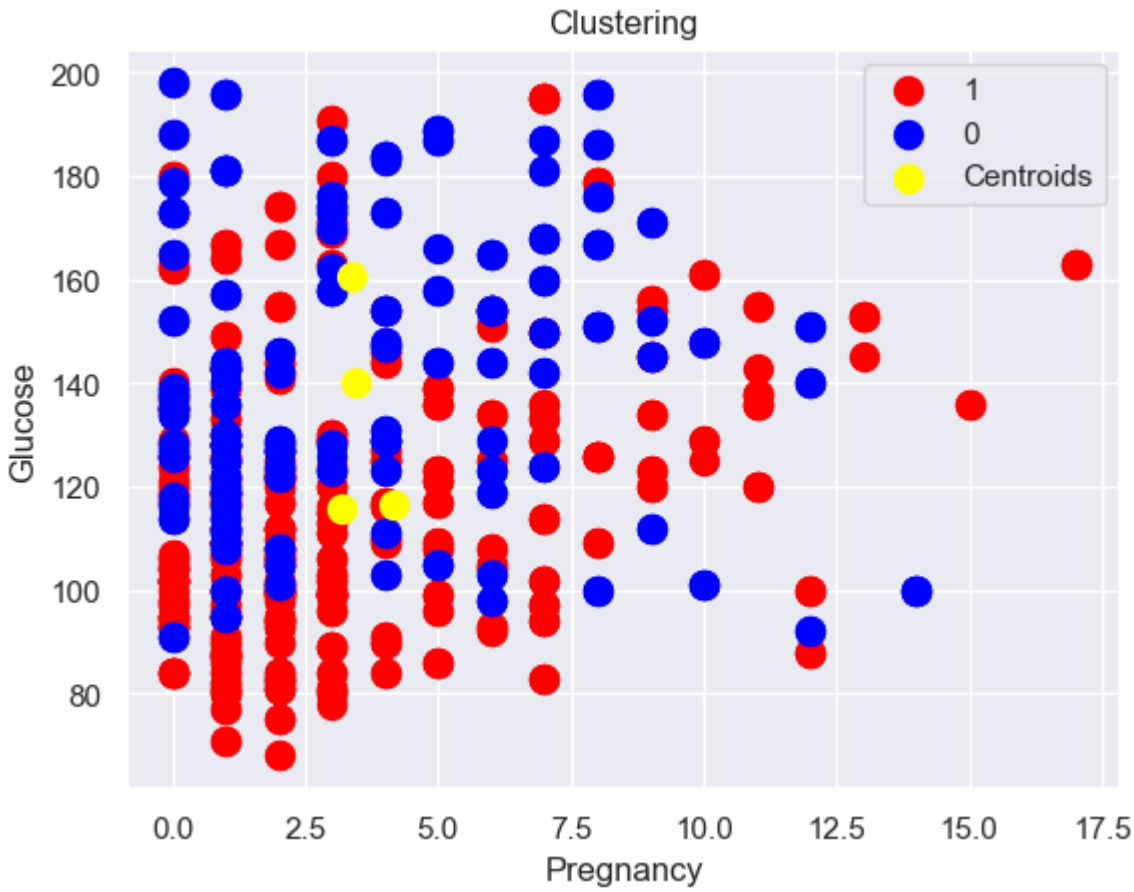
Out[95]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
5	5	116	74	0	0	25.6	0.2
...	...	...	...	...	...	...	
2763	2	75	64	24	55	29.7	0.3
2764	8	179	72	42	130	32.7	0.7
2765	6	85	78	0	0	31.2	0.3
2766	0	129	110	46	130	67.1	0.3
2767	2	81	72	15	76	30.1	0.5

