PEEKA SAIKUMAR

20KQ1A05H8

Import Necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib
import theano
import tensorflow
import keras
import sklearn
import seaborn
```

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation
for BLAS functions.
```

Versions of Libraries

```
print("Numpy Version:", np.__version__)
print("Pandas Version:", pd.__version__)
print("Matplotlib Version:", matplotlib.__version__)
print("Theano Version:", theano.__version__)
print("Tensorflow Version:", tensorflow.__version__)
print("Keras Version:", keras.__version__)
print("Sklearn Version:", sklearn.__version__)
print("Seaborn Version:", seaborn.__version__)
```

```
Numpy Version: 1.22.3
Pandas Version: 2.0.3
Matplotlib Version: 3.7.2
Theano Version: 1.0.5
Tensorflow Version: 2.10.0
Keras Version: 2.10.0
Sklearn Version: 1.3.0
Seaborn Version: 0.13.2
```

Read the dataset from the CSV file

```
dataset = pd.read_csv('C:/Users/Pavan Kalyan/OneDrive/Desktop/Diabetes
Project/diabetes.csv')
```

Get the total number of rows in the dataset

```
print(len(dataset))
```

```
768
```

Get the total number of columns in the dataset

```
print(len(dataset.columns))
```

```
9
```

Find the colum names of the dataset

```
print(dataset.columns)
```

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

Extract features from the dataset

```
X = dataset.iloc[:, 0:8].values
```

Extract the target variable from the dataset

```
y = dataset.iloc[:, 8].values
```

Import LabelEncoder and OneHotEncoder for encoding

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

Apply LabelEncoder to each column in the specified range

```
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

labelencoder_X_3 = LabelEncoder()
X[:, 3] = labelencoder_X_3.fit_transform(X[:, 3])

labelencoder_X_4 = LabelEncoder()
X[:, 4] = labelencoder_X_4.fit_transform(X[:, 4])

labelencoder_X_5 = LabelEncoder()
X[:, 5] = labelencoder_X_5.fit_transform(X[:, 5])
```

```
labelencoder_X_6 = LabelEncoder()
X[:, 6] = labelencoder_X_6.fit_transform(X[:, 6])

labelencoder_X_7 = LabelEncoder()
X[:, 7] = labelencoder_X_7.fit_transform(X[:, 7])
```

Apply OneHotEncoding to the features to create dummy variables

```
onehotencoder = OneHotEncoder()
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

Split the Dataset into Training and Testing sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.2)
```

Initialize StandardScaler to scale features

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

Standardize the training data and test data using the same scaler, fittng on the training data only

```
X_train = sc.fit_transform(X_train)
X = sc.fit_transform(X)
X_test = sc.transform(X_test)
```

Initialize the neural network classifier as a sequential model

```
from keras.models import Sequential
from keras.layers import Dense
classifier = Sequential()
```

Add layers to the neural network

First layer: Fully connected with 6 units, ReLU activation, and input dimension from X_train

```
classifier.add(Dense(units=6, kernel_initializer='uniform', 
activation='relu', input_dim=X_train.shape[1]))
```

Second layer: Fully connected with 6 units and ReLU activation

```
classifier.add(Dense(units=6, kernel_initializer='uniform',
activation='relu'))
```

Output layer: Fully connected with 1 unit and sigmoid activation for binary classification

```
classifier.add(Dense(units=1, kernel_initializer='uniform',
activation='sigmoid'))
```

Compile the model using Adam optimizer, binary crossentropy loss, and accuracy as a metric

```
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Train the neural network model using the training data

```
classifier.fit(X_train, y_train, batch_size=10, epochs=100)

Epoch 1/100
62/62 [==============================] - 1s 3ms/step - loss: 0.6884 -
accuracy: 0.6515
Epoch 2/100
62/62 [==============================] - 0s 3ms/step - loss: 0.6523 -
accuracy: 0.6515
Epoch 3/100
62/62 [==============================] - 0s 3ms/step - loss: 0.5529 -
accuracy: 0.6515
Epoch 4/100
62/62 [==============================] - 0s 3ms/step - loss: 0.4304 -
accuracy: 0.6515
Epoch 5/100
62/62 [==============================] - 0s 3ms/step - loss: 0.3410 -
accuracy: 0.6612
Epoch 6/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2899 -
accuracy: 0.8893
Epoch 7/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2613 -
accuracy: 0.9674
Epoch 8/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2423 -
accuracy: 0.9805
Epoch 9/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2288 -
accuracy: 0.9870
Epoch 10/100
```

```
62/62 [==============================] - 0s 3ms/step - loss: 0.2177 -
accuracy: 0.9870
Epoch 11/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2083 -
accuracy: 0.9870
Epoch 12/100
62/62 [==============================] - 0s 3ms/step - loss: 0.2000 -
accuracy: 0.9886
Epoch 13/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1920 -
accuracy: 0.9886
Epoch 14/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1851 -
accuracy: 0.9886
Epoch 15/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1786 -
accuracy: 0.9886
Epoch 16/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1725 -
accuracy: 0.9886
Epoch 17/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1666 -
accuracy: 0.9886
Epoch 18/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1611 -
accuracy: 0.9886
Epoch 19/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1559 -
accuracy: 0.9886
Epoch 20/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1509 -
accuracy: 0.9886
Epoch 21/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1463 -
accuracy: 0.9886
Epoch 22/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1418 -
accuracy: 0.9886
Epoch 23/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1376 -
accuracy: 0.9886
Epoch 24/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1336 -
accuracy: 0.9886
Epoch 25/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1299 -
accuracy: 0.9886
Epoch 26/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1262 -
```

```
accuracy: 0.9886
Epoch 27/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1229 -
accuracy: 0.9886
Epoch 28/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1196 -
accuracy: 0.9886
Epoch 29/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1164 -
accuracy: 0.9886
Epoch 30/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1134 -
accuracy: 0.9886
Epoch 31/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1106 -
accuracy: 0.9886
Epoch 32/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1079 -
accuracy: 0.9886
Epoch 33/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1053 -
accuracy: 0.9886
Epoch 34/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1029 -
accuracy: 0.9886
Epoch 35/100
62/62 [==============================] - 0s 3ms/step - loss: 0.1005 -
accuracy: 0.9886
Epoch 36/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0983 -
accuracy: 0.9886
Epoch 37/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0961 -
accuracy: 0.9886
Epoch 38/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0941 -
accuracy: 0.9886
Epoch 39/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0921 -
accuracy: 0.9886
Epoch 40/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0902 -
accuracy: 0.9886
Epoch 41/100
62/62 [==============================] - 0s 4ms/step - loss: 0.0885 -
accuracy: 0.9886
Epoch 42/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0868 -
accuracy: 0.9886
```

```
Epoch 43/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0852 -
accuracy: 0.9886
Epoch 44/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0836 -
accuracy: 0.9886
Epoch 45/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0821 -
accuracy: 0.9886
Epoch 46/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0807 -
accuracy: 0.9886
Epoch 47/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0793 -
accuracy: 0.9886
Epoch 48/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0781 -
accuracy: 0.9886
Epoch 49/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0768 -
accuracy: 0.9886
Epoch 50/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0756 -
accuracy: 0.9886
Epoch 51/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0745 -
accuracy: 0.9886
Epoch 52/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0734 -
accuracy: 0.9886
Epoch 53/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0724 -
accuracy: 0.9886
Epoch 54/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0714 -
accuracy: 0.9886
Epoch 55/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0705 -
accuracy: 0.9886
Epoch 56/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0696 -
accuracy: 0.9886
Epoch 57/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0687 -
accuracy: 0.9886
Epoch 58/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0679 -
accuracy: 0.9886
Epoch 59/100
```

```
62/62 [==============================] - 0s 2ms/step - loss: 0.0671 -
accuracy: 0.9886
Epoch 60/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0663 -
accuracy: 0.9886
Epoch 61/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0656 -
accuracy: 0.9886
Epoch 62/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0649 -
accuracy: 0.9886
Epoch 63/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0643 -
accuracy: 0.9886
Epoch 64/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0636 -
accuracy: 0.9886
Epoch 65/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0630 -
accuracy: 0.9886
Epoch 66/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0625 -
accuracy: 0.9886
Epoch 67/100
62/62 [==============================] - 0s 4ms/step - loss: 0.0619 -
accuracy: 0.9886
Epoch 68/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0614 -
accuracy: 0.9886
Epoch 69/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0608 -
accuracy: 0.9886
Epoch 70/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0604 -
accuracy: 0.9886
Epoch 71/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0599 -
accuracy: 0.9886
Epoch 72/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0595 -
accuracy: 0.9886
Epoch 73/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0591 -
accuracy: 0.9886
Epoch 74/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0587 -
accuracy: 0.9886
Epoch 75/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0583 -
```

