

Diabetes Prediction using Machine Learning



Importing Necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings  
warnings.filterwarnings('ignore')
```

Data Collection

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

EDA

```
df.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
```

```
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
```

```
df.tail()
```

```
Pregnancies Glucose BloodPressure SkinThickness Insulin BMI  
\  
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  
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

```

df.shape
(768, 9)
df.columns
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

```

About the Dataset

Context

The dataset originates from the National Institute of Diabetes and Digestive and Kidney Diseases. The primary aim is to predict whether a patient has diabetes based on several diagnostic measurements. This dataset is crucial for developing models that can assist in early diagnosis and intervention for diabetes, potentially improving patient outcomes and reducing healthcare costs.

Content

The dataset includes several constraints and specifics regarding the patient selection:

- **Patient Demographics:**
 - All patients are females.
 - Patients are at least 21 years old.
 - All patients are of Pima Indian heritage.

The dataset comprises the following features:

1. **Pregnancies:**
 - **Description:** Number of times the patient has been pregnant.
 - **Type:** Numeric (Integer)
 - **Relevance:** Previous pregnancies can impact the likelihood of diabetes due to physiological changes during pregnancy.
2. **Glucose:**
 - **Description:** Plasma glucose concentration measured two hours after an oral glucose tolerance test.
 - **Type:** Numeric (Float)
 - **Relevance:** High plasma glucose levels are a key indicator of diabetes.
3. **BloodPressure:**
 - **Description:** Diastolic blood pressure in millimeters of mercury (mm Hg).

- **Type:** Numeric (Float)
 - **Relevance:** Blood pressure levels can be indicative of overall cardiovascular health, which is often linked to diabetes.
4. **SkinThickness:**
- **Description:** Triceps skin fold thickness in millimeters.
 - **Type:** Numeric (Float)
 - **Relevance:** This measurement can indicate subcutaneous fat levels, which may be associated with insulin resistance and diabetes.
5. **Insulin:**
- **Description:** 2-hour serum insulin in micro-units per milliliter (mu U/ml).
 - **Type:** Numeric (Float)
 - **Relevance:** Insulin levels are crucial for understanding insulin resistance, a precursor to diabetes.
6. **BMI (Body Mass Index):**
- **Description:** Weight in kilograms divided by the square of height in meters (kg/m^2).
 - **Type:** Numeric (Float)
 - **Relevance:** BMI is a significant indicator of obesity, which is a major risk factor for diabetes.
7. **DiabetesPedigreeFunction:**
- **Description:** A function that scores the likelihood of diabetes based on family history.
 - **Type:** Numeric (Float)
 - **Relevance:** Family history is a strong predictor of diabetes risk.
8. **Age:**
- **Description:** Age of the patient in years.
 - **Type:** Numeric (Integer)
 - **Relevance:** Age is a risk factor, as the likelihood of developing diabetes increases with age.
9. **Outcome:**
- **Description:** Class variable indicating the presence of diabetes.
 - 0: Non-diabetic
 - 1: Diabetic
 - **Type:** Categorical (Binary)
 - **Relevance:** This is the target variable that models aim to predict.

Data Summary

The dataset is structured to include comprehensive features that can contribute to the predictive modeling of diabetes. By analyzing these features, one can develop various machine learning models to classify patients based on their likelihood of having diabetes. The features represent a mix of personal medical history, physiological measurements, and family history, all of which are critical in understanding and predicting diabetes.

```
df.duplicated().sum()
```



```

0

df.isnull().sum()

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

df.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

df.describe()

```

	Pregnancies	Glucose	BloodPressure	SkinThickness
Insulin \ count	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458
79.799479				
std	3.369578	31.972618	19.355807	15.952218
115.244002				
min	0.000000	0.000000	0.000000	0.000000
0.000000				
25%	1.000000	99.000000	62.000000	0.000000
0.000000				
50%	3.000000	117.000000	72.000000	23.000000
30.500000				
75%	6.000000	140.250000	80.000000	32.000000



```

127.250000
max      17.000000 199.000000      122.000000      99.000000
846.000000

          BMI  DiabetesPedigreeFunction      Age      Outcome
count  768.000000            768.000000  768.000000  768.000000
mean   31.992578            0.471876   33.240885  0.348958
std    7.884160            0.331329   11.760232  0.476951
min    0.000000            0.078000   21.000000  0.000000
25%   27.300000            0.243750   24.000000  0.000000
50%   32.000000            0.372500   29.000000  0.000000
75%   36.600000            0.626250   41.000000  1.000000
max   67.100000            2.420000   81.000000  1.000000

df.unique()

Pregnancies           17
Glucose                136
BloodPressure           47
SkinThickness           51
Insulin                 186
BMI                     248
DiabetesPedigreeFunction 517
Age                      52
Outcome                  2
dtype: int64

object_columns = df.select_dtypes(include=['object']).columns
print("Object type columns:")
print(object_columns)

numerical_columns = df.select_dtypes(include=['int64',
'float64']).columns
print("\nNumerical type columns:")
print(numerical_columns)

Object type columns:
Index([], dtype='object')

Numerical type columns:
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
dtype='object')

def classify_features(df):
    categorical_features = []
    non_categorical_features = []
    discrete_features = []
    continuous_features = []

```



```

for column in df.columns:
    if df[column].dtype == 'object':
        if df[column].nunique() < 20:
            categorical_features.append(column)
        else:
            non_categorical_features.append(column)
    elif df[column].dtype in ['int64', 'float64']:
        if df[column].nunique() < 10:
            discrete_features.append(column)
        else:
            continuous_features.append(column)

return categorical_features, non_categorical_features,
discrete_features, continuous_features

categorical, non_categorical, discrete, continuous =
classify_features(df)

print("Categorical Features:", categorical)
print("Non-Categorical Features:", non_categorical)
print("Discrete Features:", discrete)
print("Continuous Features:", continuous)

Categorical Features: []
Non-Categorical Features: []
Discrete Features: ['Outcome']
Continuous Features: ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

df['Outcome'].unique()

array([1, 0], dtype=int64)

df['Outcome'].value_counts()

Outcome
0    500
1    268
Name: count, dtype: int64

```

Data Visualization

```

for i in discrete:
    plt.figure(figsize=(15, 6))
    ax = sns.countplot(x=i, data=df, palette='hls')

    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}',


```

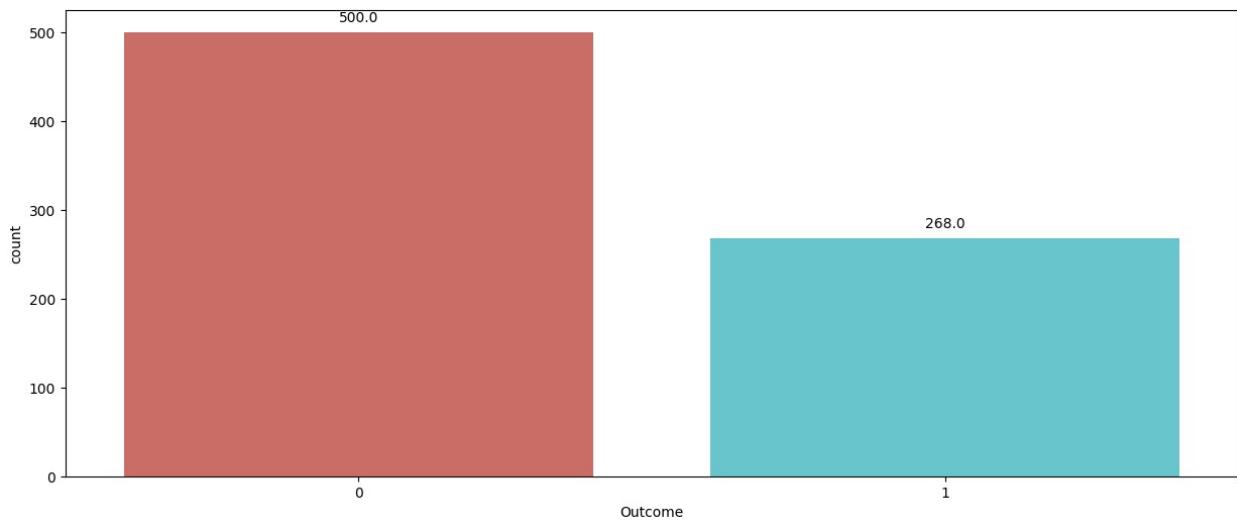


```

        xy=(p.get_x() + p.get_width() / 2., height),
        xytext=(0, 10),
        textcoords='offset points',
        ha='center', va='center')

plt.show()

```



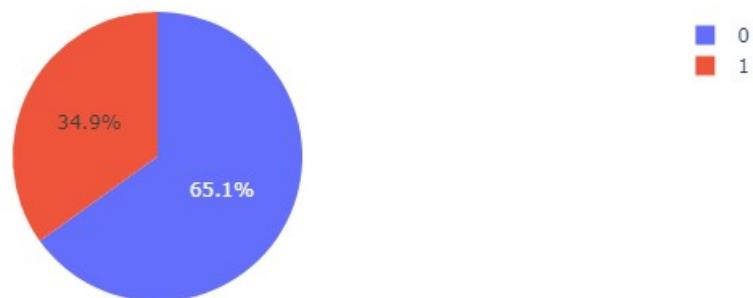
```

import plotly.express as px

for i in discrete:
    counts = df[i].value_counts()
    fig = px.pie(counts, values=counts.values, names=counts.index,
title=f'Distribution of {i}')
    fig.show()

```

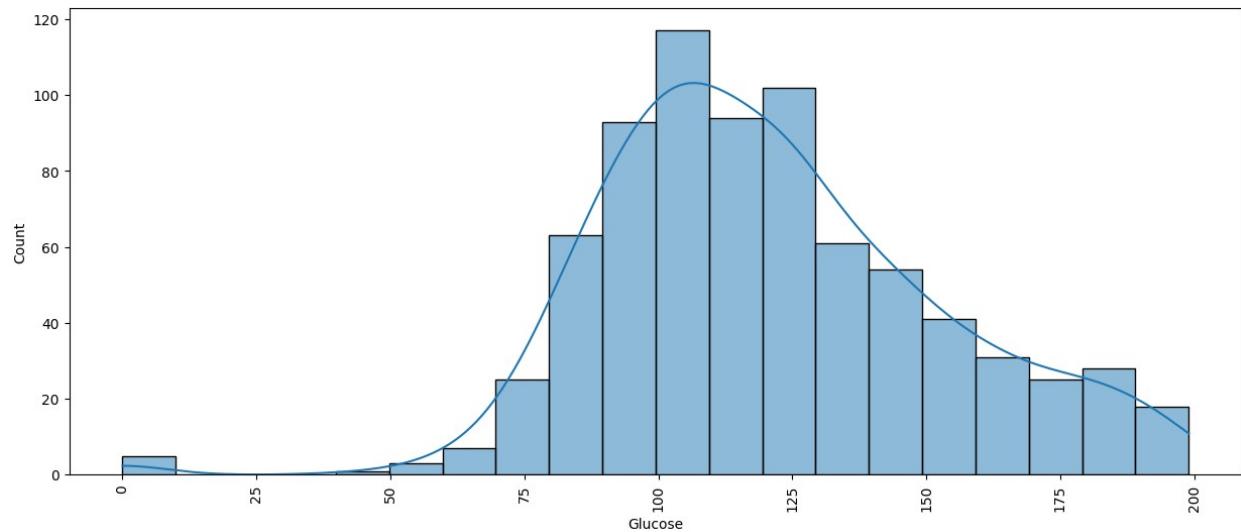
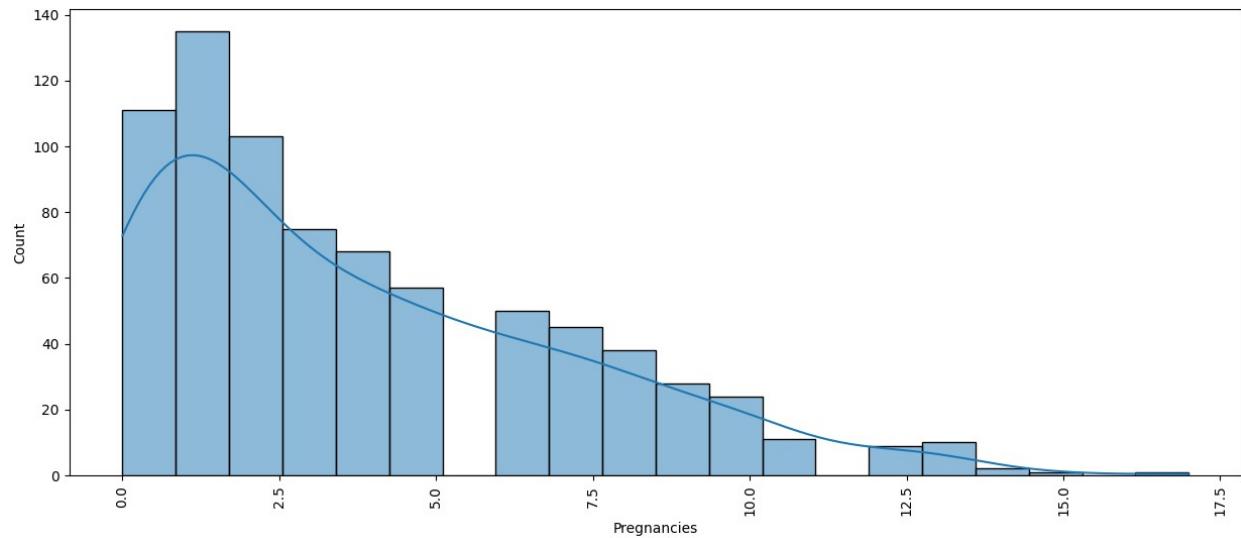
Distribution of Outcome