2767 rows × 10 columns

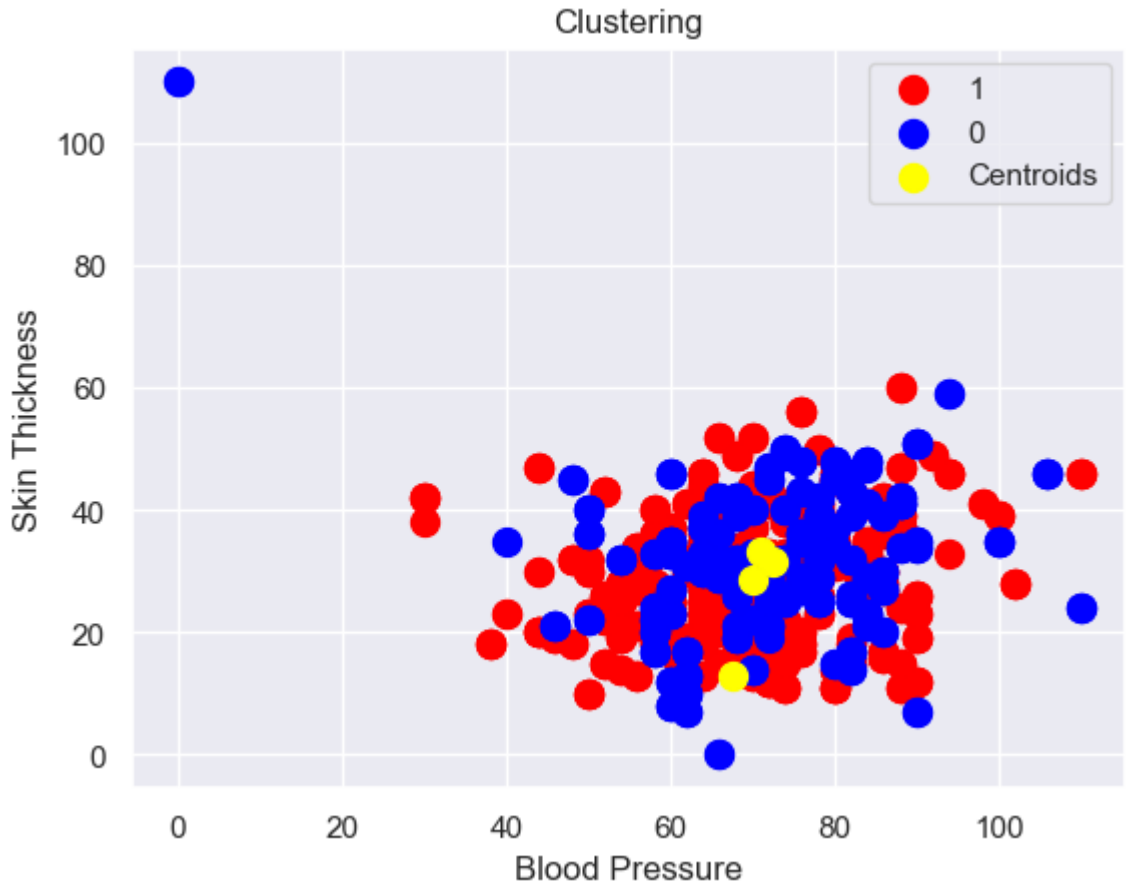
In [93]:
```
#Visualising the clusters
#This line creates a scatter plot of data points that belong to cluster 0. It 
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red', labe
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue', lab


#Plotting the centroids of the clusters  index 0 and 1
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 1

plt.title('Clustering')
plt.xlabel('Pregnancy')
plt.ylabel('Glucose')
plt.legend()
```