```
accuracy: 0.9886
Epoch 76/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0580 -
accuracy: 0.9886
Epoch 77/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0576 -
accuracy: 0.9886
Epoch 78/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0573 -
accuracy: 0.9886
Epoch 79/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0570 -
accuracy: 0.9886
Epoch 80/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0566 -
accuracy: 0.9886
Epoch 81/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0564 -
accuracy: 0.9886
Epoch 82/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0561 -
accuracy: 0.9886
Epoch 83/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0558 -
accuracy: 0.9886
Epoch 84/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0556 -
accuracy: 0.9886
Epoch 85/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0553 -
accuracy: 0.9886
Epoch 86/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0551 -
accuracy: 0.9886
Epoch 87/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0549 -
accuracy: 0.9886
Epoch 88/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0547 -
accuracy: 0.9886
Epoch 89/100
62/62 [==============================] - 0s 4ms/step - loss: 0.0545 -
accuracy: 0.9886
Epoch 90/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0543 -
accuracy: 0.9886
Epoch 91/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0541 -
accuracy: 0.9886
```

```
Epoch 92/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0539 -
accuracy: 0.9886
Epoch 93/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0537 -
accuracy: 0.9886
Epoch 94/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0536 -
accuracy: 0.9886
Epoch 95/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0534 -
accuracy: 0.9886
Epoch 96/100
62/62 [==============================] - 0s 2ms/step - loss: 0.0533 -
accuracy: 0.9886
Epoch 97/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0532 -
accuracy: 0.9886
Epoch 98/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0530 -
accuracy: 0.9886
Epoch 99/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0529 -
accuracy: 0.9886
Epoch 100/100
62/62 [==============================] - 0s 3ms/step - loss: 0.0528 -
accuracy: 0.9886

<keras.callbacks.History at 0x26538d83550>
```

Predict the labels for the test set

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

5/5 [==============================] - 0s 3ms/step
```

Calculate the confusion matrix to assess the classification model's performance

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
```

Get the confusion matrix to evaluate model performance

```
print(cm)

[[85 15]
 [36 18]]
```

Get the accuracy of the model based on the test data and predicted values

```
print('Accuracy:', accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6688311688311688
```

Import necessary libraries for data visualization

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

Set a color palette suitable for categorical data

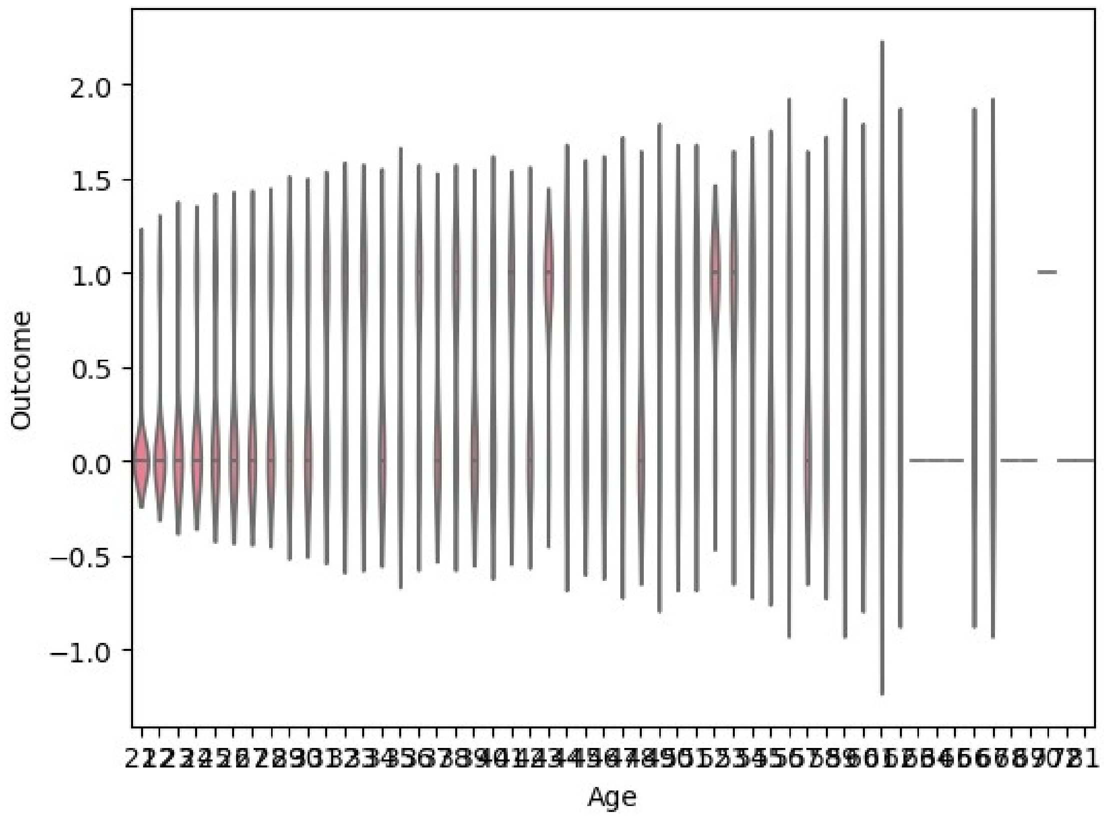
```
sns.set_palette('husl')
```

Drop the pregancies column

```
tmp = dataset.drop('Pregnancies', axis=1)
```

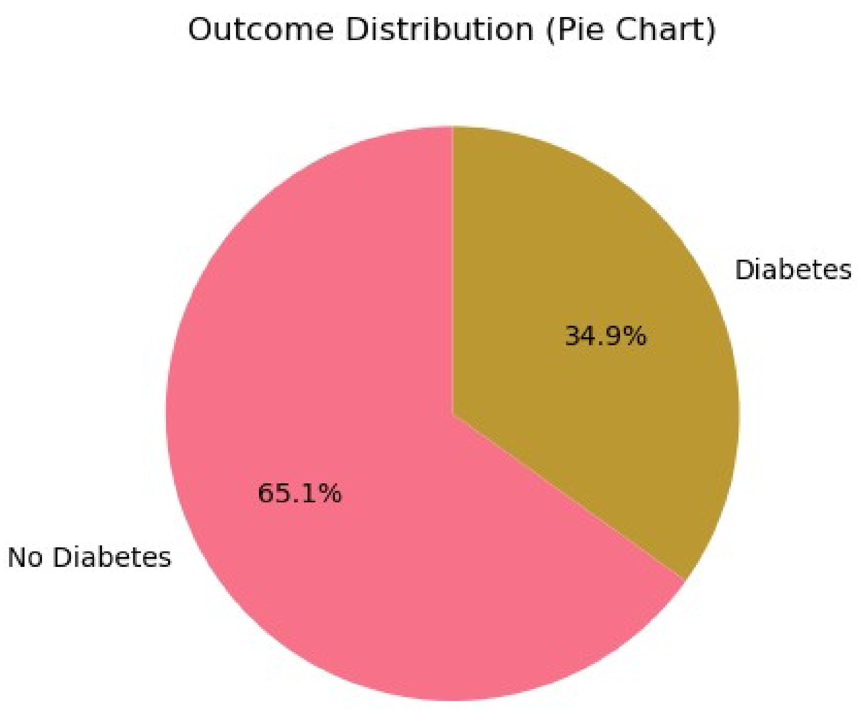
Violin plot for Age vs Outcome