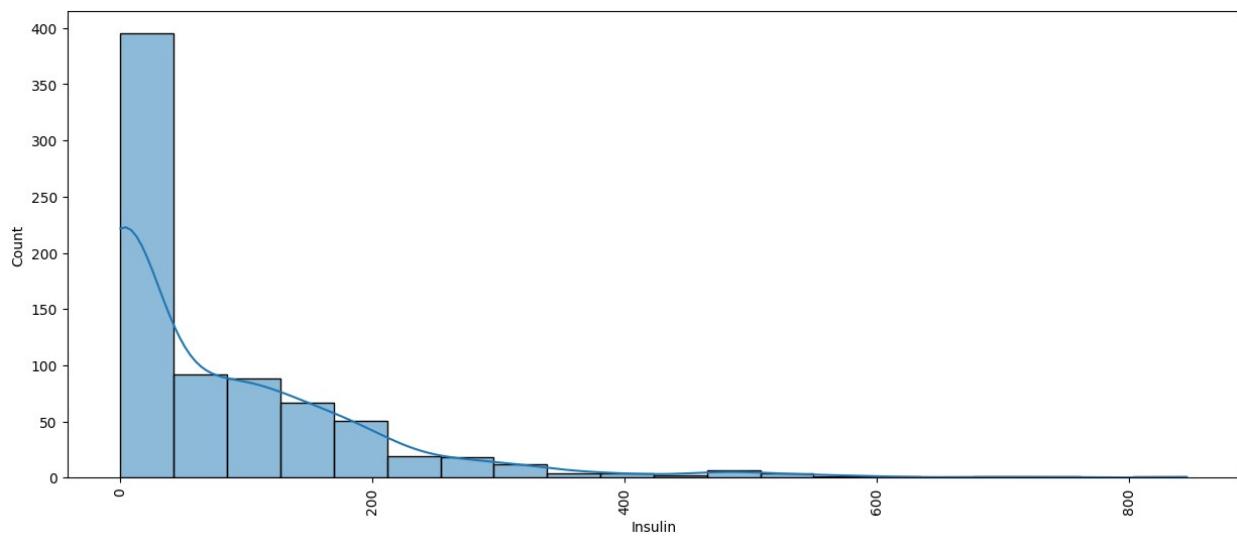
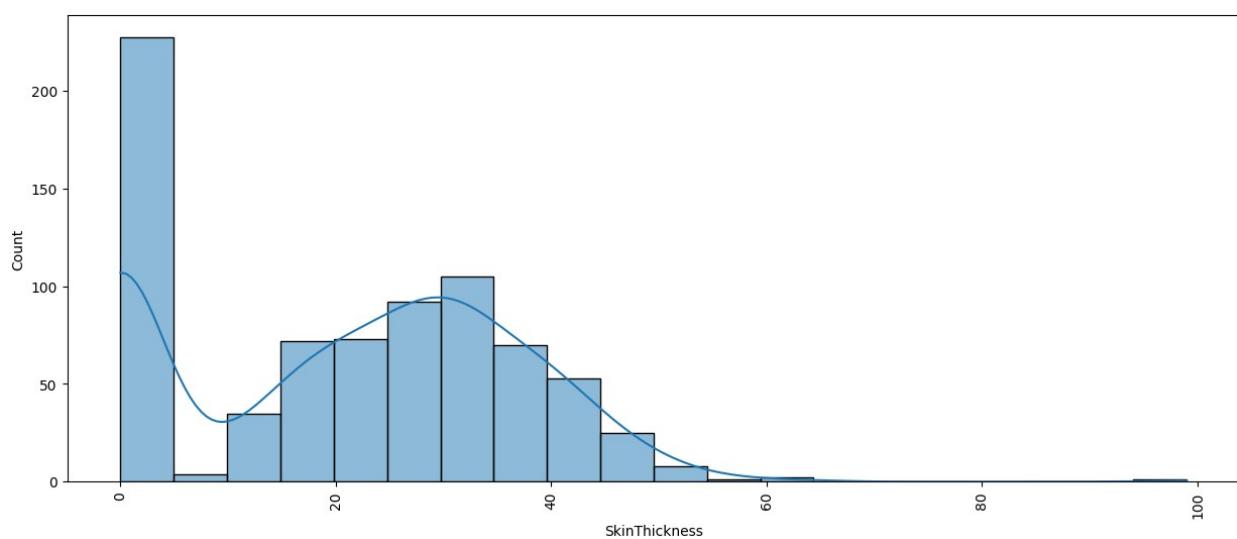
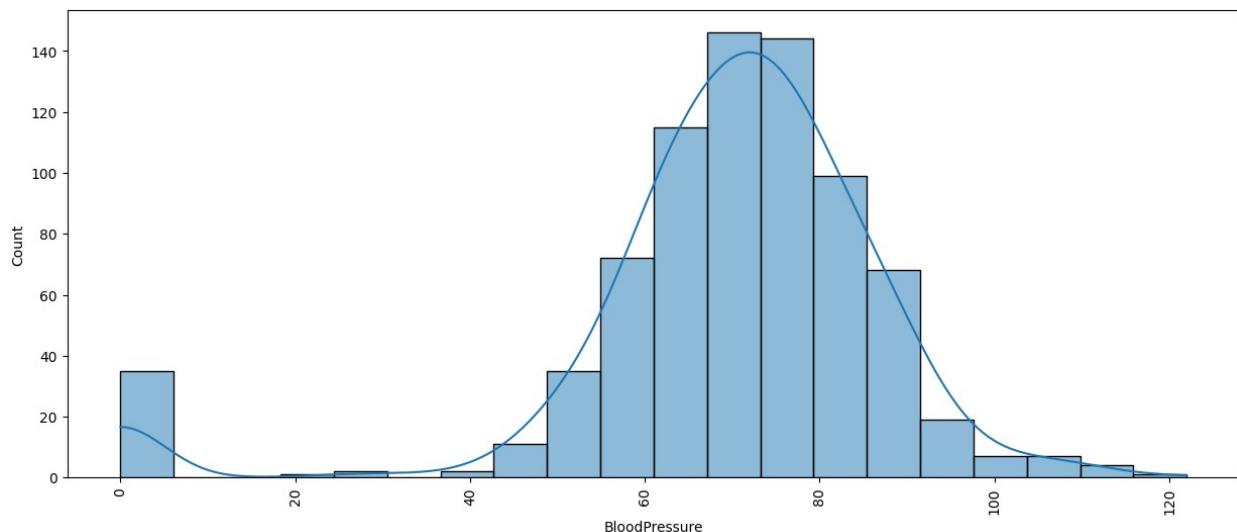


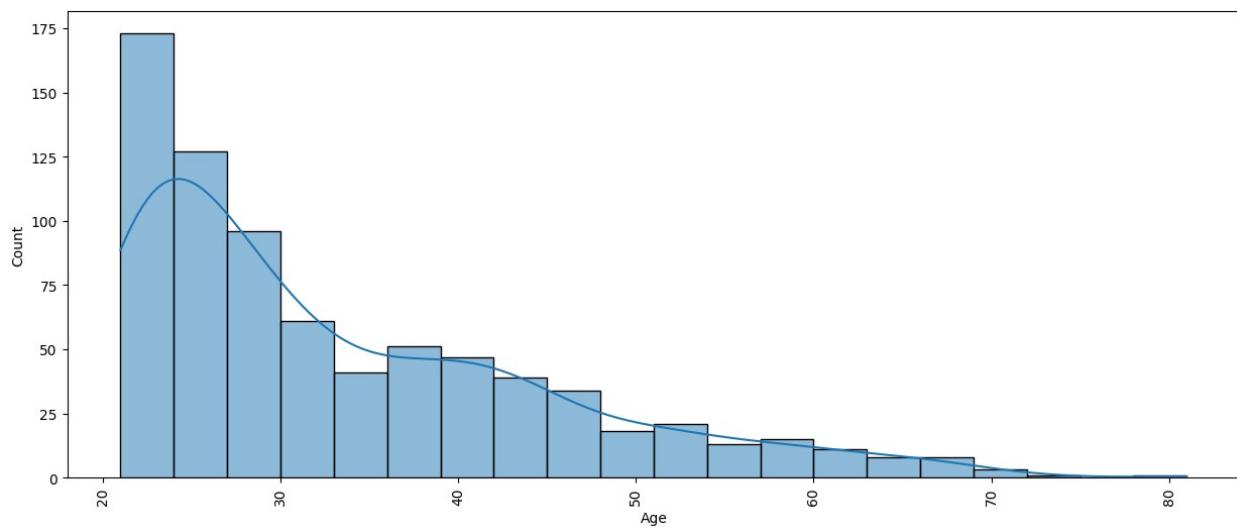
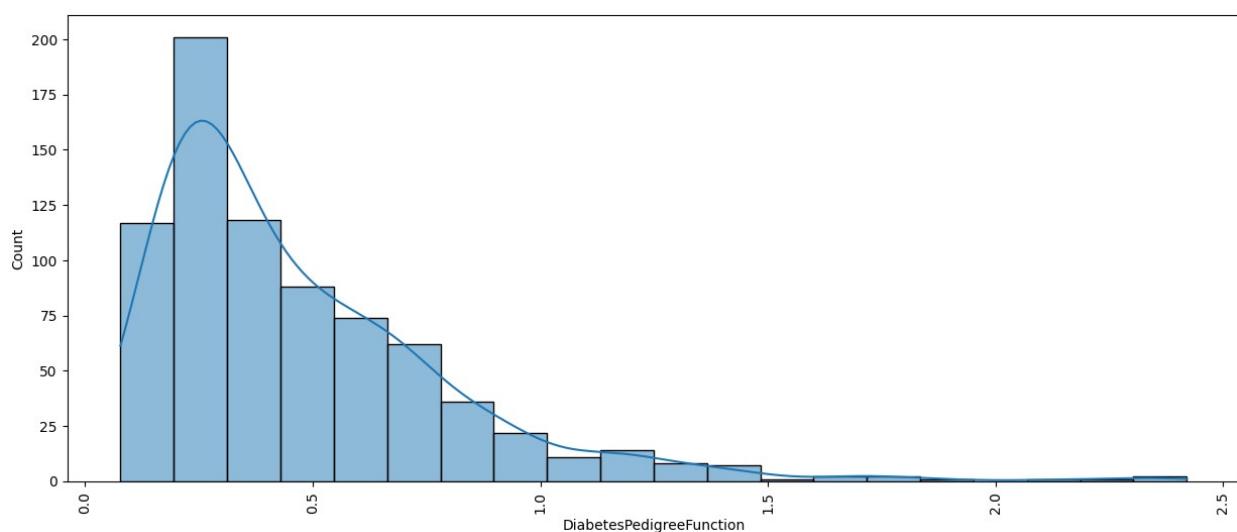
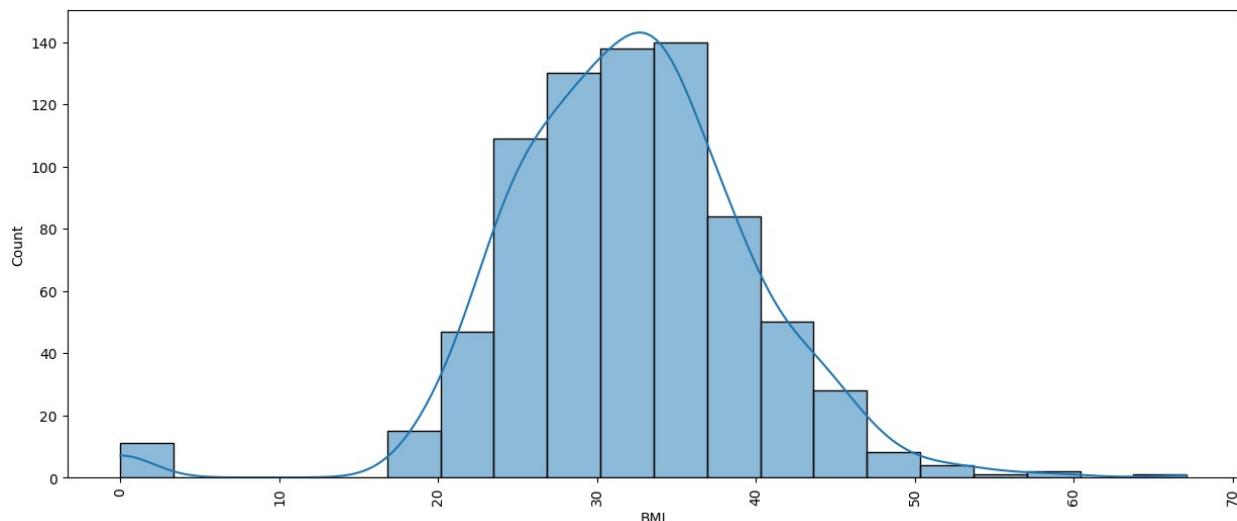
```

for i in continuous:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], bins = 20, kde = True, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()

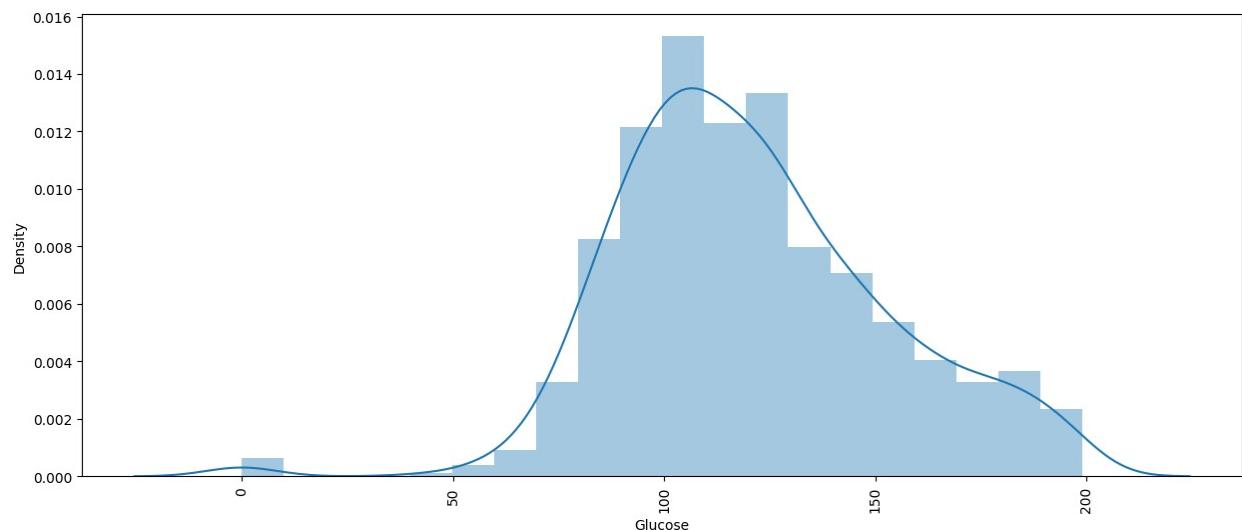
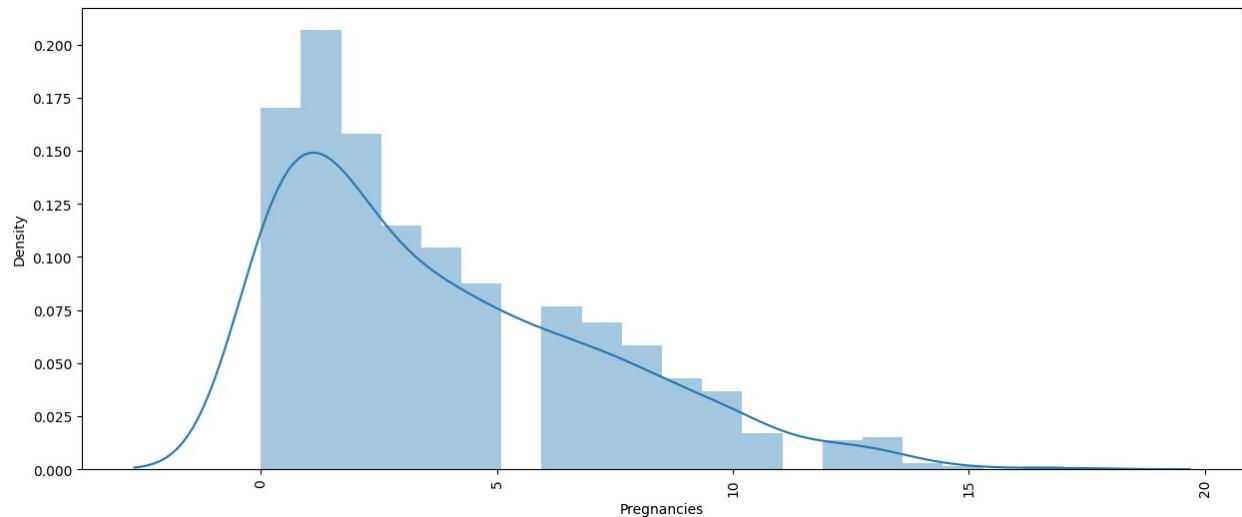
```