Out[93]: <matplotlib.legend.Legend at 0x2459ddcc040>



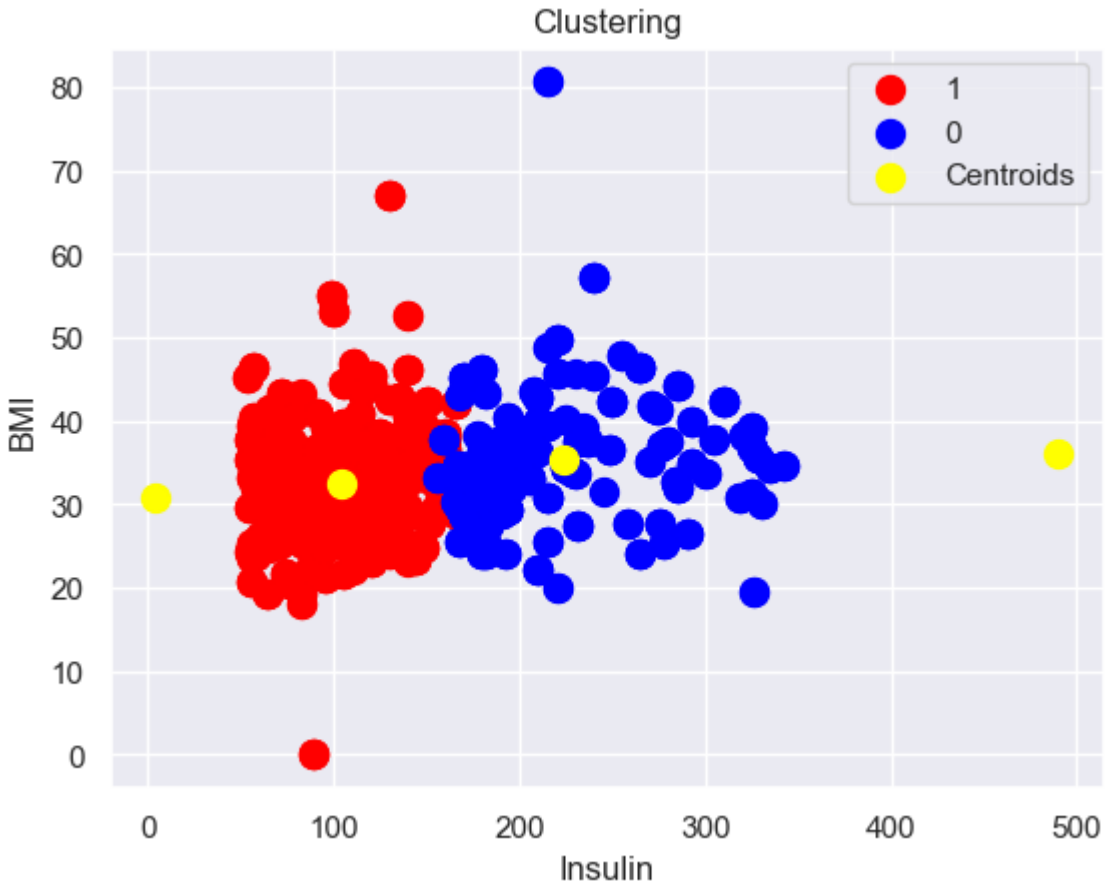
In [94]:
```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 2], x[y_kmeans == 0, 3], s = 100, c = 'red', label
plt.scatter(x[y_kmeans == 1, 2], x[y_kmeans == 1, 3], s = 100, c = 'blue', labe


#Plotting the centroids of the clusters of index 2  and 3
plt.scatter(kmeans.cluster_centers_[:, 2], kmeans.cluster_centers_[:,3], s = 1


plt.title('Clustering')
plt.xlabel('Blood Pressure')
plt.ylabel('Skin Thickness')
plt.legend()
```