```
sns.violinplot(y='Outcome', x='Age', data=dataset, inner='quartile')
plt.show()
```



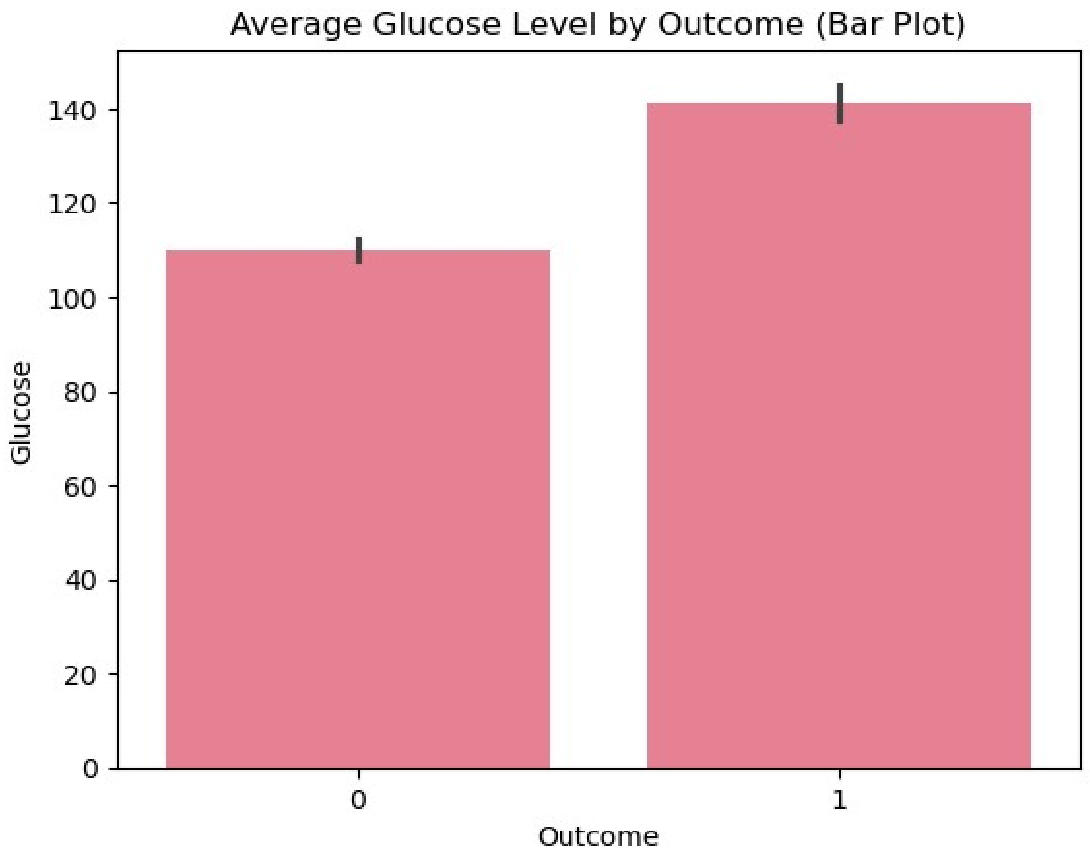
Pie chart for the distribution of 'Outcome'

```
outcome_counts = dataset['Outcome'].value_counts()
plt.pie(outcome_counts, labels=['No Diabetes', 'Diabetes'],
autopct='%1.1f%%', startangle=90)
plt.title('Outcome Distribution (Pie Chart)')
plt.show()
```



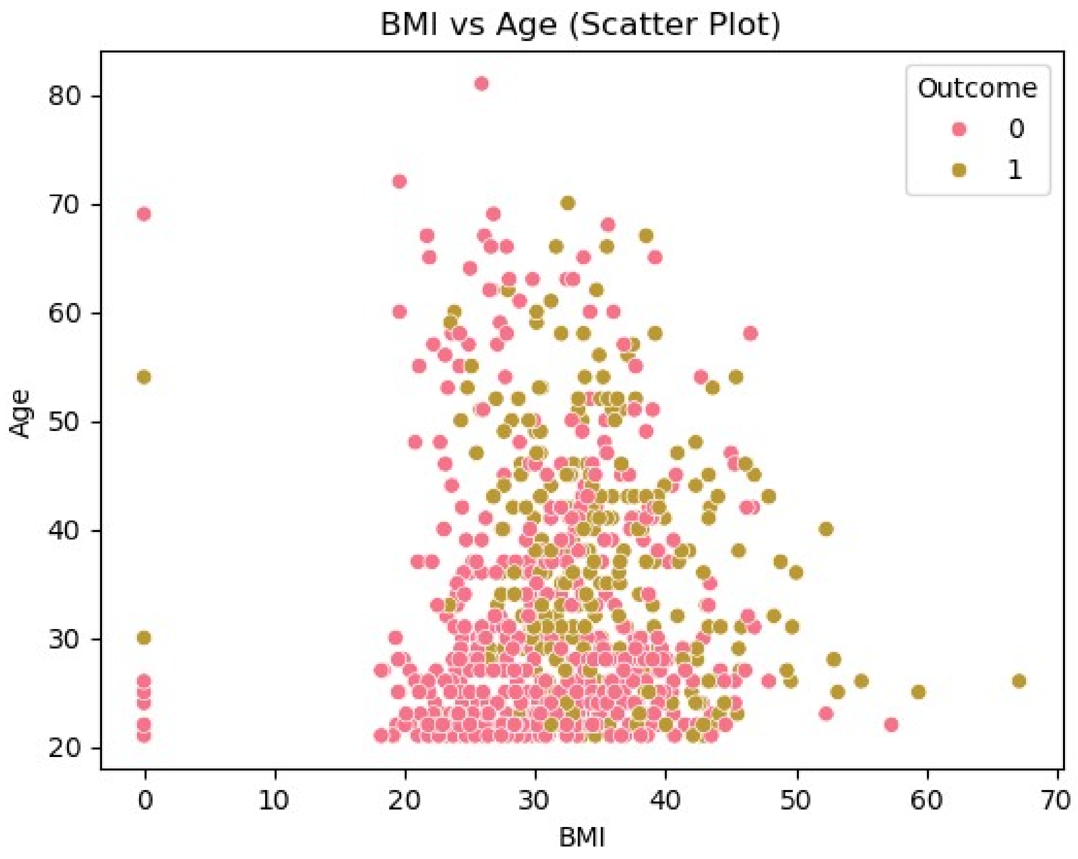
Bar Plot for Glucose level by 'Outcome'

```
sns.barplot(x='Outcome', y='Glucose', data=dataset)
plt.title('Average Glucose Level by Outcome (Bar Plot)')
plt.show()
```



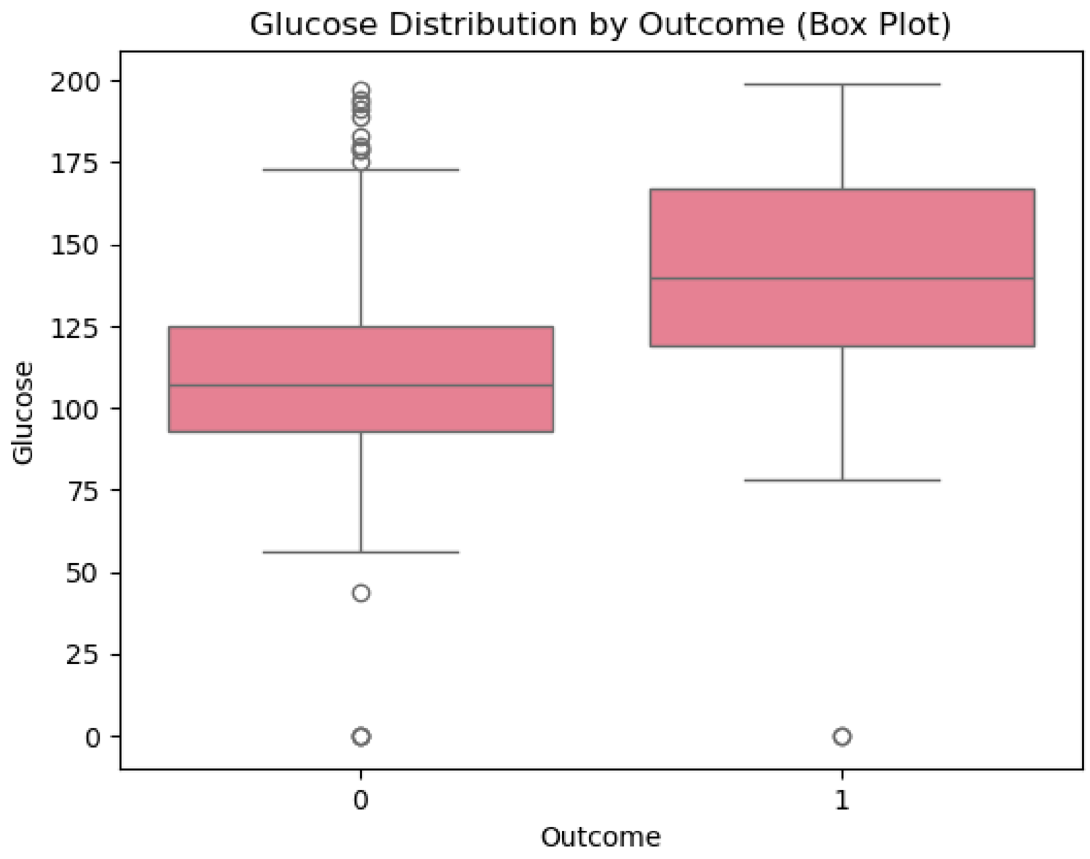
Scatter Plot (BMI vs Age)