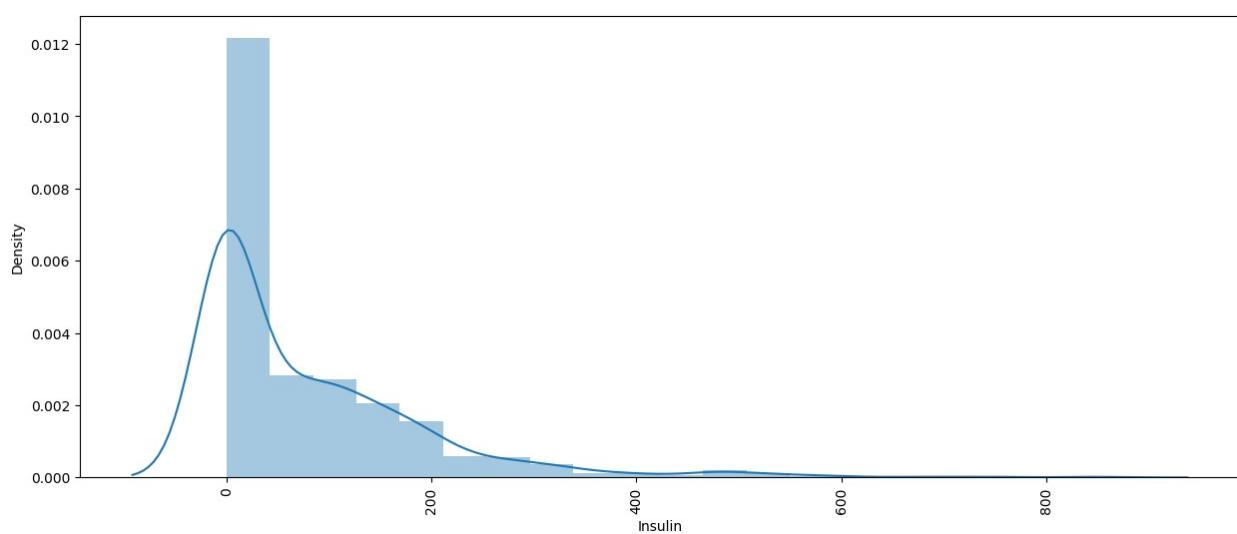
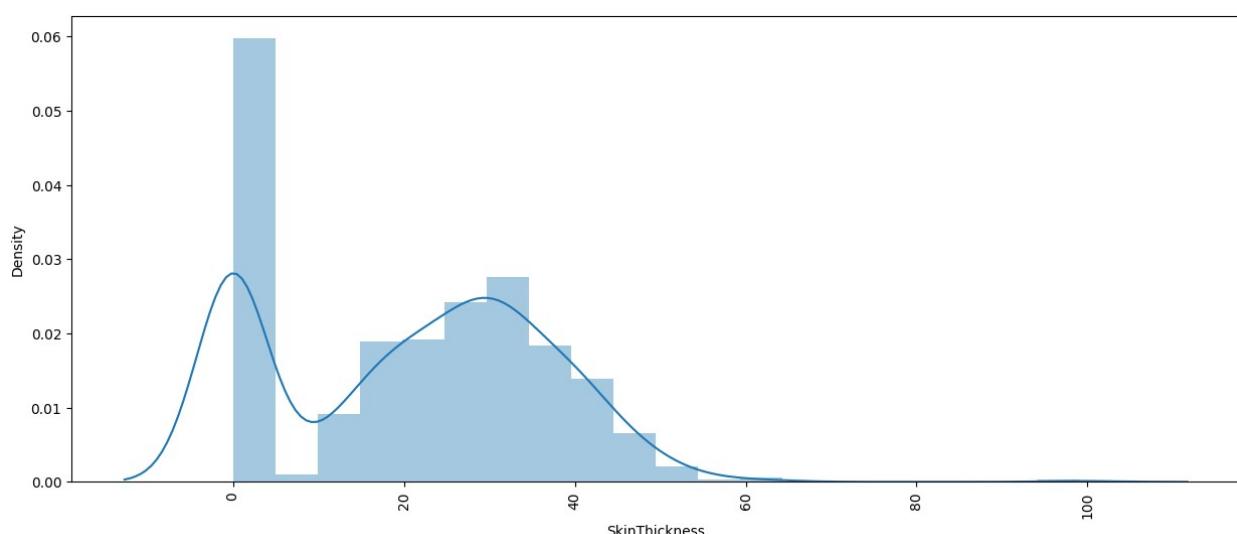
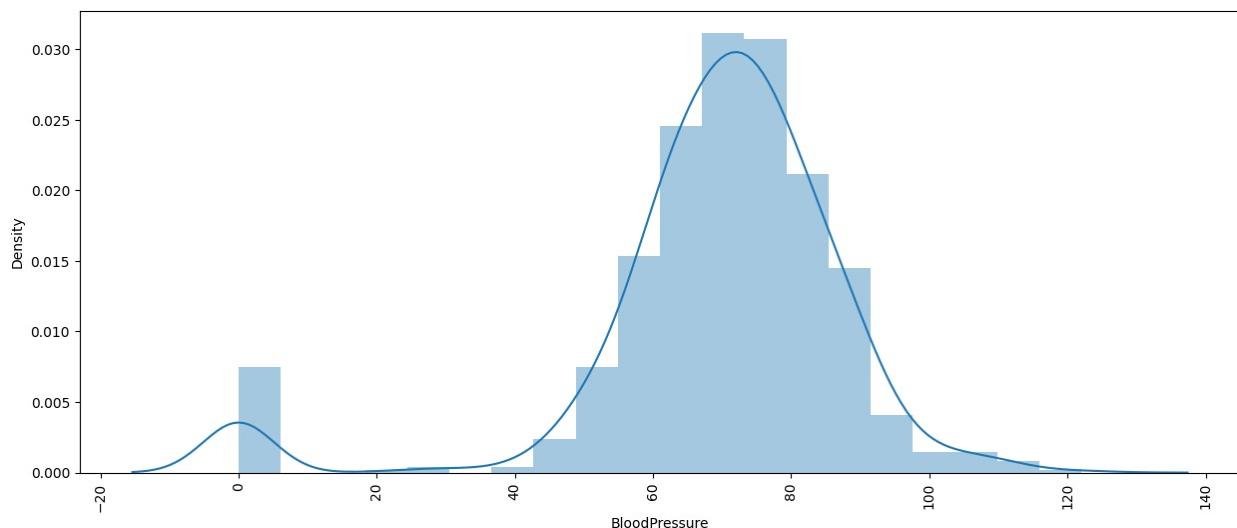


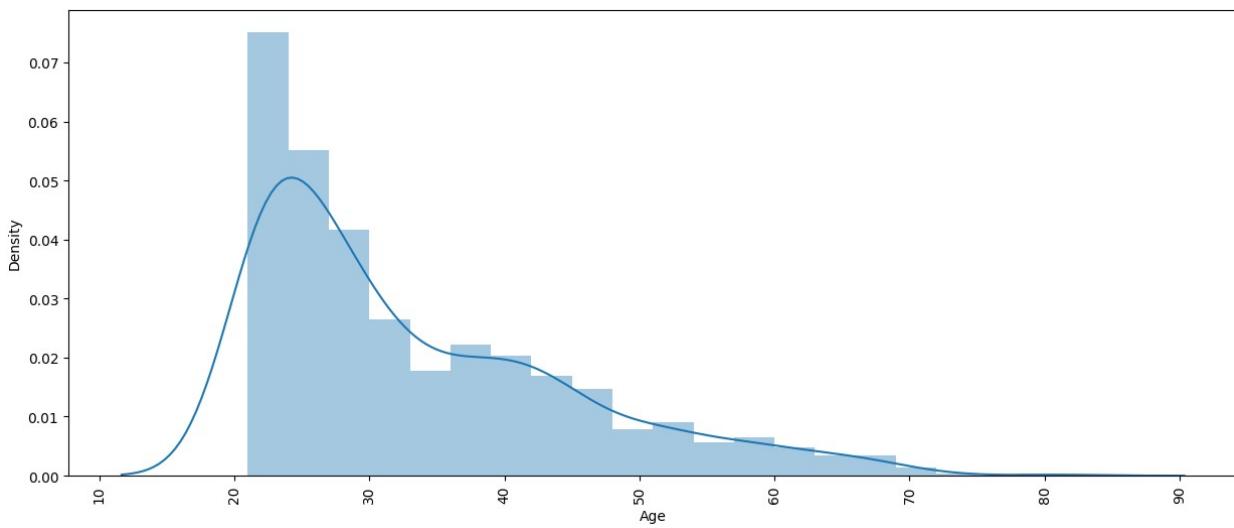
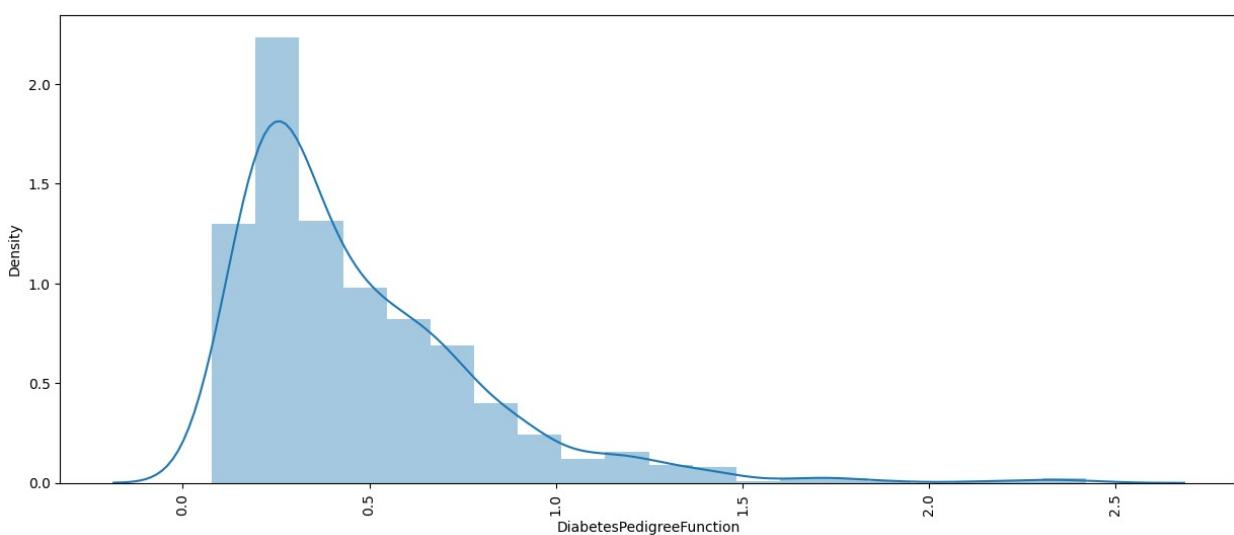
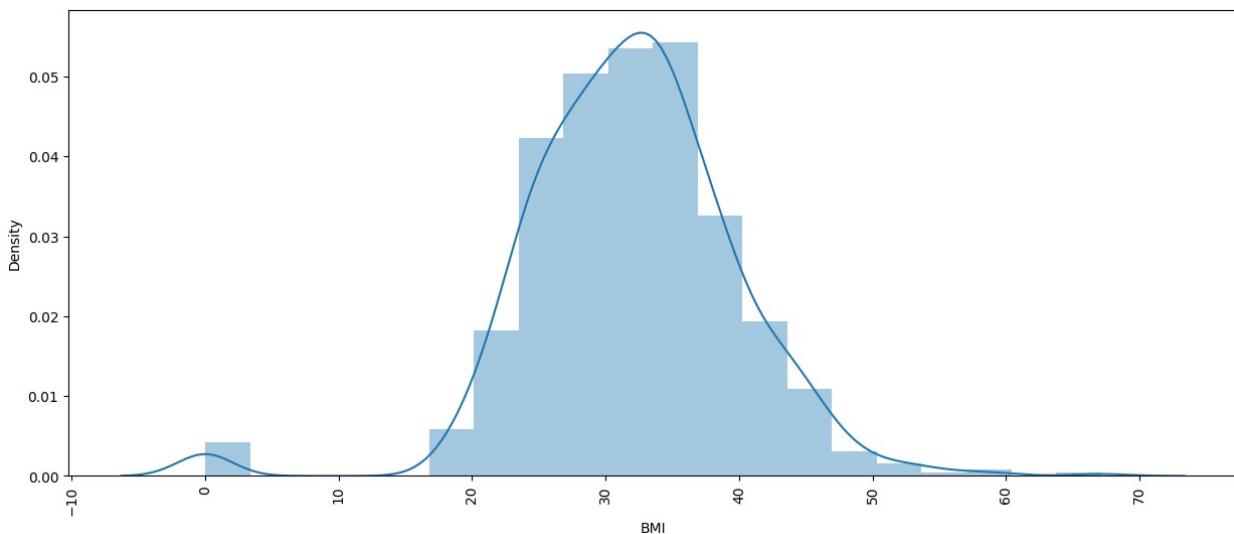




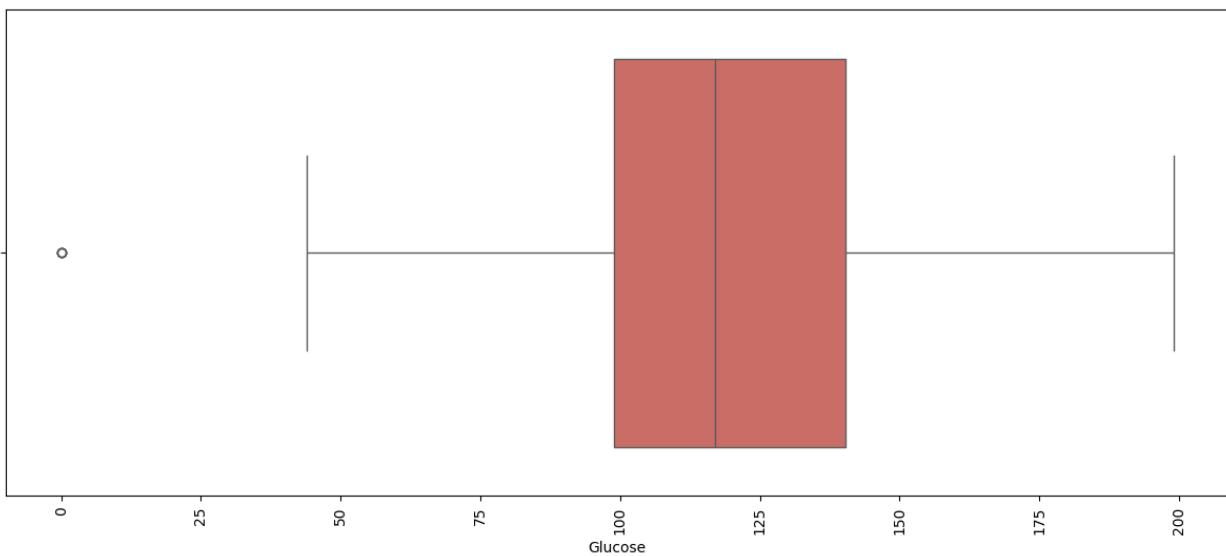
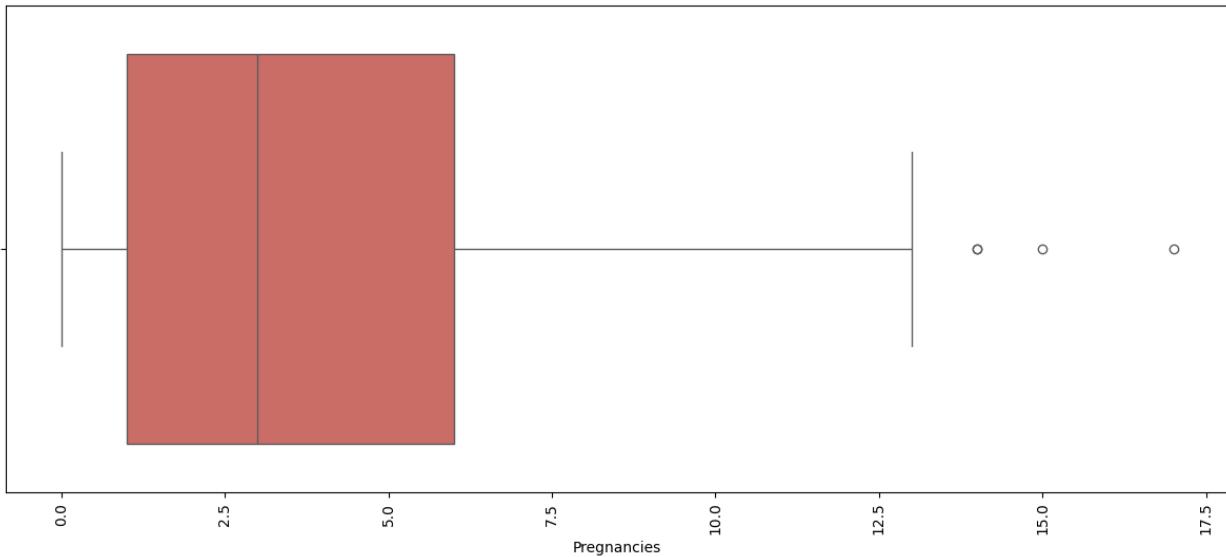
```
for i in continuous:  
    plt.figure(figsize=(15,6))  
    sns.distplot(df[i], bins = 20, kde = True)  
    plt.xticks(rotation = 90)  
    plt.show()
```