Out[94]: <matplotlib.legend.Legend at 0x2459dca53d0>



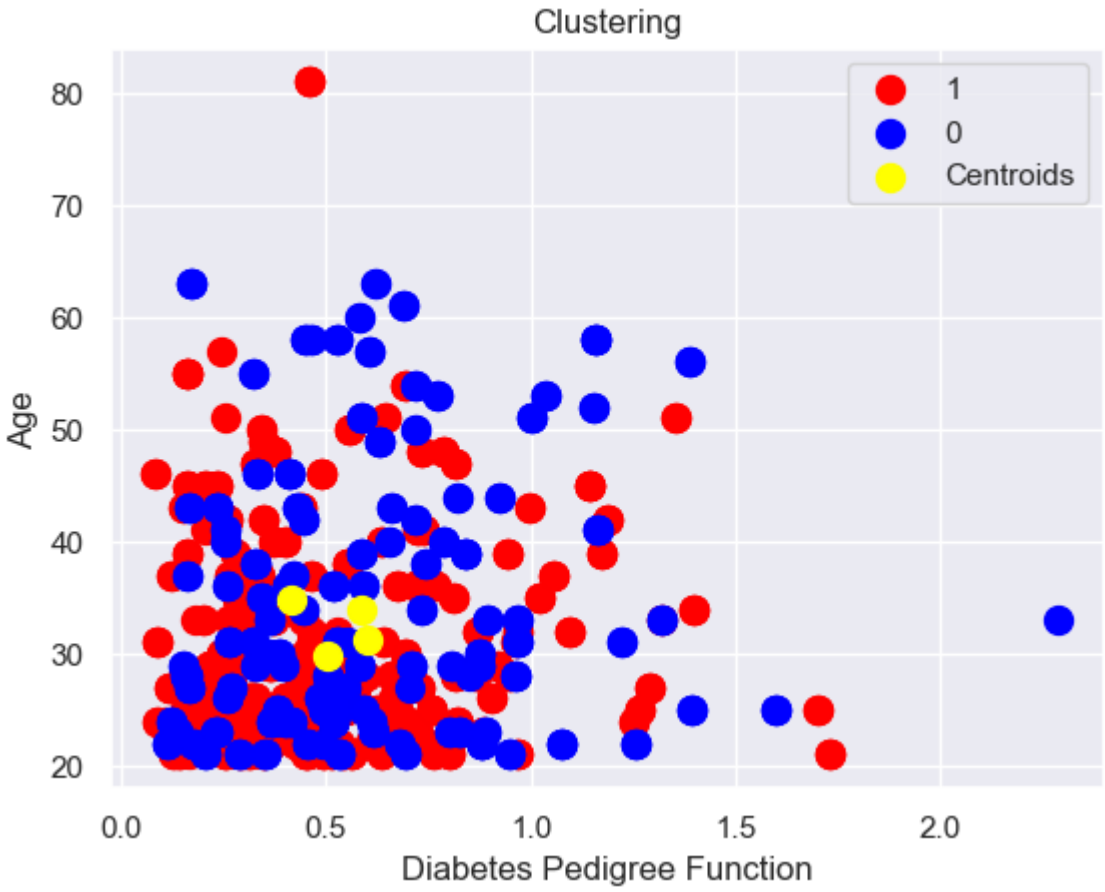
In [96]:
```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 4], x[y_kmeans == 0, 5], s = 100, c = 'red', label
plt.scatter(x[y_kmeans == 1, 4], x[y_kmeans == 1, 5], s = 100, c = 'blue', lab


#Plotting the centroids of the clusters of index 4 and 5
plt.scatter(kmeans.cluster_centers_[:, 4], kmeans.cluster_centers_[:,5], s = 1
plt.title('Clustering')
plt.xlabel('Insulin')
plt.ylabel('BMI')
plt.legend()
```

Out[96]: <matplotlib.legend.Legend at 0x2459dd248b0>



In [97]:
```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 6], x[y_kmeans == 0, 7], s = 100, c = 'red', label
plt.scatter(x[y_kmeans == 1, 6], x[y_kmeans == 1, 7], s = 100, c = 'blue', lab

#Plotting the centroids of the clusters of index 6 and 7
plt.scatter(kmeans.cluster_centers_[:, 6], kmeans.cluster_centers_[:,7], s = 1

plt.title('Clustering')
plt.xlabel('Diabetes Pedigree Function')
plt.ylabel('Age')
plt.legend()
```

Out[97]: <matplotlib.legend.Legend at 0x2459e2b0df0>



In [87]:
```
from sklearn.metrics import silhouette_score
silhouette_avg = silhouette_score(x,y_kmeans)
print("For n_clusters =",4 ,
          "The average silhouette_score is :", silhouette_avg)
```

For n_clusters = 4 The average silhouette_score is : 0.4279899882760121