```
sns.scatterplot(x='BMI', y='Age', data=dataset, hue='Outcome')
plt.title('BMI vs Age (Scatter Plot)')
plt.show()
```



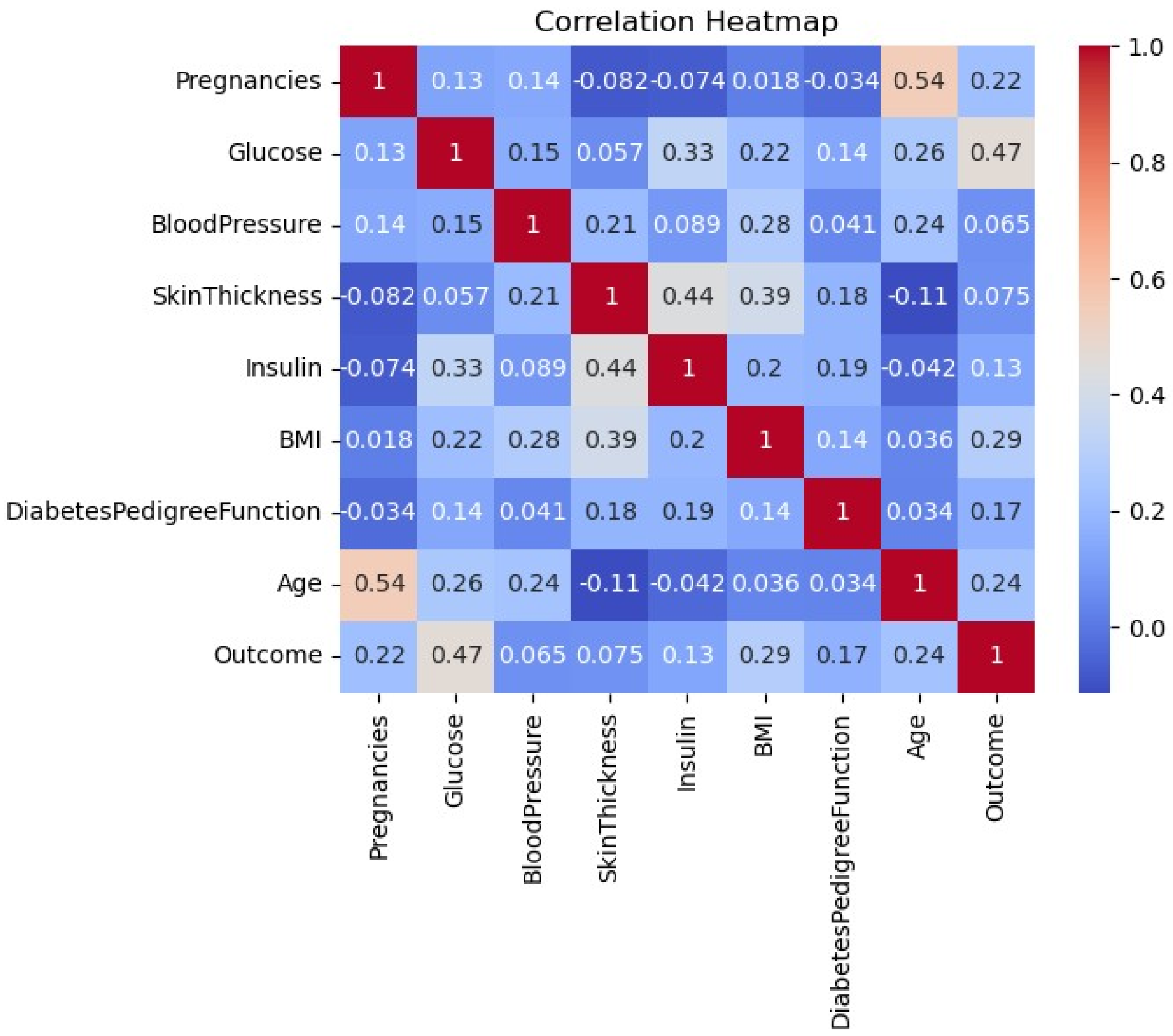
Box Plot for Glucose distribution by Outcome

```
sns.boxplot(x='Outcome', y='Glucose', data=dataset)
plt.title('Glucose Distribution by Outcome (Box Plot)')
plt.show()
```



Heatmap for correlation matrix

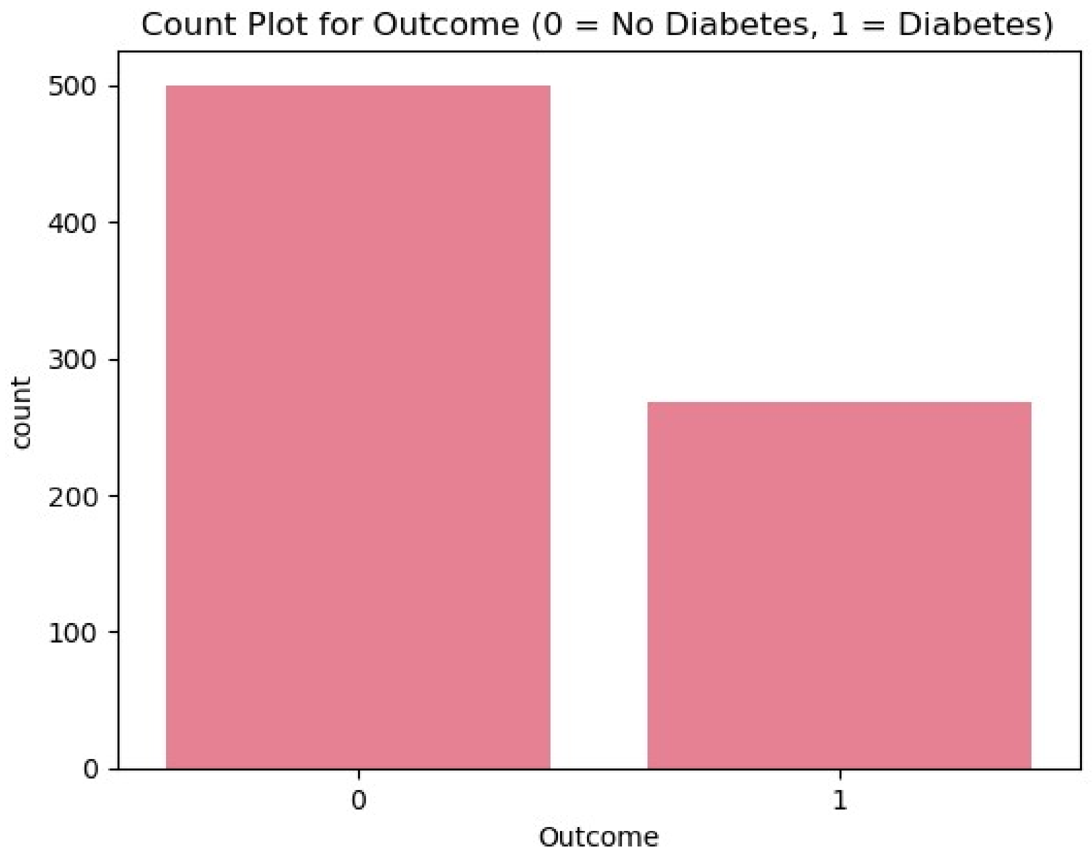
```
correlation_matrix = dataset.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