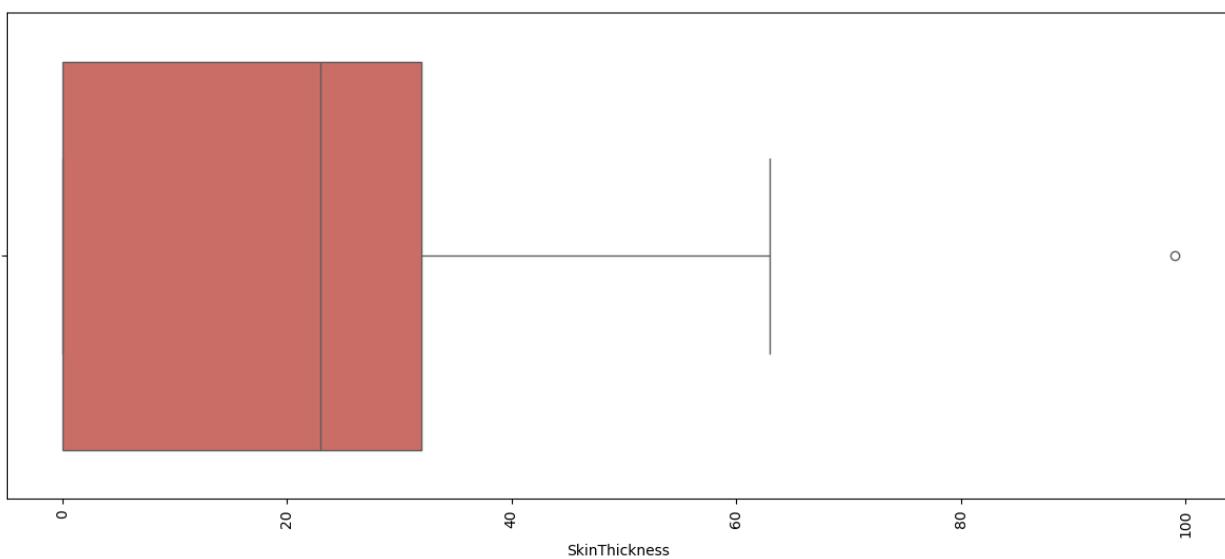
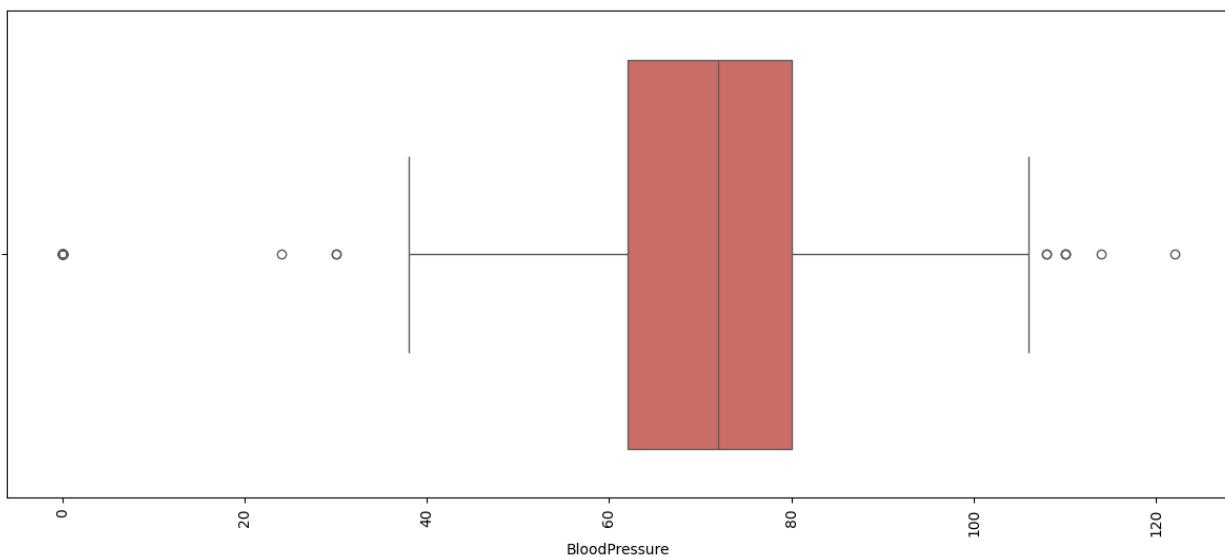


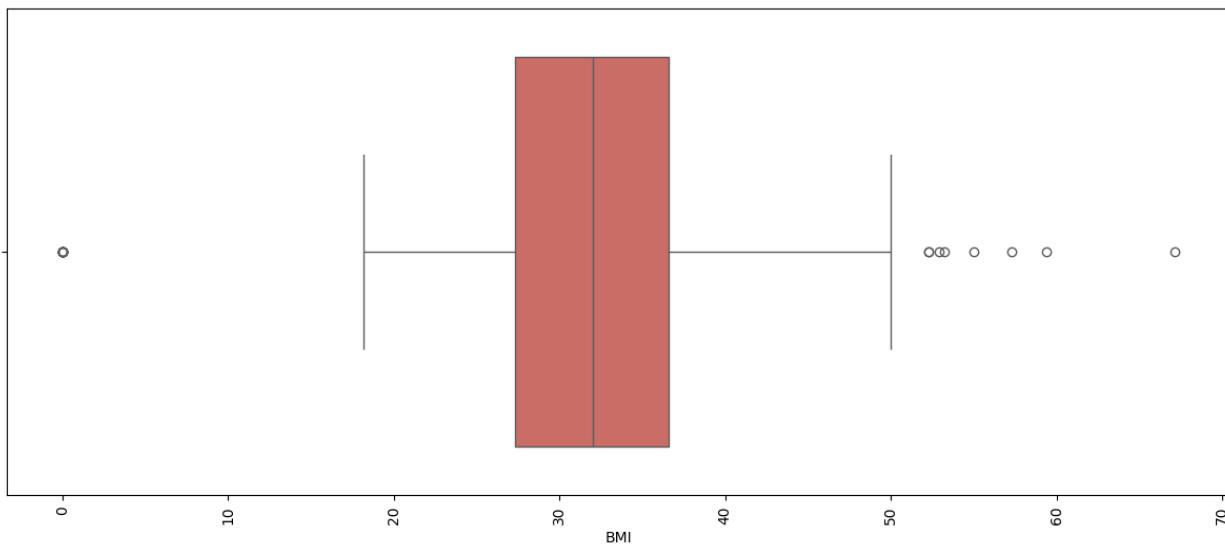
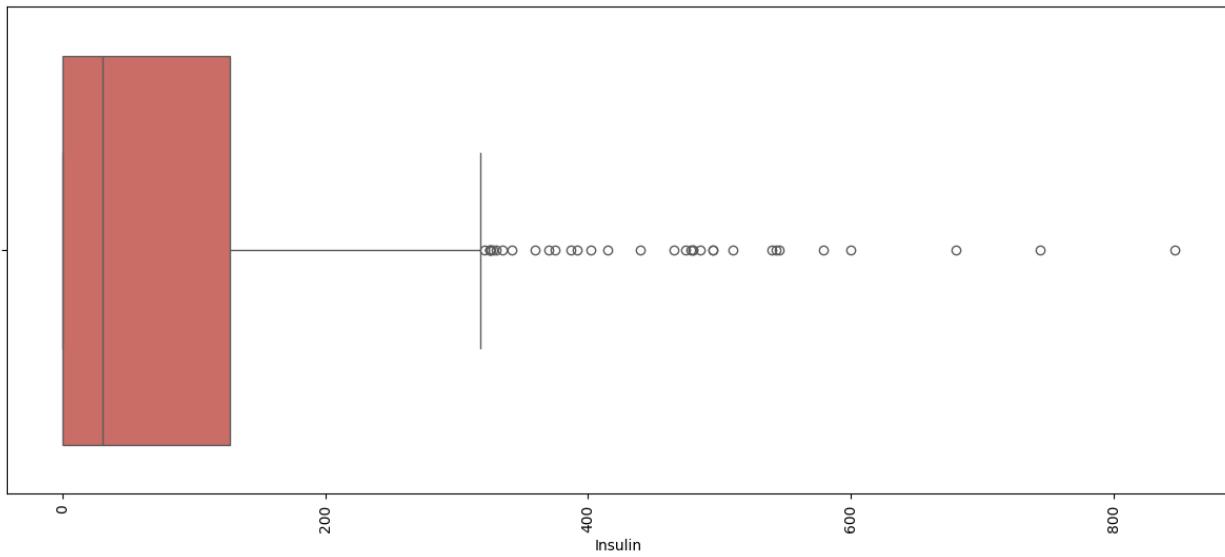


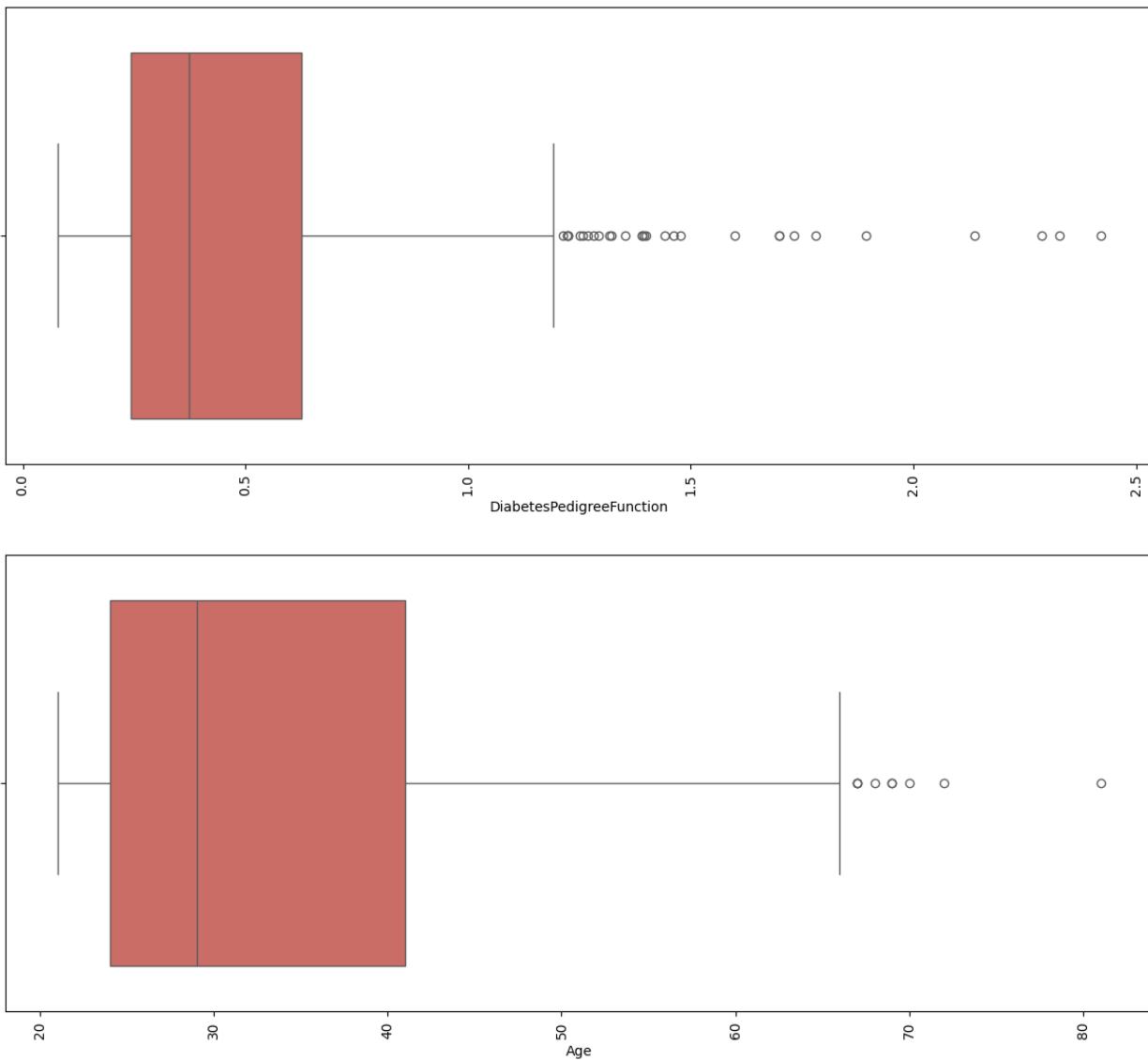


```
for i in continuous:  
    plt.figure(figsize=(15, 6))  
    sns.boxplot(x=i, data=df, palette='hls')  
    plt.xticks(rotation=90)  
    plt.show()
```

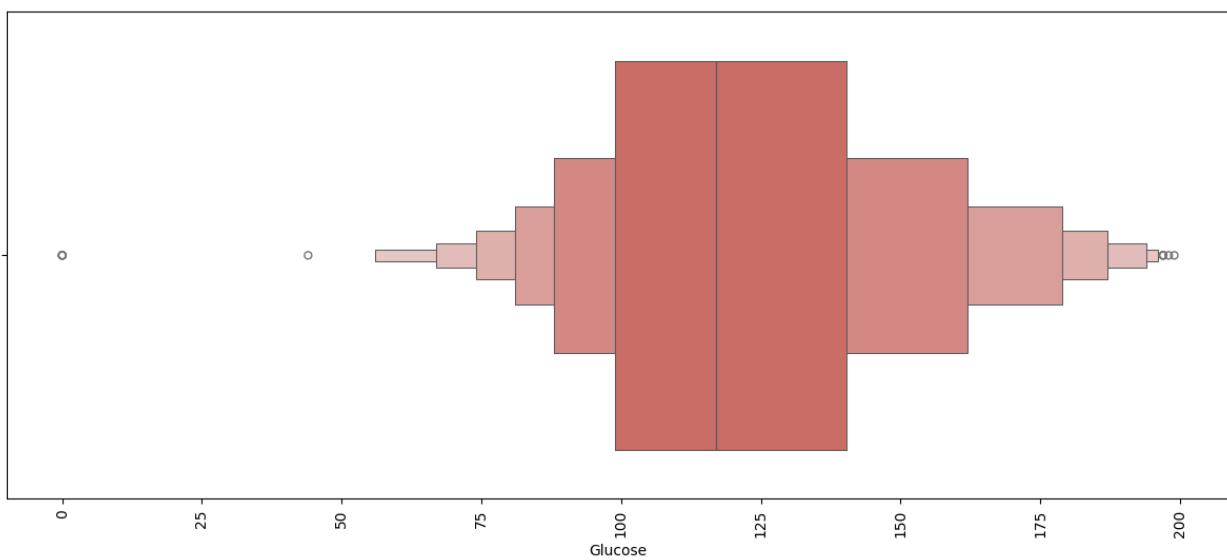
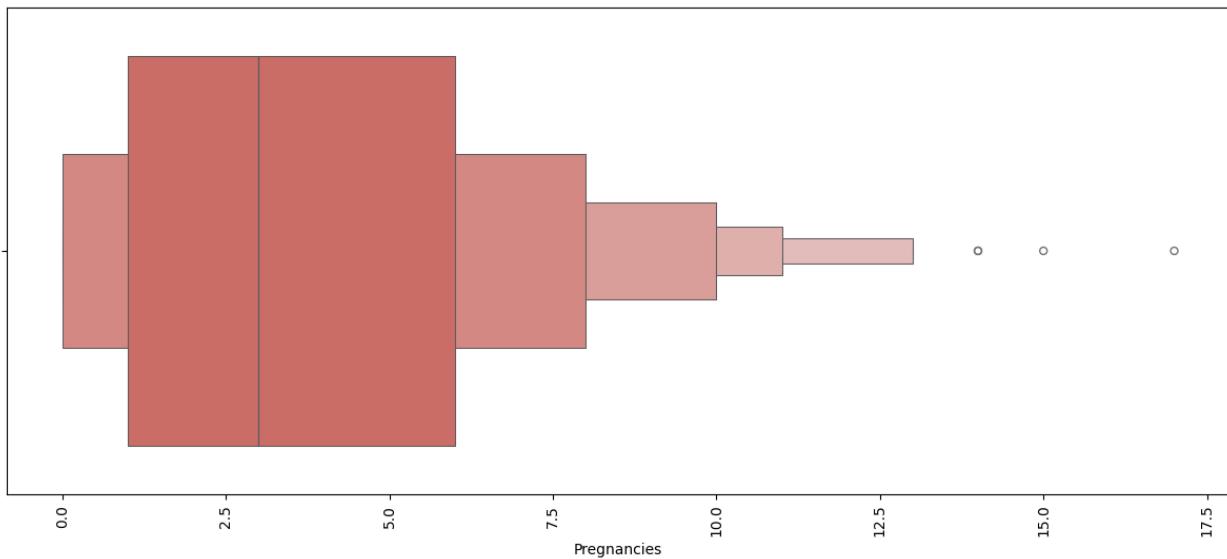


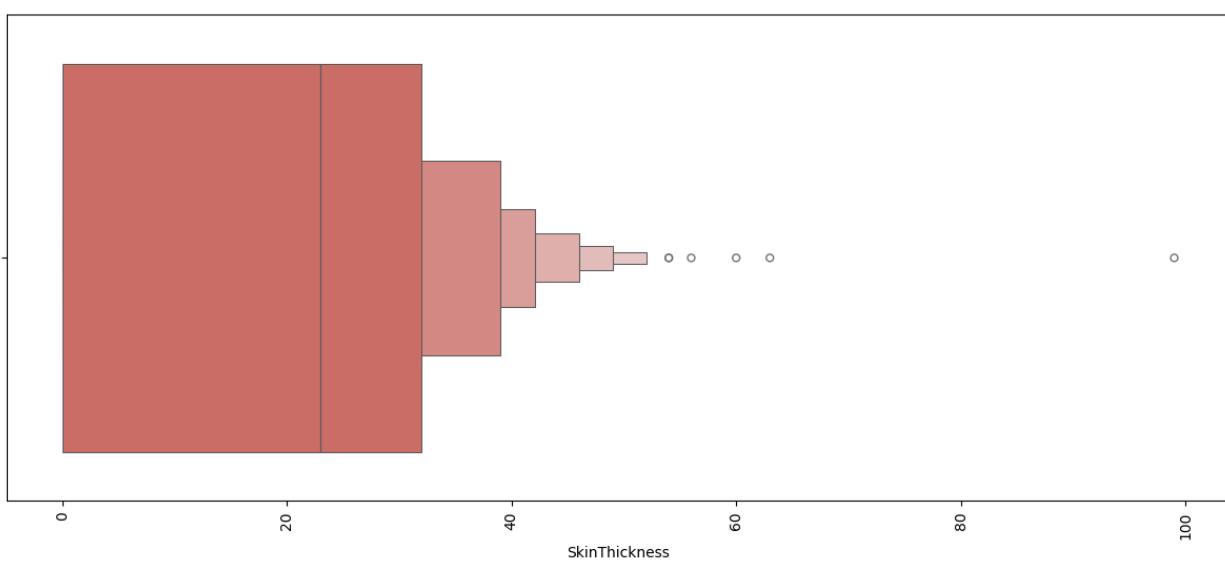
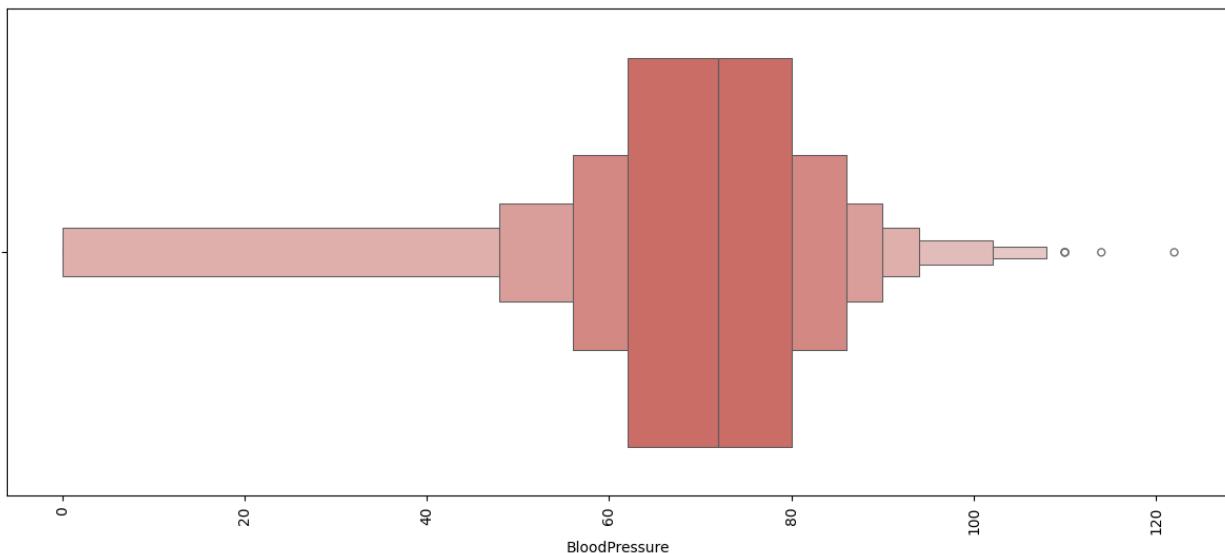


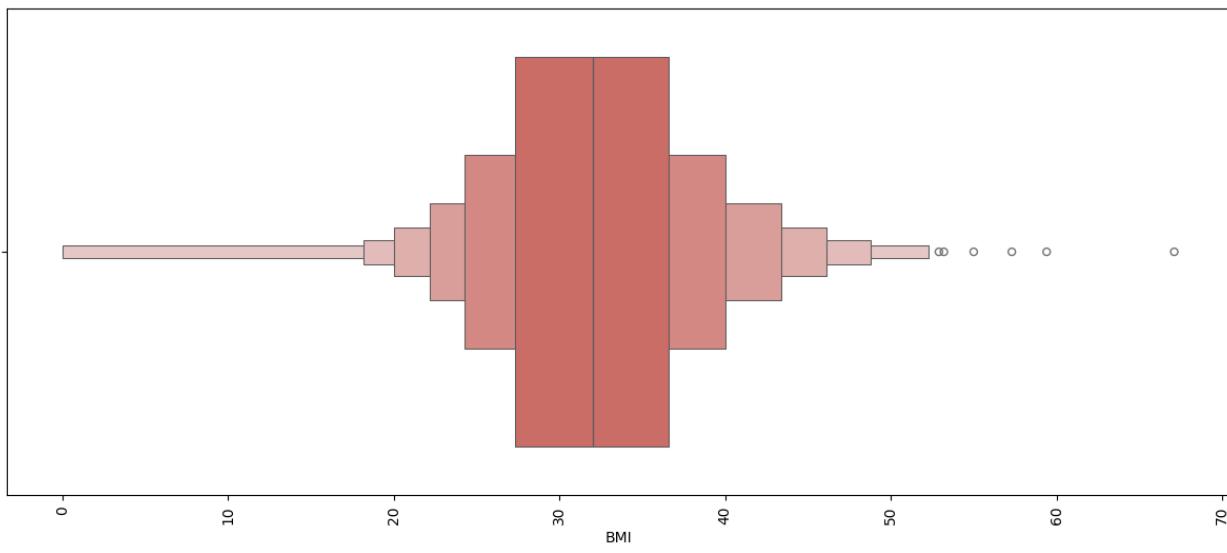
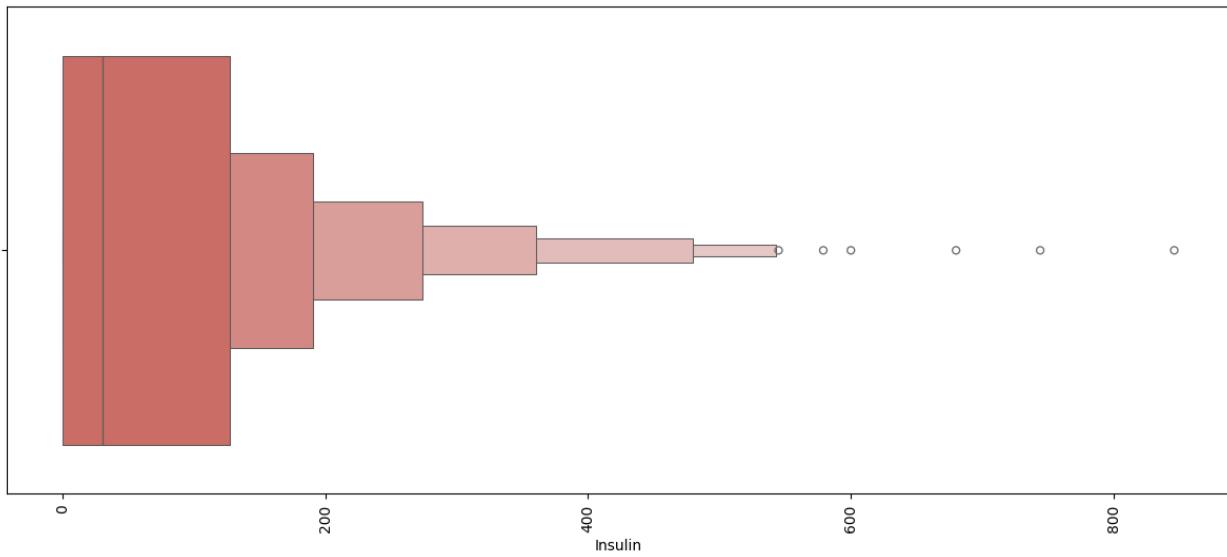