Count Plot for Outcome distribution (count of 0s and 1s)

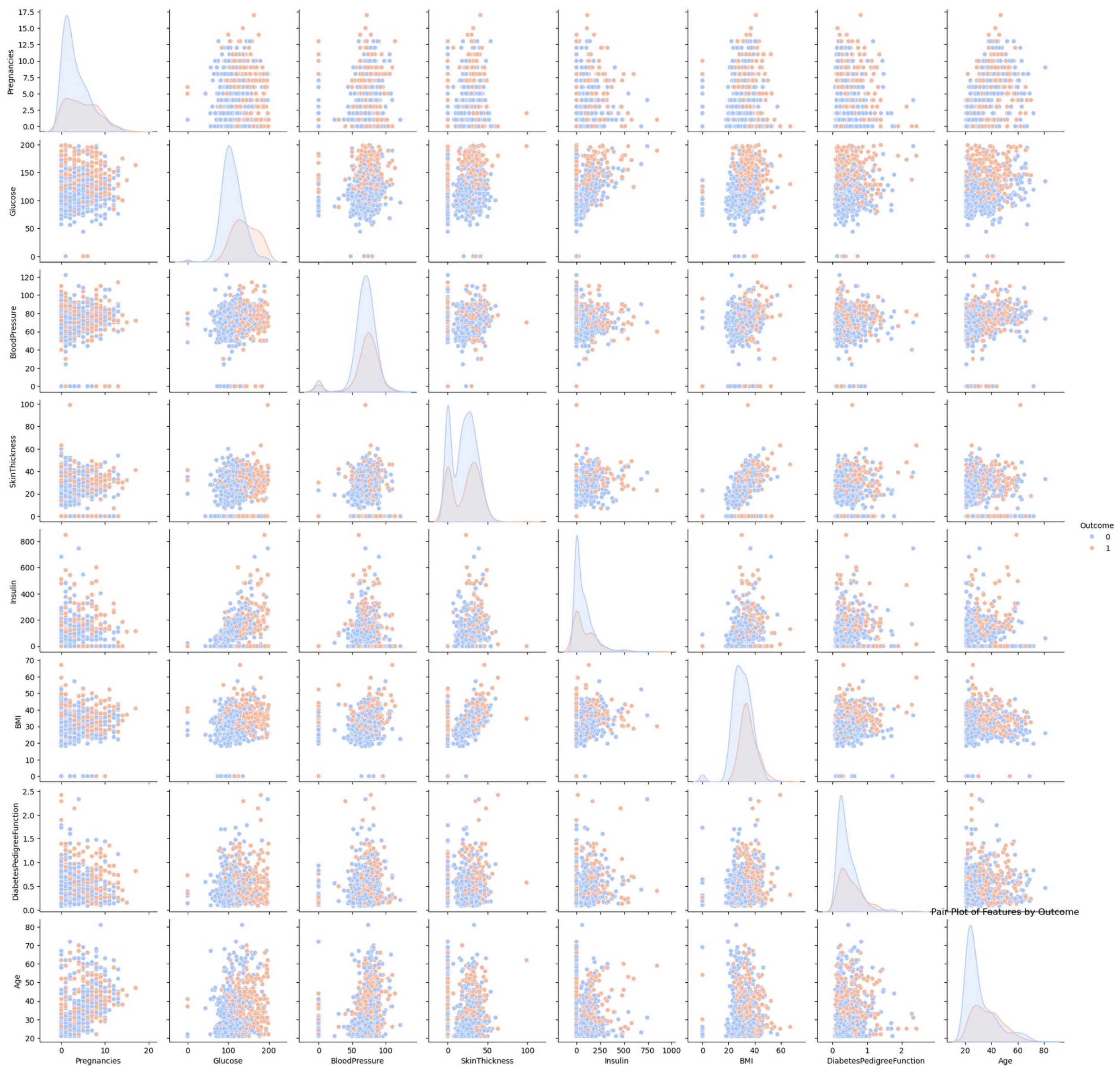
```
sns.countplot(x='Outcome', data=dataset)
plt.title('Count Plot for Outcome (0 = No Diabetes, 1 = Diabetes)')
plt.show()
```



Pair Plot for all numeric features to see relationships

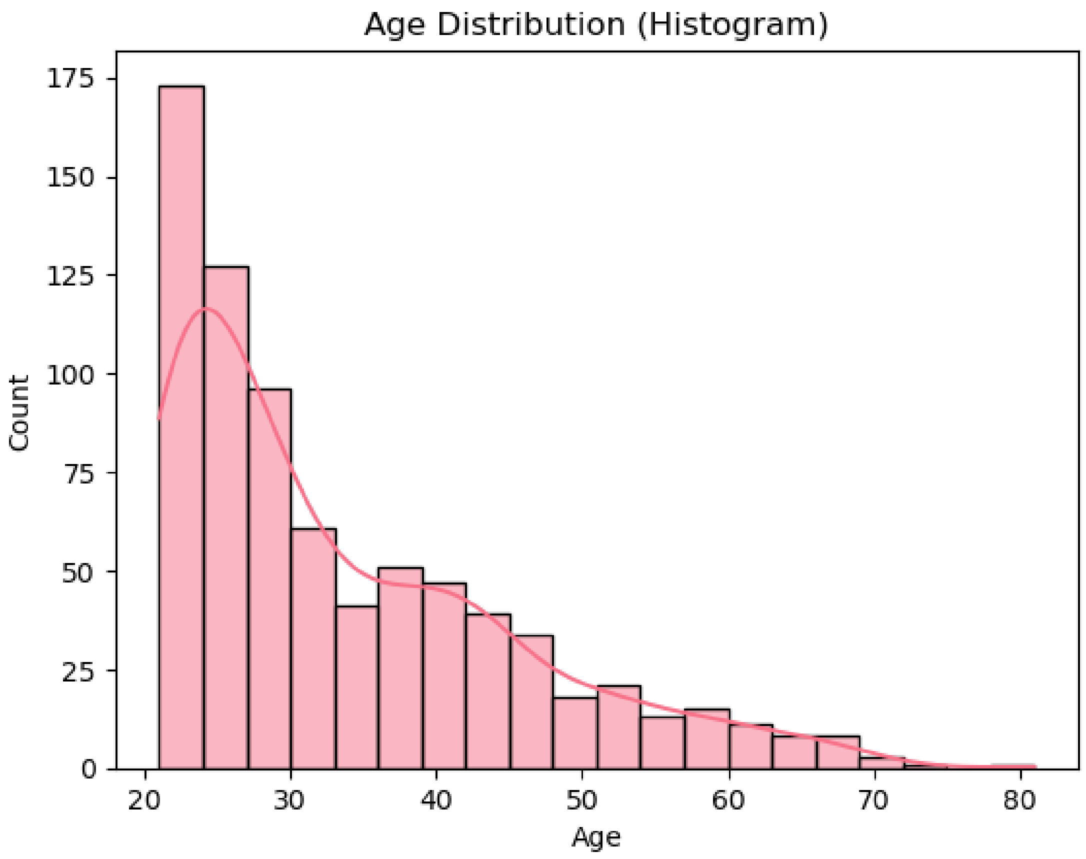
```
sns.pairplot(dataset, hue='Outcome', palette='coolwarm')
plt.title('Pair Plot of Features by Outcome')
plt.show()
```

```
C:\Users\Pavan Kalyan\anaconda3\envs\theano_env\lib\site-packages\
seaborn\axisgrid.py:123: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



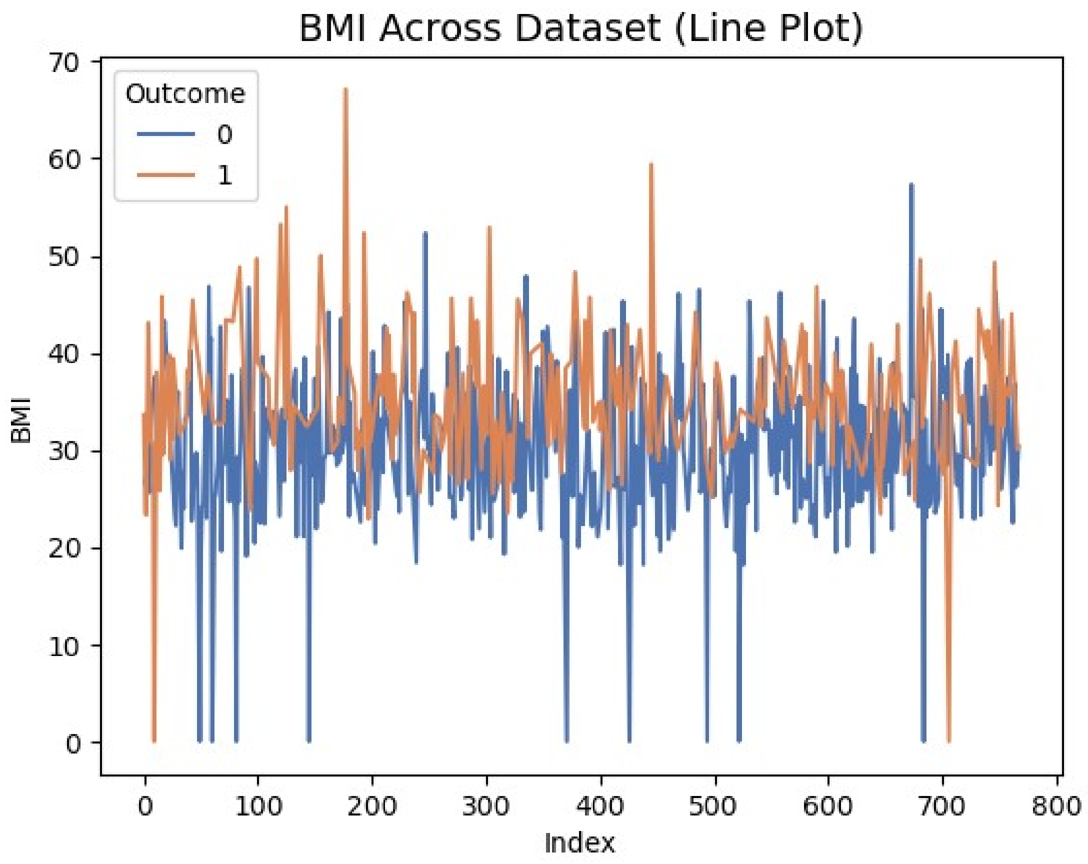
Histogram for Age distribution