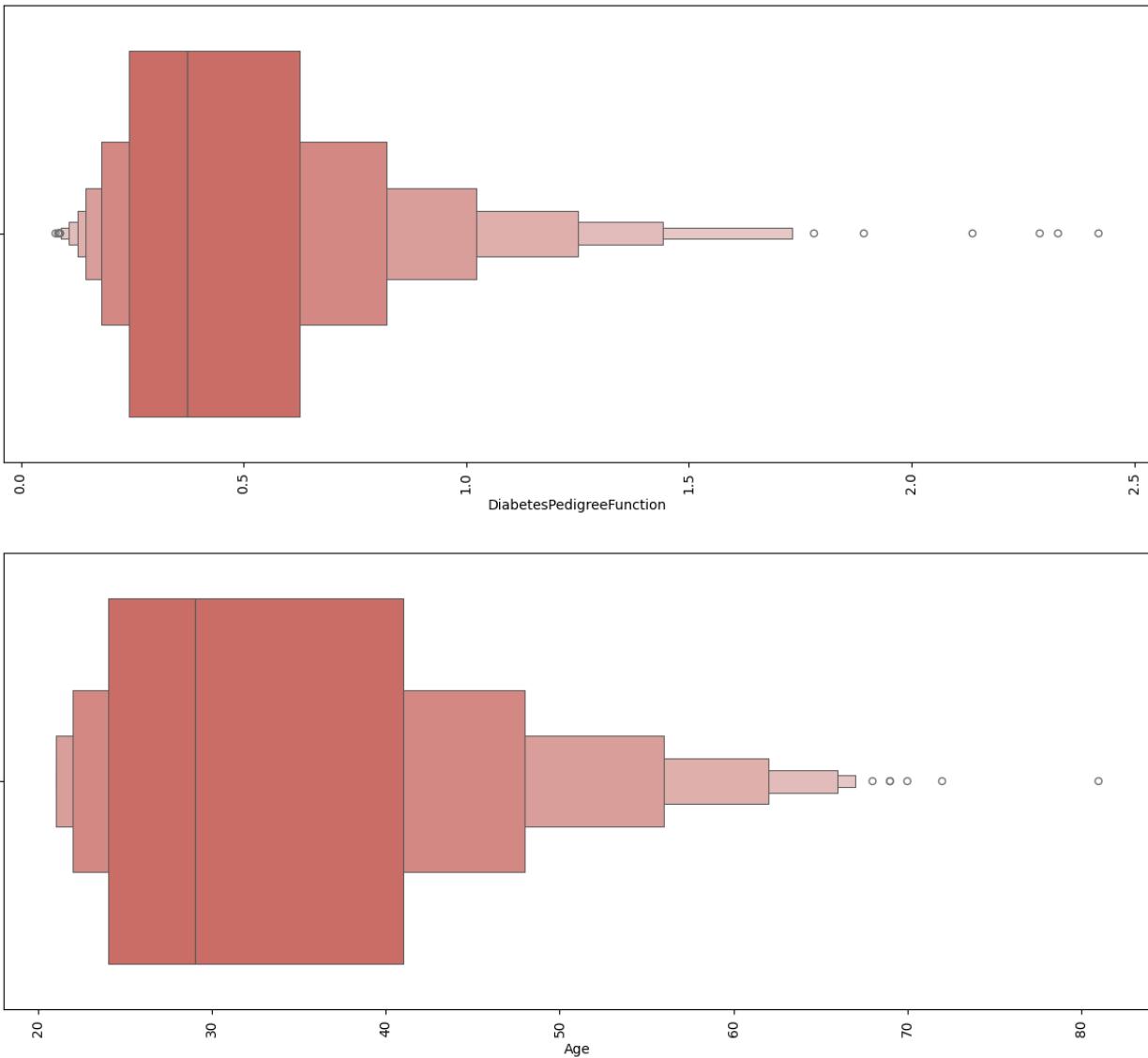


```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.boxenplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```

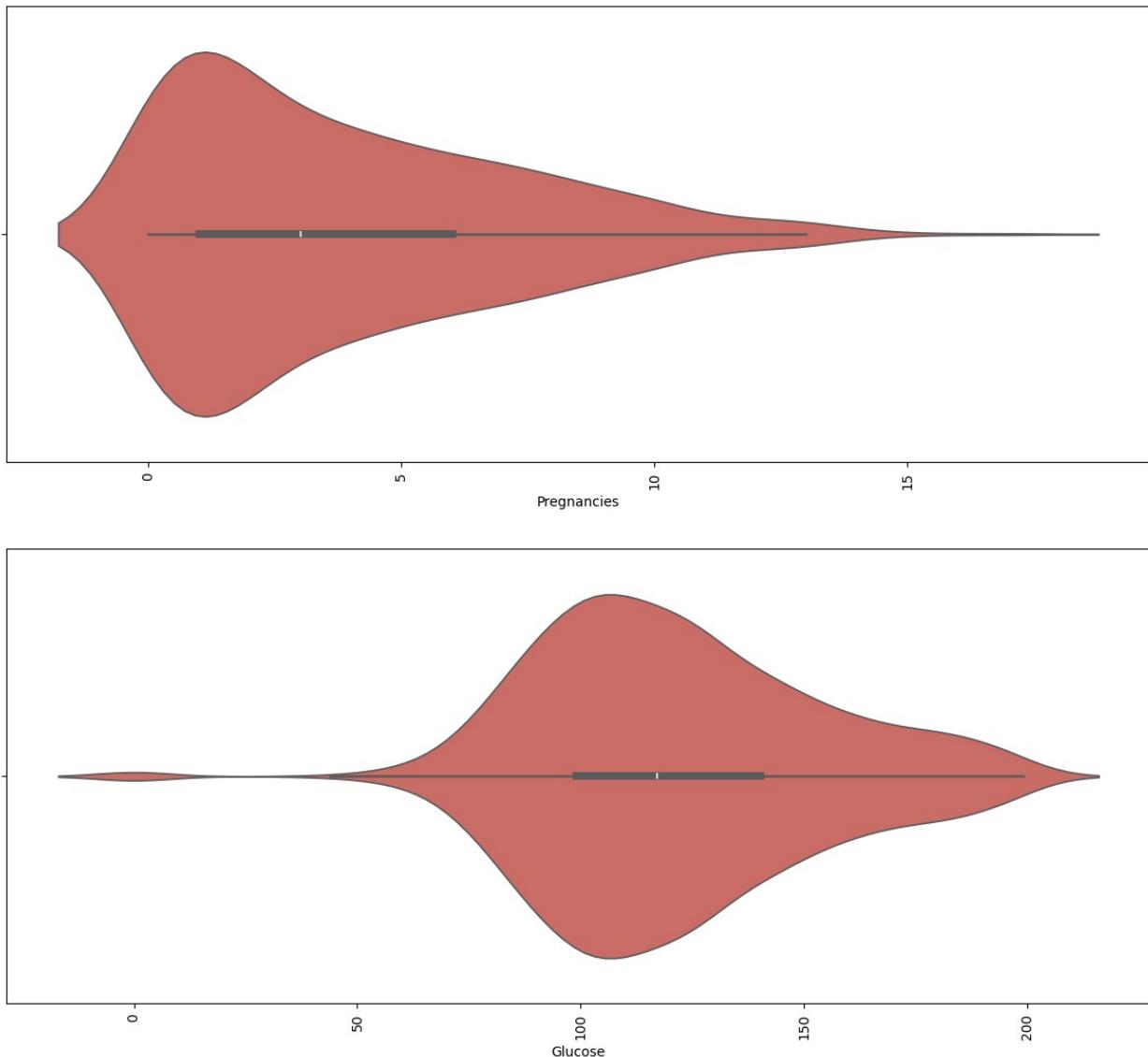


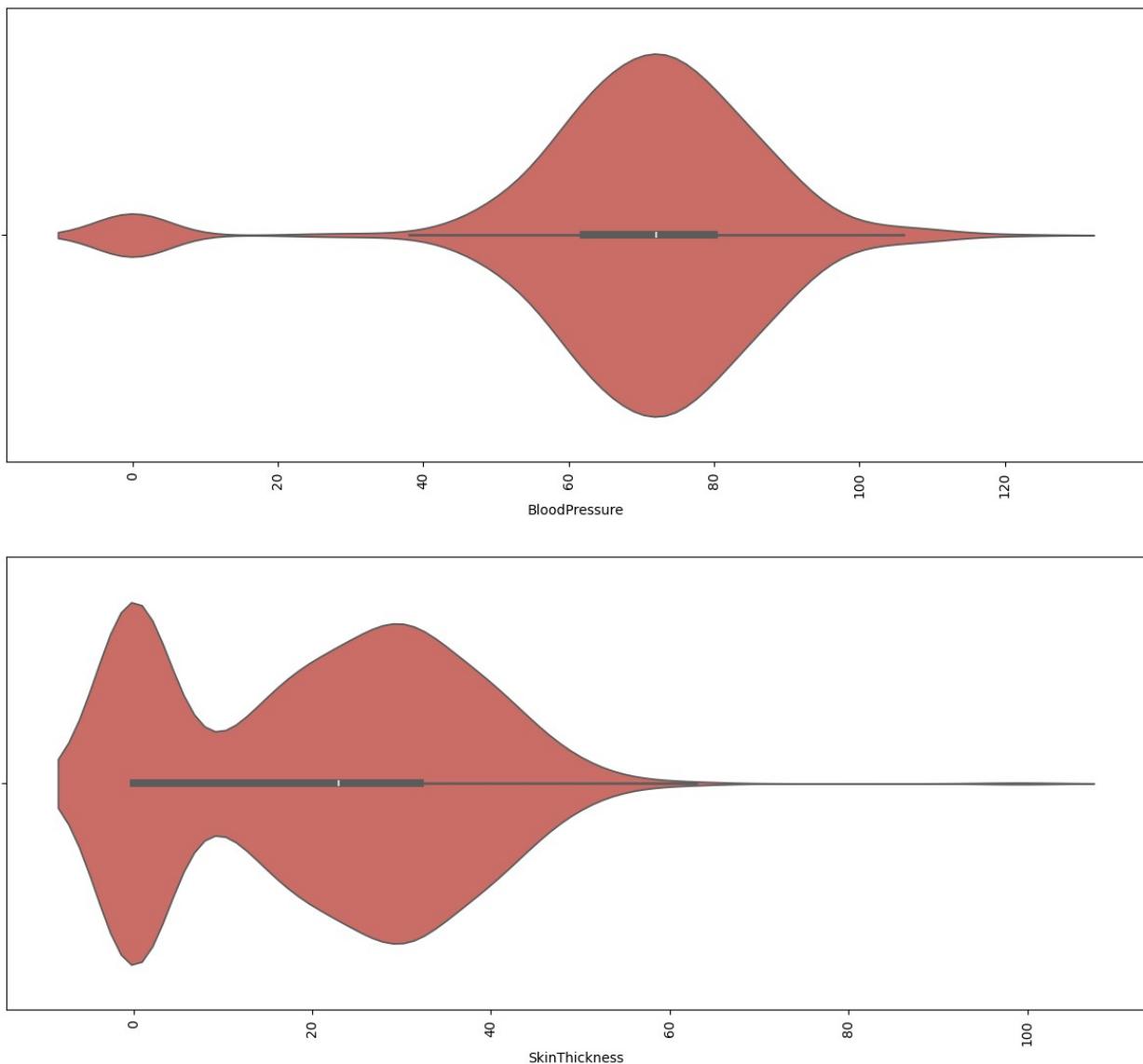


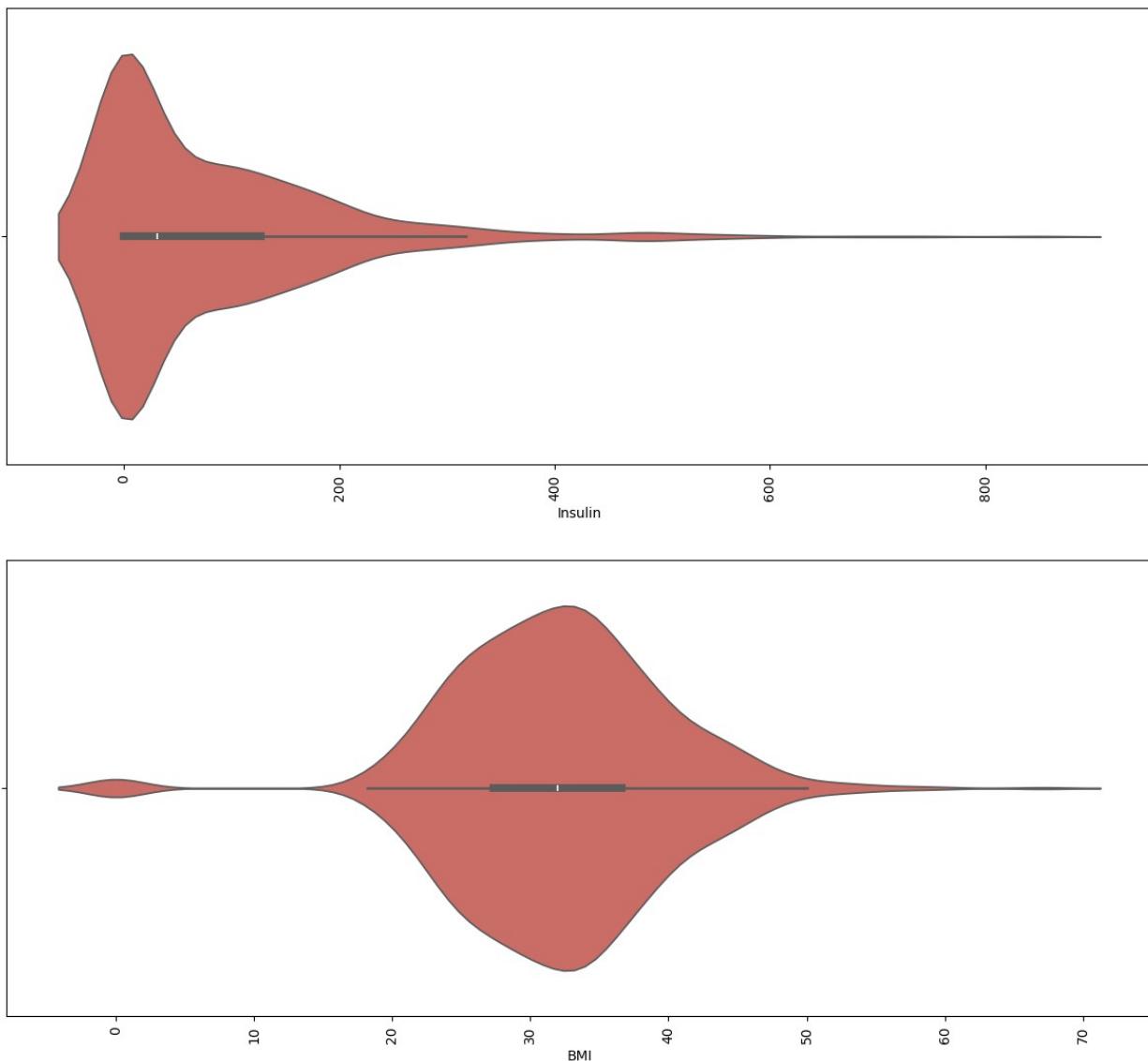


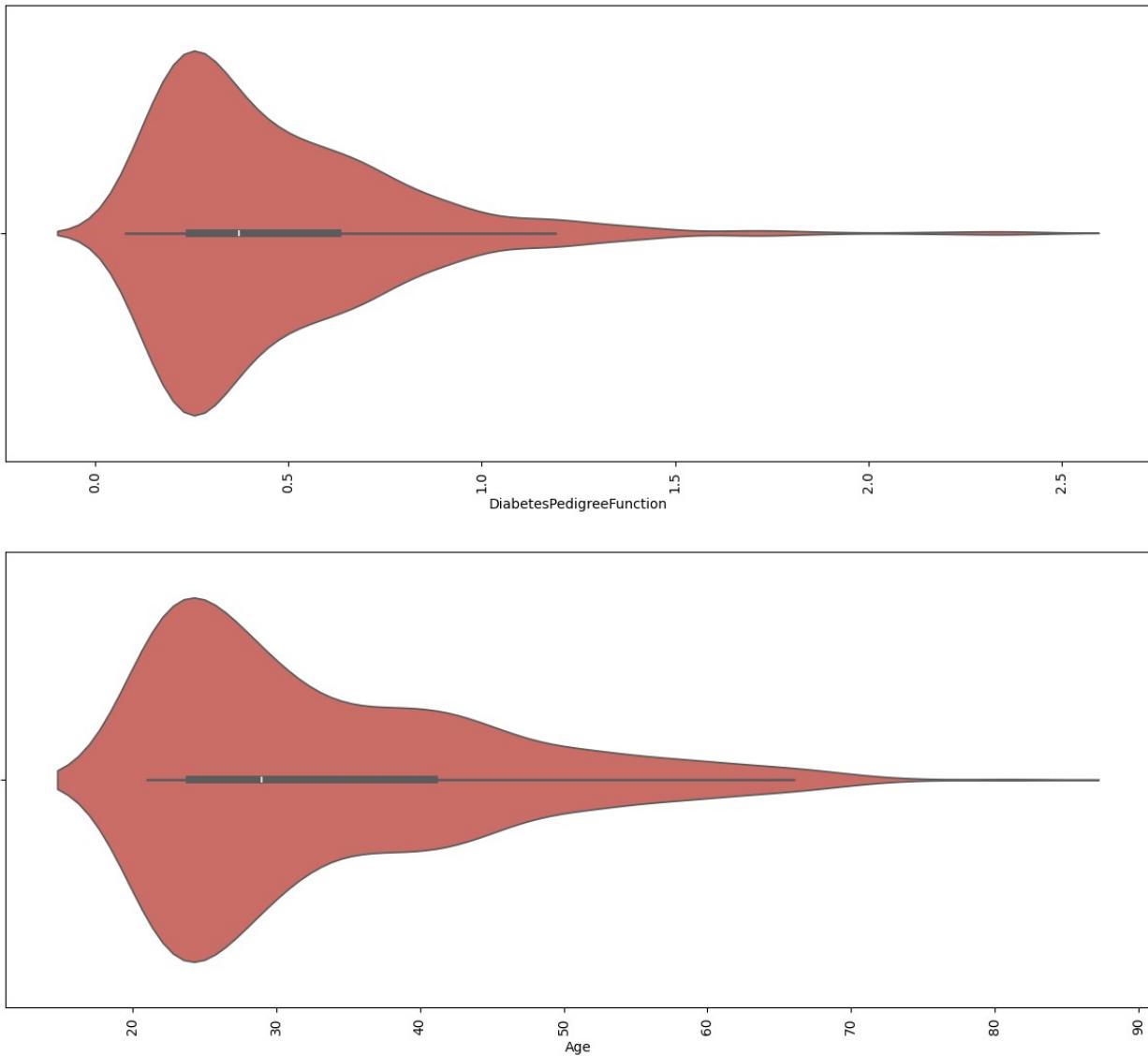


```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.violinplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```





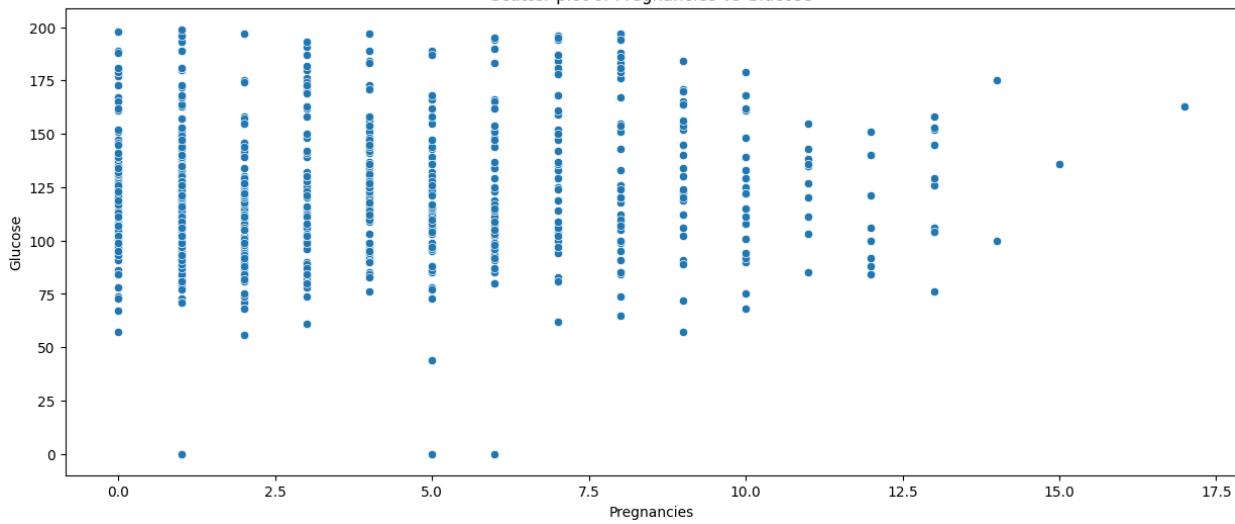




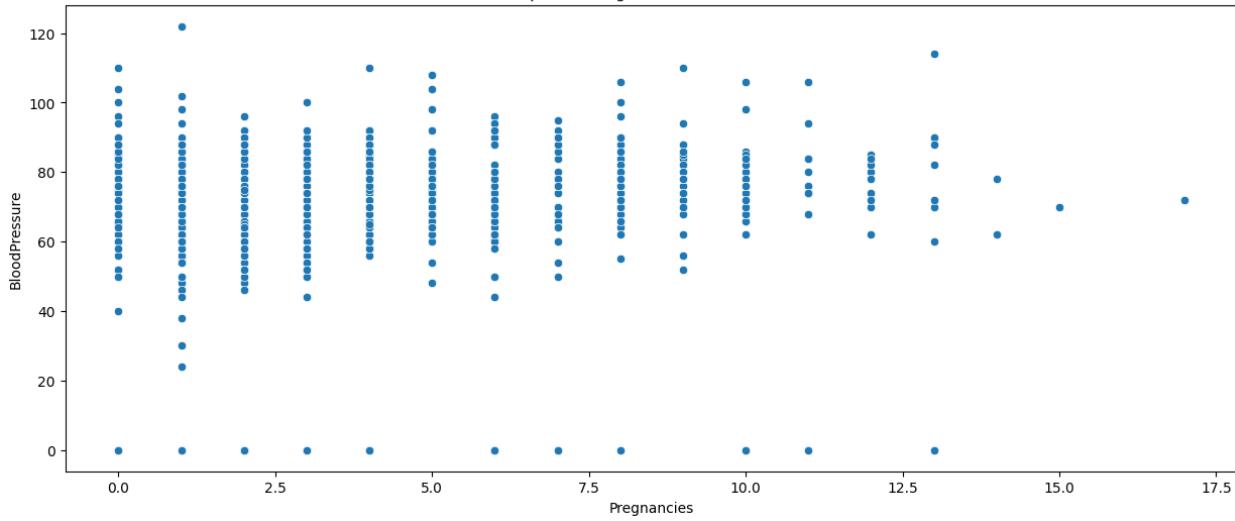
```

for i in range(len(continuous)):
    for j in range(i + 1, len(continuous)):
        plt.figure(figsize=(15, 6))
        sns.scatterplot(x=continuous[i], y=continuous[j], data=df,
palette='hls')
        plt.title(f'Scatter plot of {continuous[i]} vs
{continuous[j]}')
        plt.show()
    
```