```
sns.histplot(dataset['Age'], kde=True, bins=20)
plt.title('Age Distribution (Histogram)')
plt.show()
```



Line Plot tracking BMI across the dataset with Outcome hue

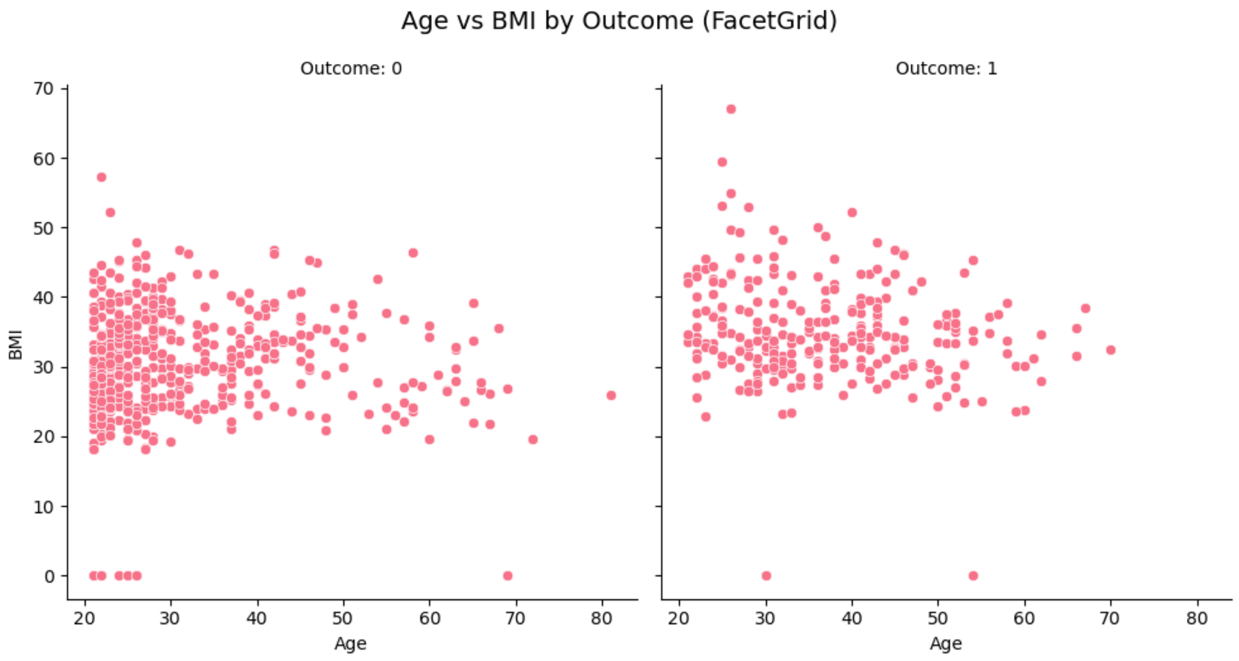
```
sns.lineplot(x=dataset.index, y=dataset['BMI'],
hue=dataset['Outcome'], palette='deep')
plt.title('BMI Across Dataset (Line Plot)', fontsize=14)
plt.xlabel('Index')
plt.ylabel('BMI')
plt.show()
```



FacetGrid showing Age vs BMI for different Outcomes

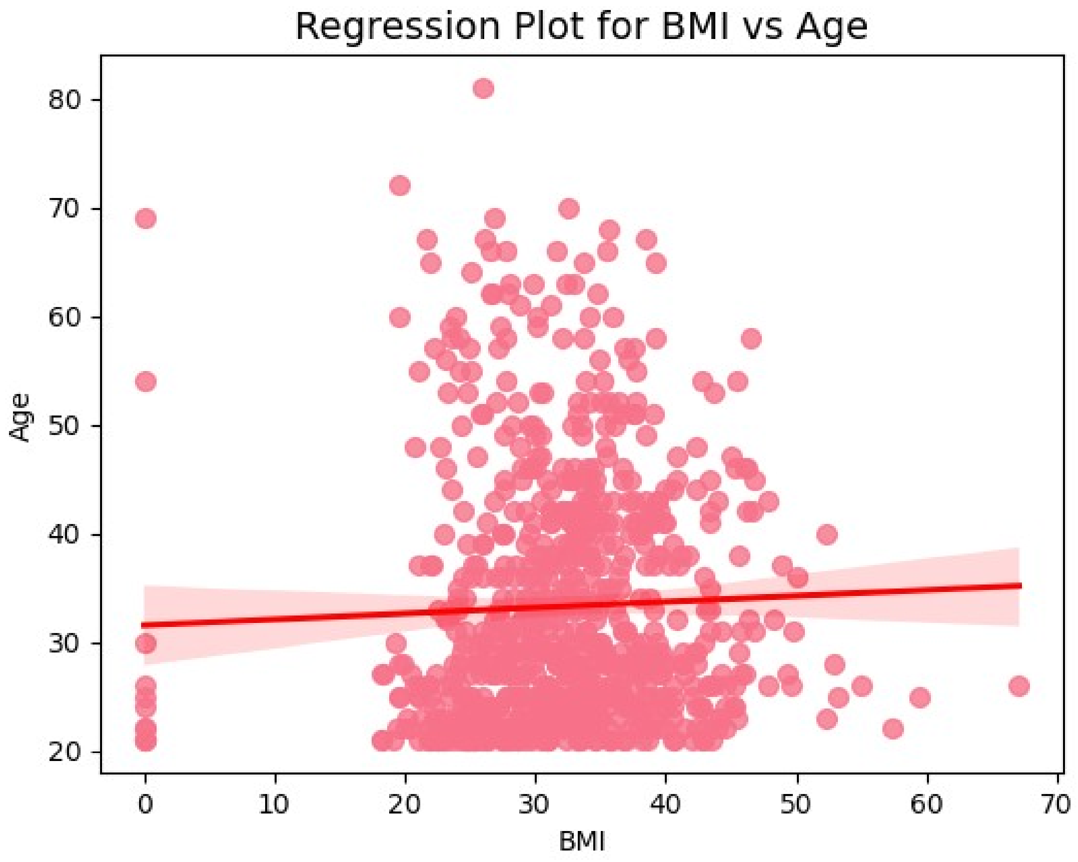
```
g = sns.FacetGrid(dataset, col='Outcome', height=5, palette='muted')
g.map(sns.scatterplot, 'Age', 'BMI')
g.set_axis_labels('Age', 'BMI')
g.set_titles('Outcome: {col_name}')
plt.suptitle('Age vs BMI by Outcome (FacetGrid)', fontsize=14, y=1.05)
plt.show()
```