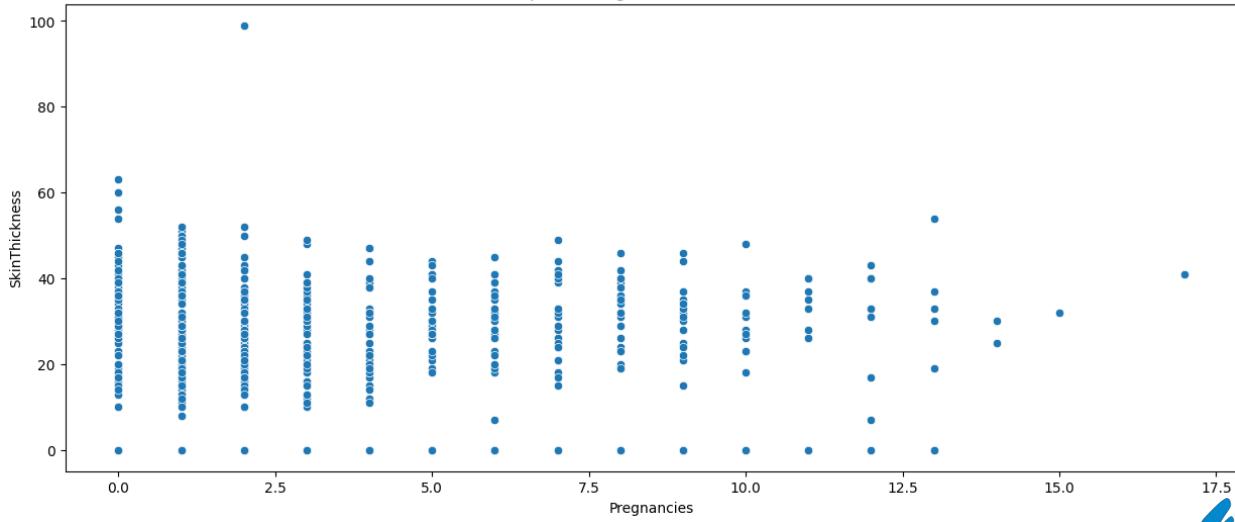
Scatter plot of Pregnancies vs Glucose



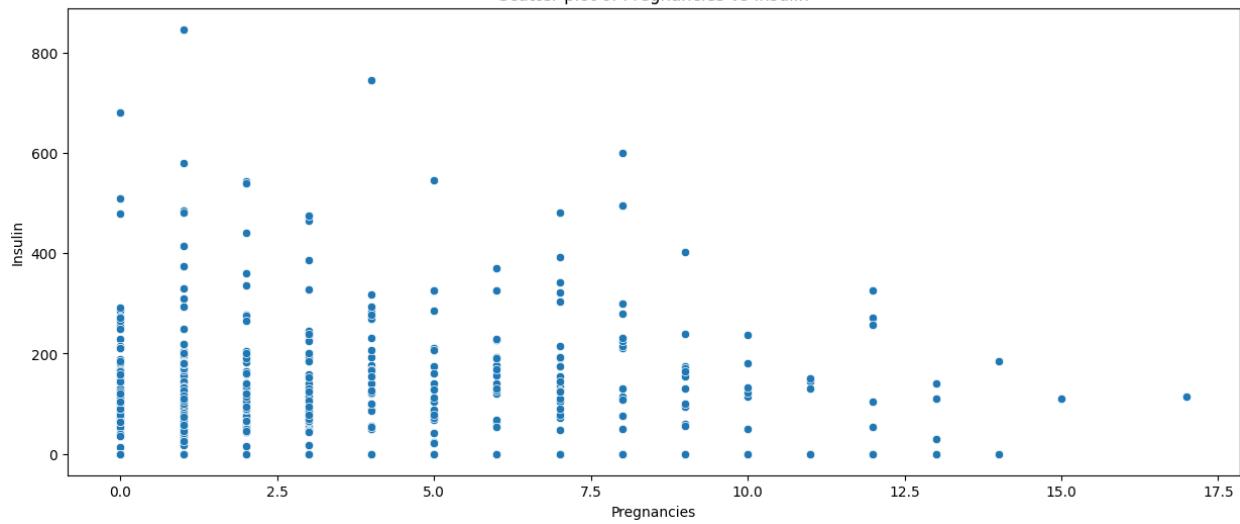
Scatter plot of Pregnancies vs BloodPressure



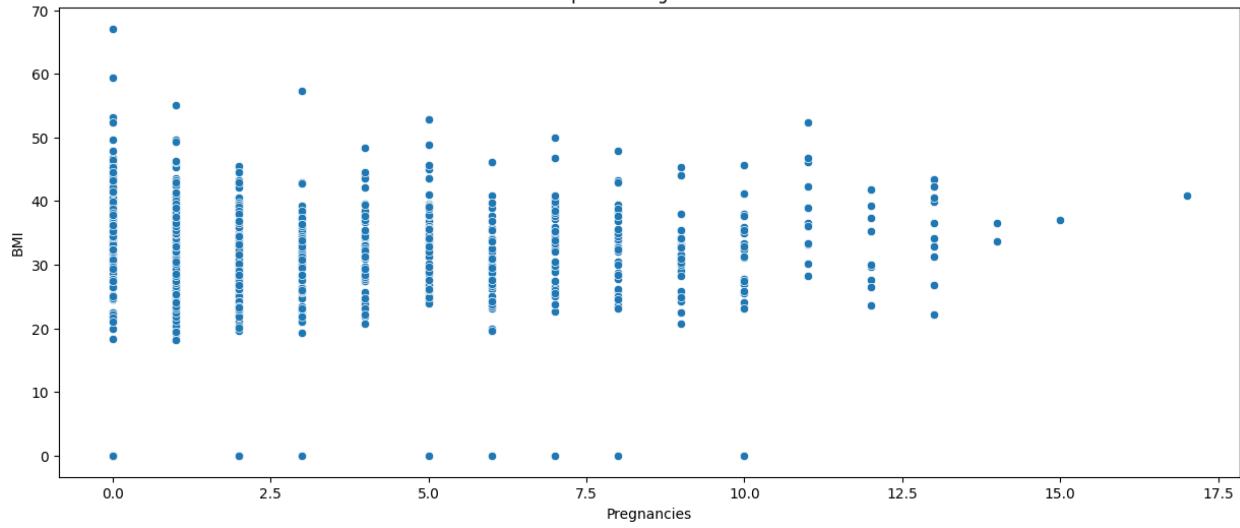
Scatter plot of Pregnancies vs SkinThickness



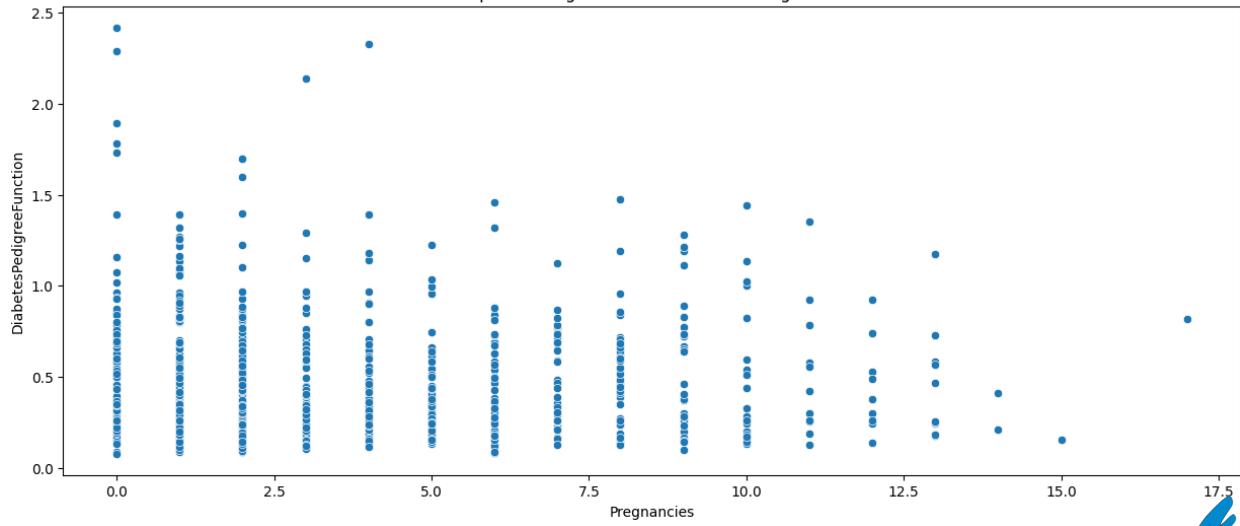
Scatter plot of Pregnancies vs Insulin



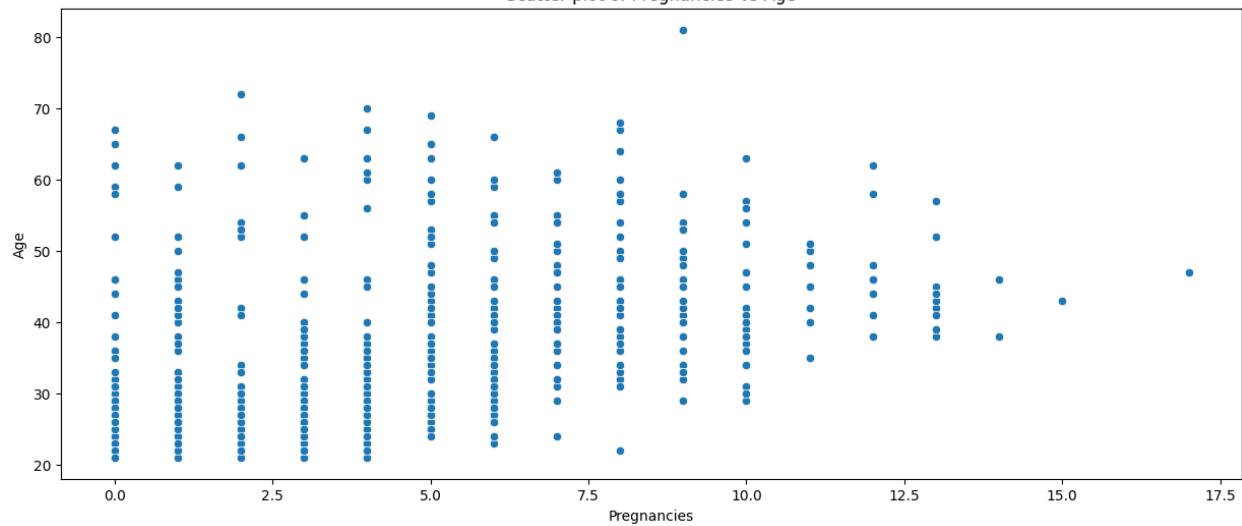
Scatter plot of Pregnancies vs BMI



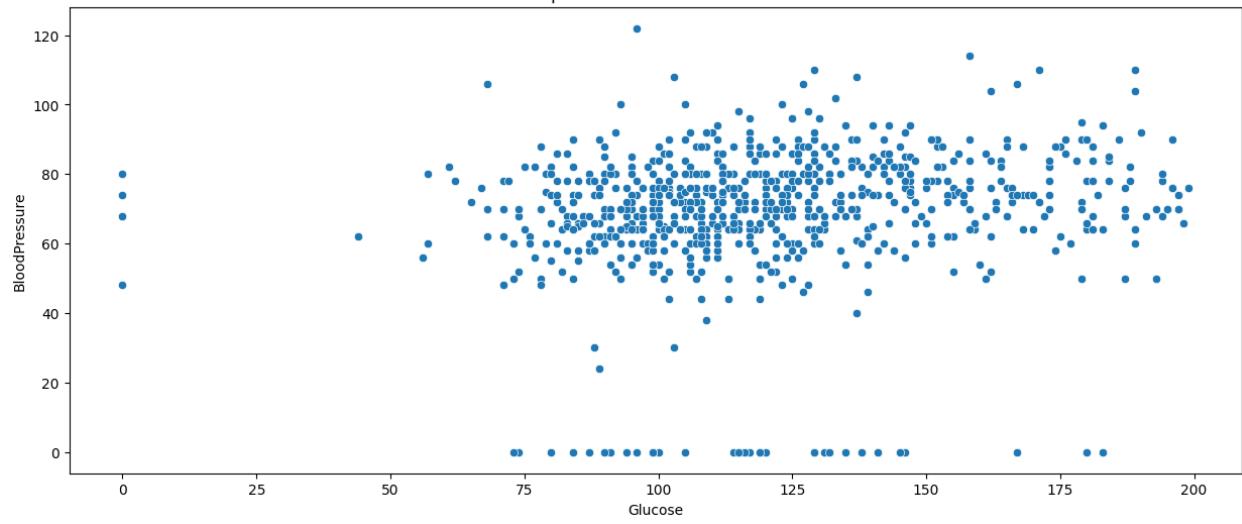
Scatter plot of Pregnancies vs DiabetesPedigreeFunction