```
C:\Users\Pavan Kalyan\anaconda3\envs\theano_env\lib\site-packages\
seaborn\axisgrid.py:123: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



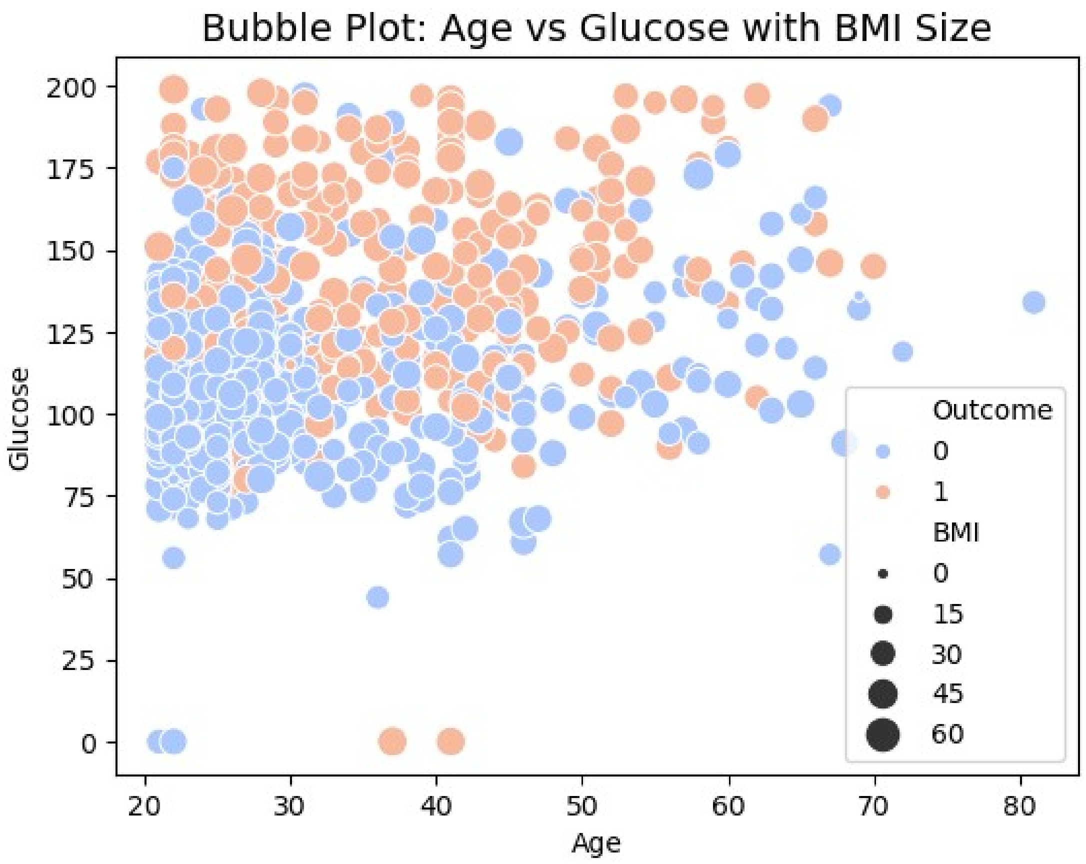
Regression Plot showing relationship between BMI and Age

```
sns.regplot(x='BMI', y='Age', data=dataset, scatter_kws={'s': 50},
line_kws={'color': 'red'})
plt.title('Regression Plot for BMI vs Age', fontsize=14)
plt.xlabel('BMI')
plt.ylabel('Age')
plt.show()
```



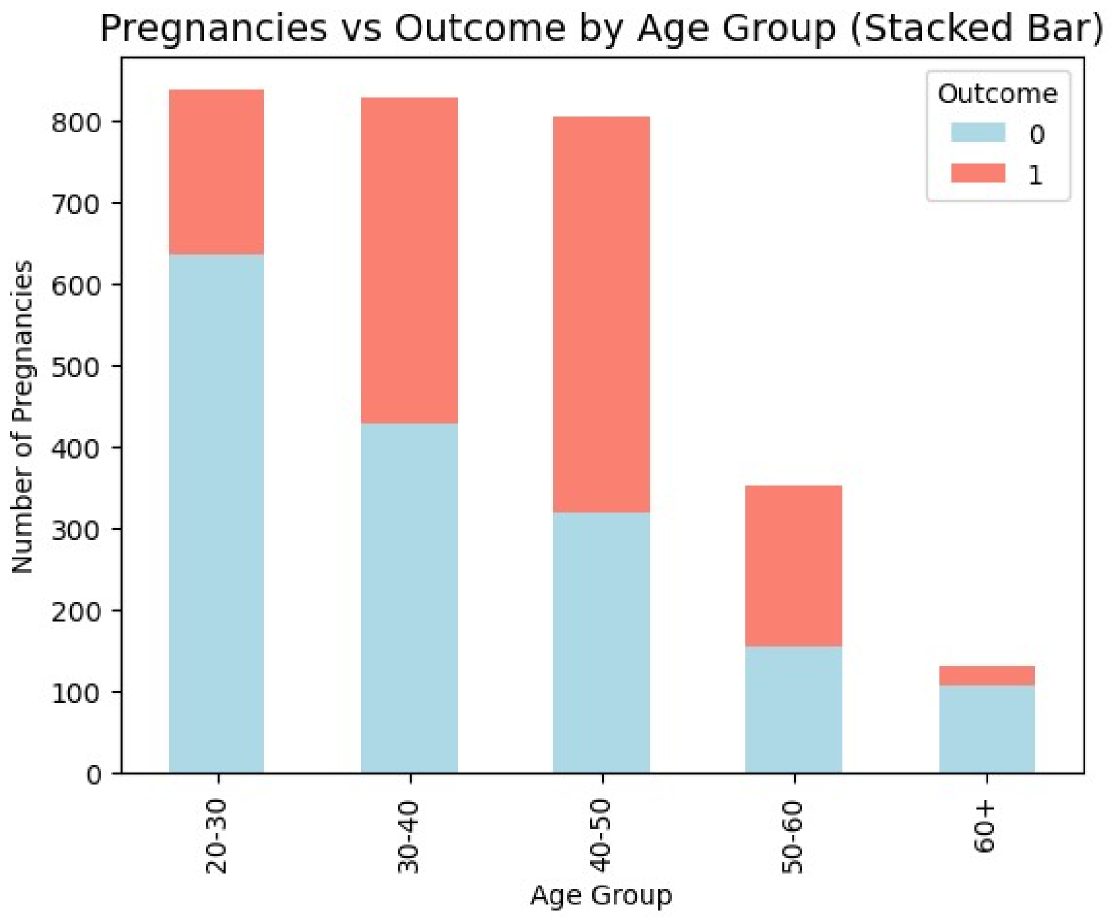
Bubble Plot showing Age vs Glucose with the size of points representing BMI

```
sns.scatterplot(x='Age', y='Glucose', size='BMI', data=dataset,
sizes=(20, 200), hue='Outcome', palette='coolwarm')
plt.title('Bubble Plot: Age vs Glucose with BMI Size', fontsize=14)
plt.xlabel('Age')
plt.ylabel('Glucose')
plt.show()
```



Stacked Bar Chart for Pregnancies vs Outcome by Age range

```
dataset['Age Group'] = pd.cut(dataset['Age'], bins=[20, 30, 40, 50,
60, 100], labels=['20-30', '30-40', '40-50', '50-60', '60+'])
age_outcome_pivot = dataset.pivot_table(index='Age Group',
columns='Outcome', values='Pregnancies', aggfunc='sum')
age_outcome_pivot.plot(kind='bar', stacked=True, color=['lightblue',
'salmon'])
plt.title('Pregnancies vs Outcome by Age Group (Stacked Bar)',
fontsize=14)
plt.xlabel('Age Group')
plt.ylabel('Number of Pregnancies')
plt.show()
```

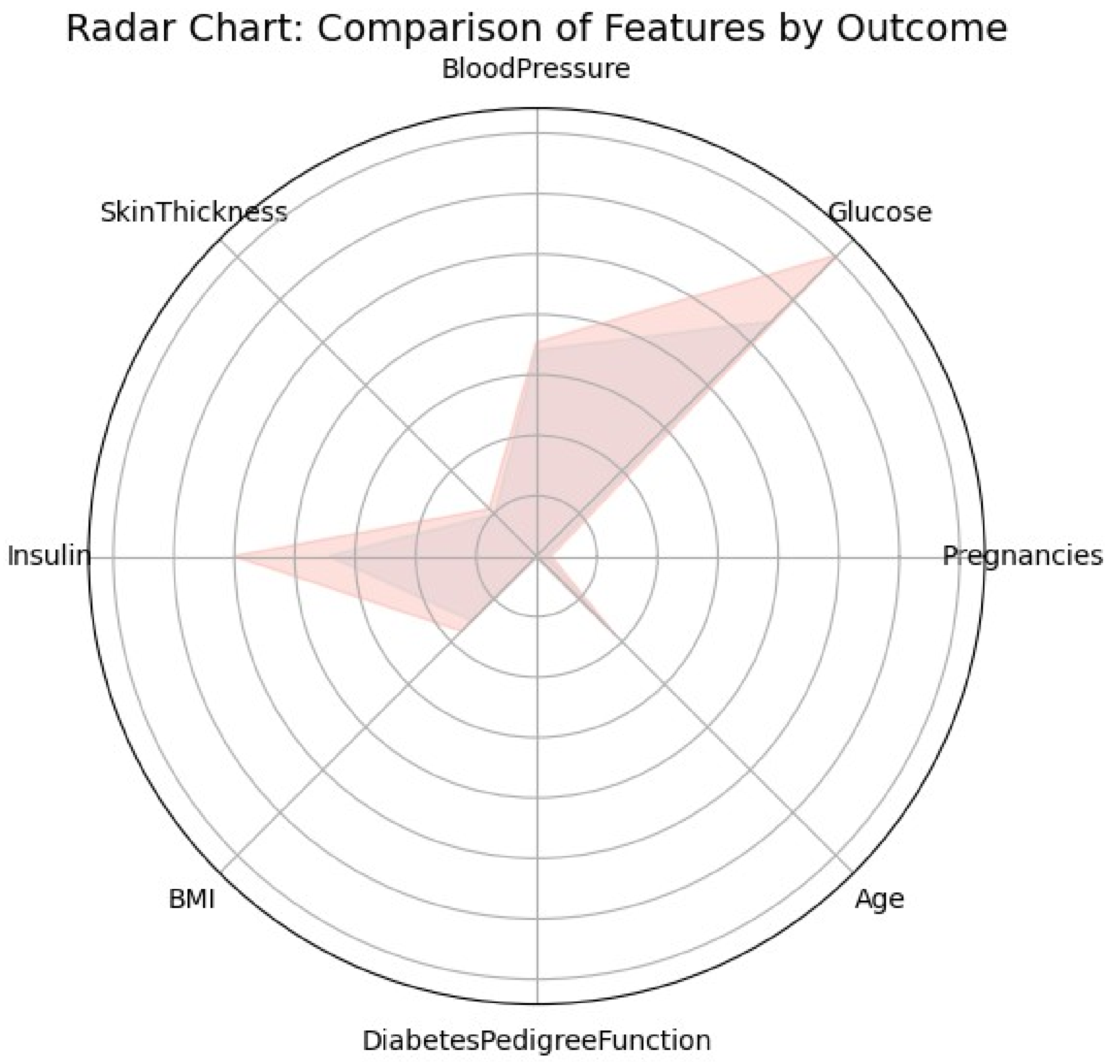


Radar Chart

Not Important

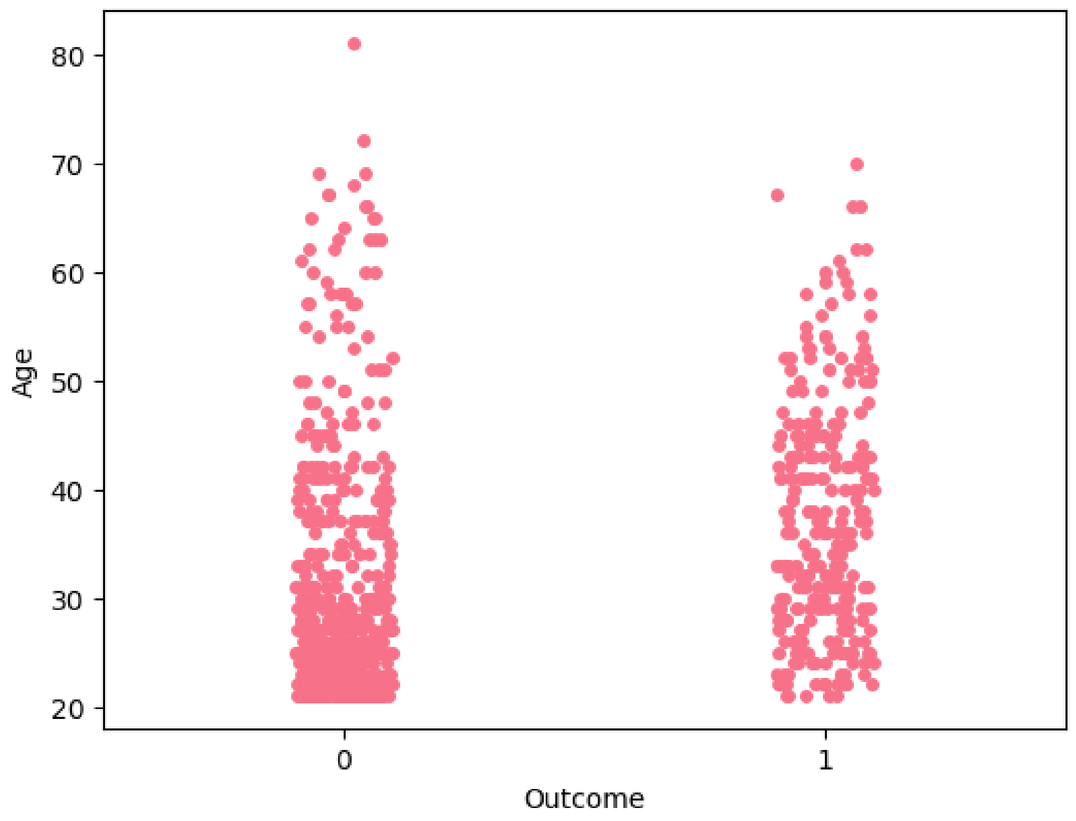
```
import numpy as np
from math import pi

# Categories to be plotted
categories = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

# Calculate mean values for each feature in the 'Outcome' categories
no_diabetes = dataset[dataset['Outcome'] == 0]
[categories].mean().values
diabetes = dataset[dataset['Outcome'] == 1][categories].mean().values

# Add the first value to the end to close the radar chart loop
values_no_diabetes = np.concatenate((no_diabetes, [no_diabetes[0]]))
values_diabetes = np.concatenate((diabetes, [diabetes[0]]))
```

```
# Compute the angles for the radar chart
angles = [n / float(len(categories)) * 2 * pi for n in
range(len(categories))]
angles += angles[:1]  # Close the loop

# Create the radar chart
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

# Plot the data
ax.fill(angles, values_no_diabetes, color='lightblue', alpha=0.25)
ax.fill(angles, values_diabetes, color='salmon', alpha=0.25)

# Add labels
ax.set_yticklabels([])  # Hide y-axis ticks
ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories, fontsize=10)

# Set the title
plt.title('Radar Chart: Comparison of Features by Outcome',
fontsize=14)

# Show the plot
plt.show()
```



Strip plot to show individual data points of 'Age' by 'Outcome'

```
sns.stripplot(x='Outcome', y='Age', data=dataset, jitter=True)
plt.show()
```


80
70
60
50
40
30
20
Age
0
1
Outcome