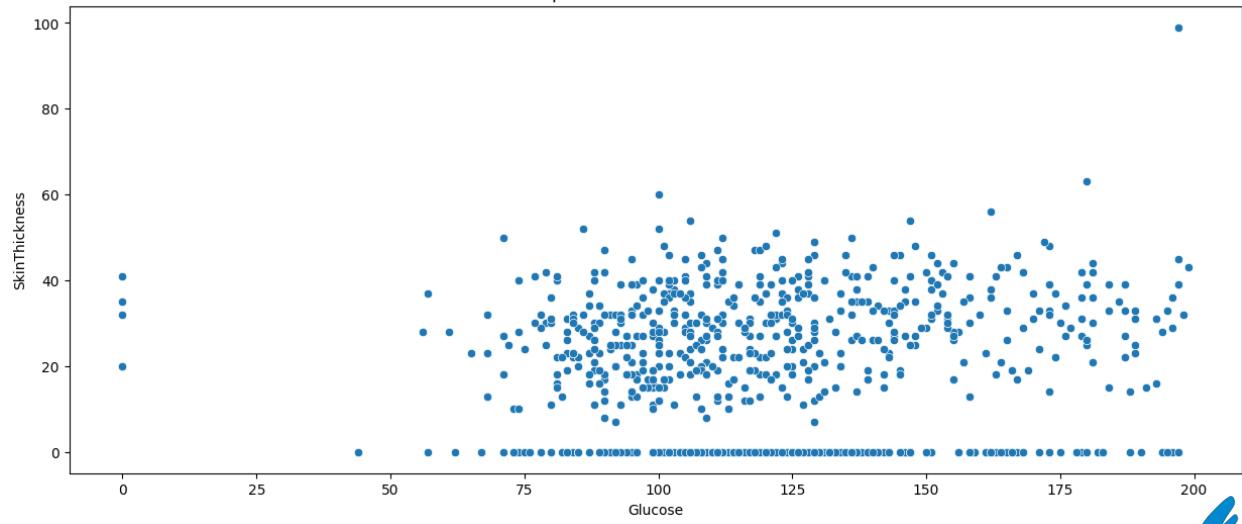
Scatter plot of Pregnancies vs Age

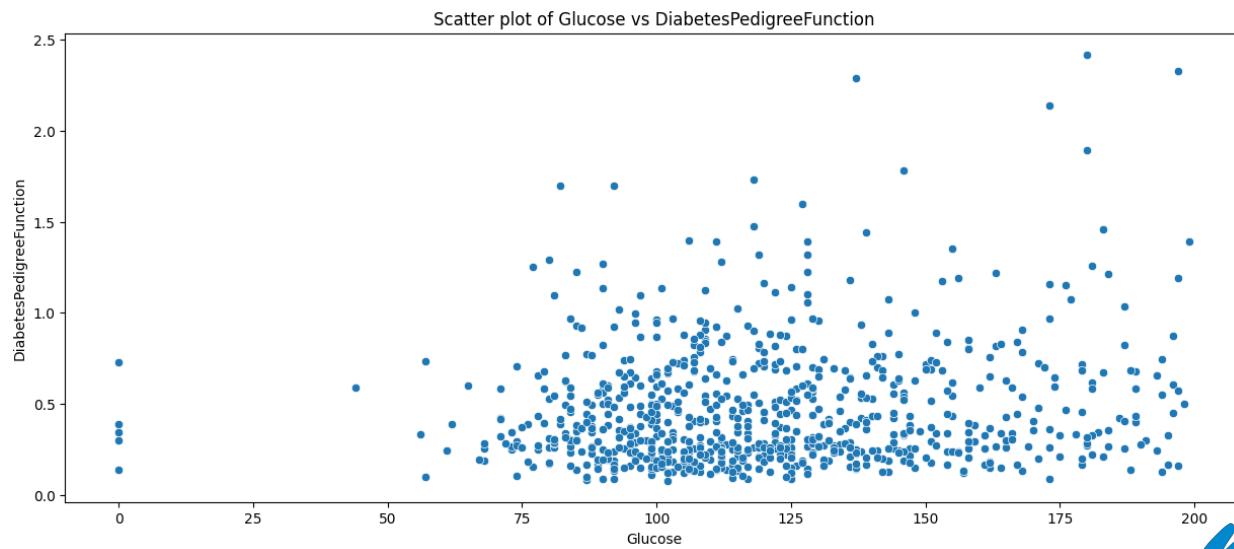
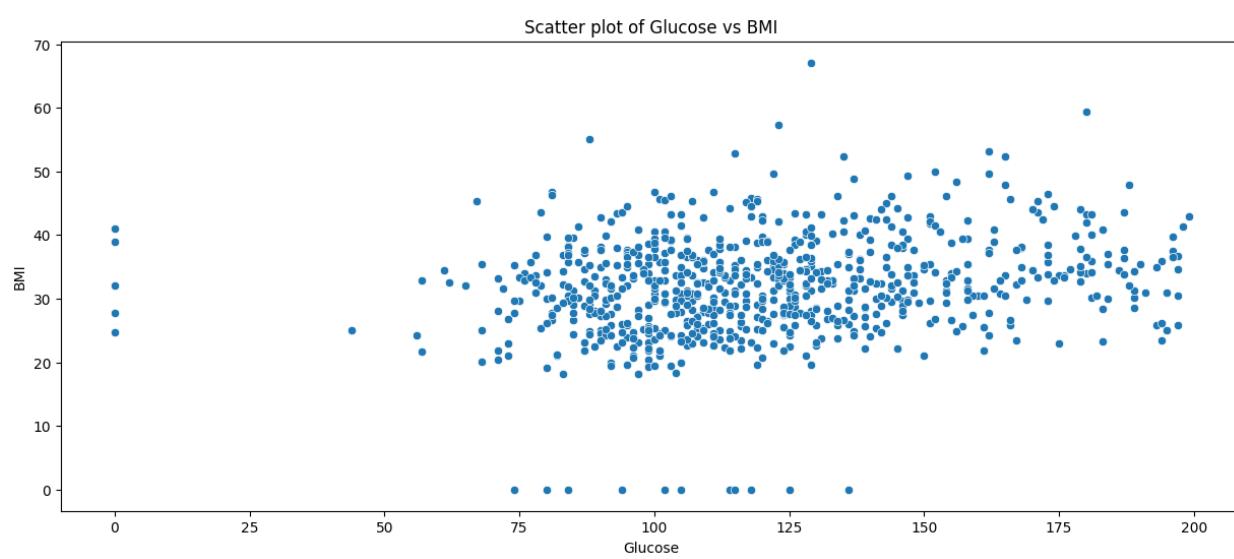
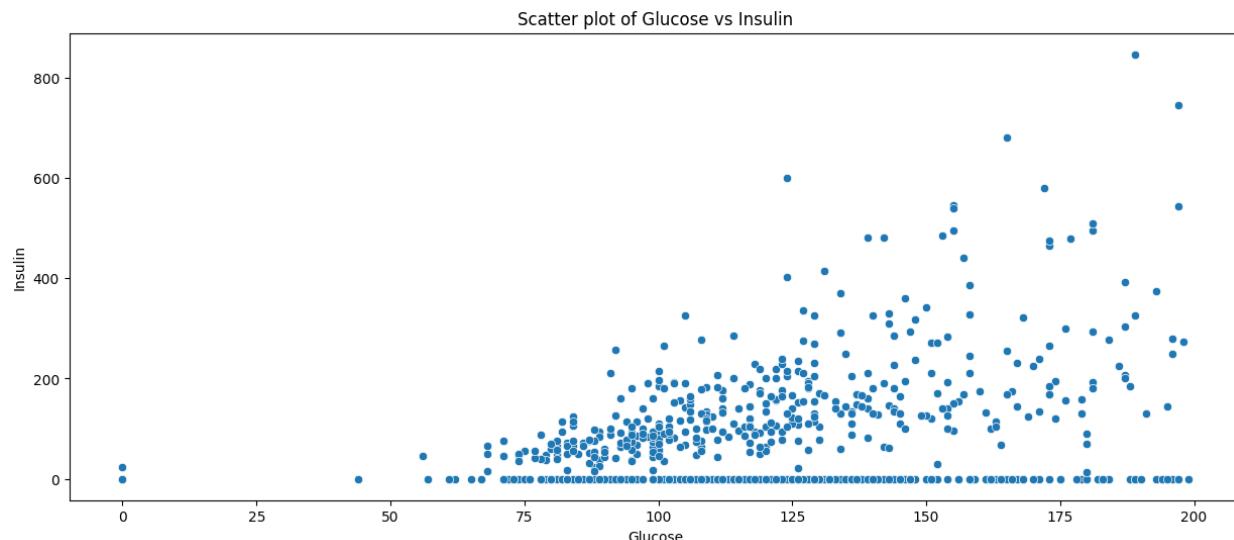


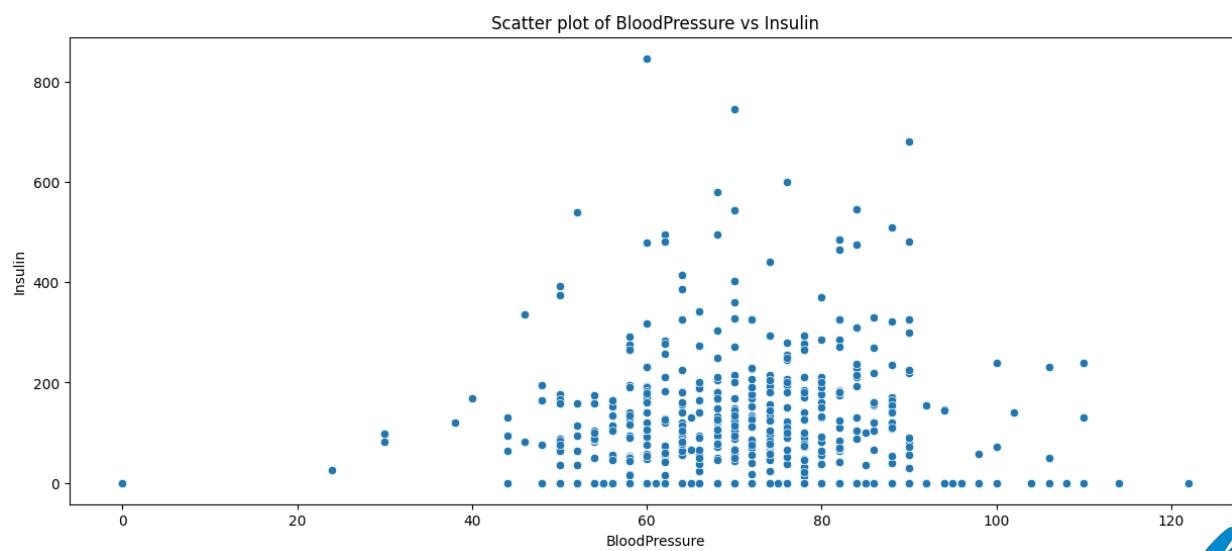
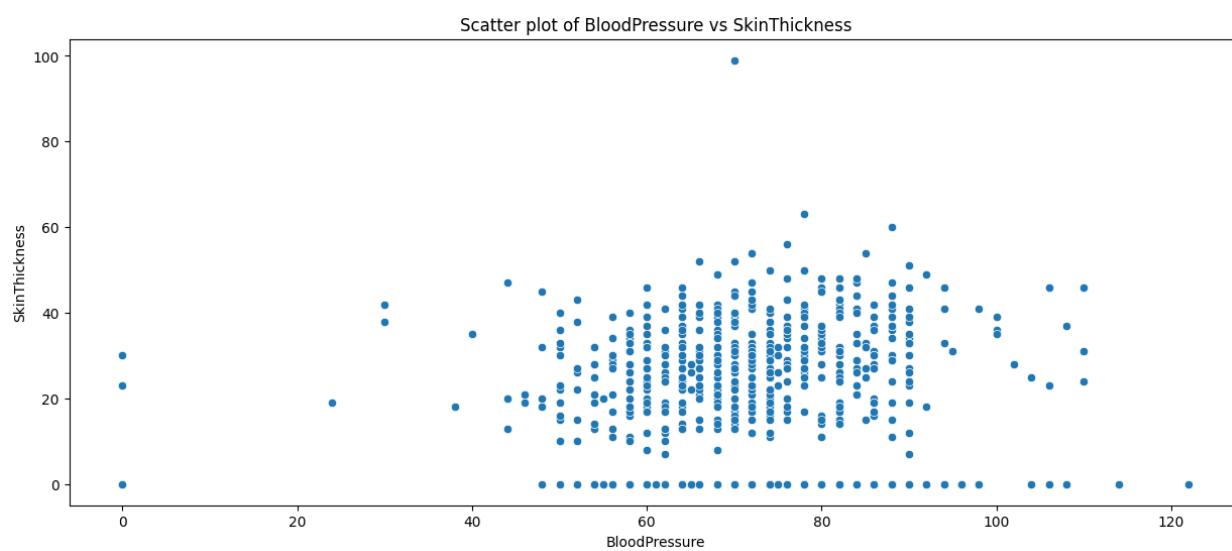
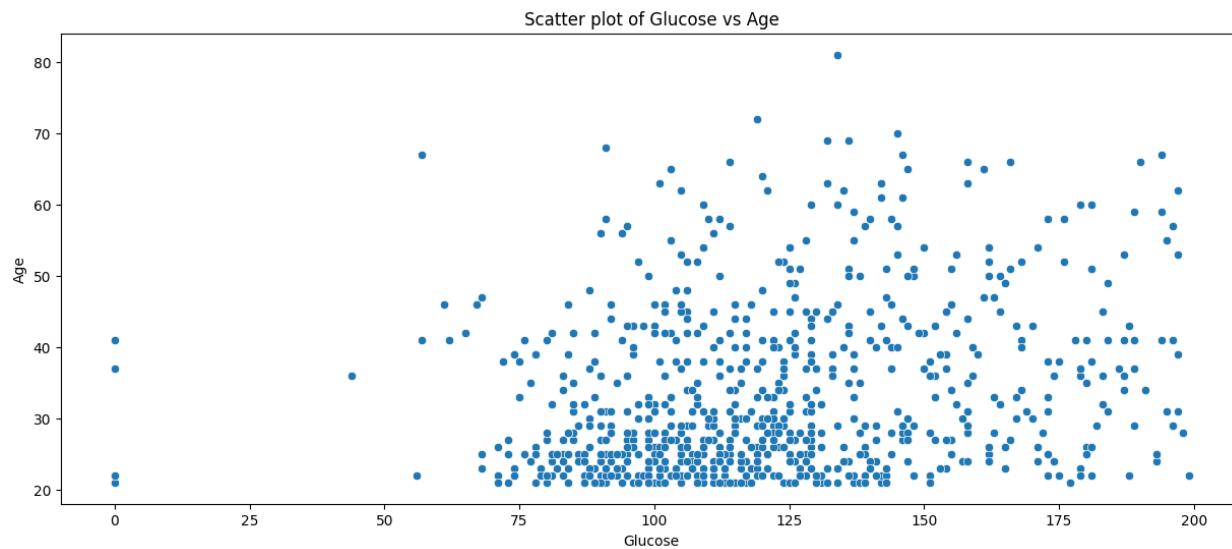
Scatter plot of Glucose vs BloodPressure

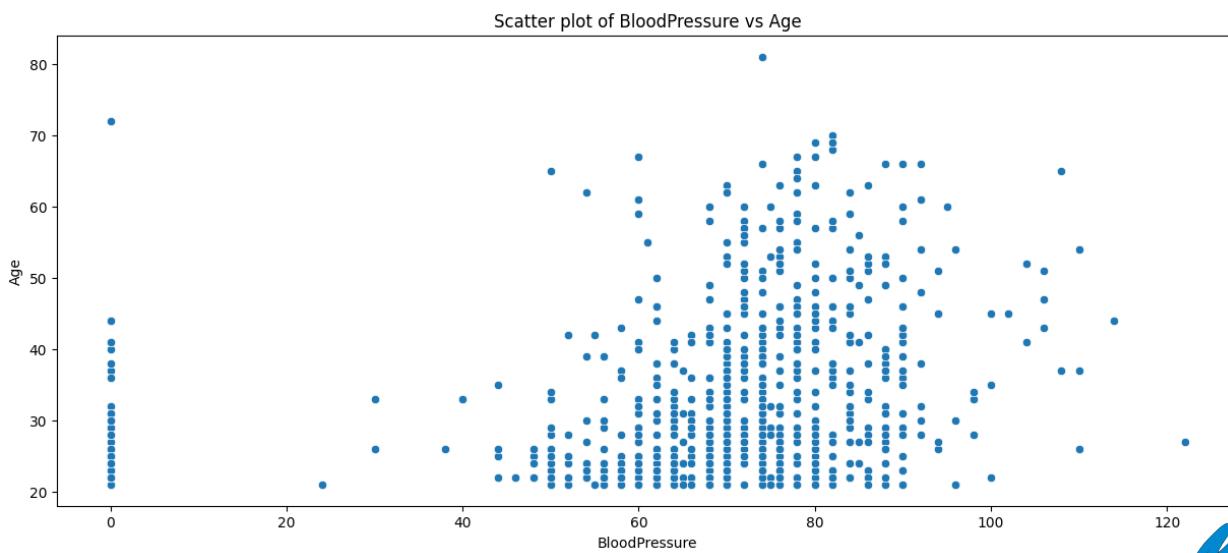
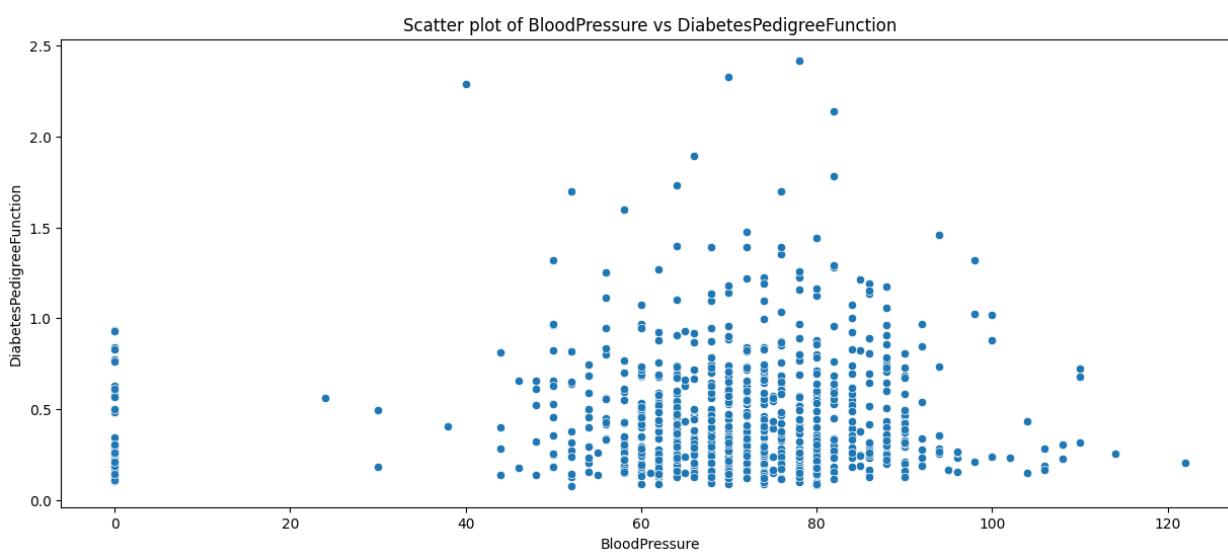
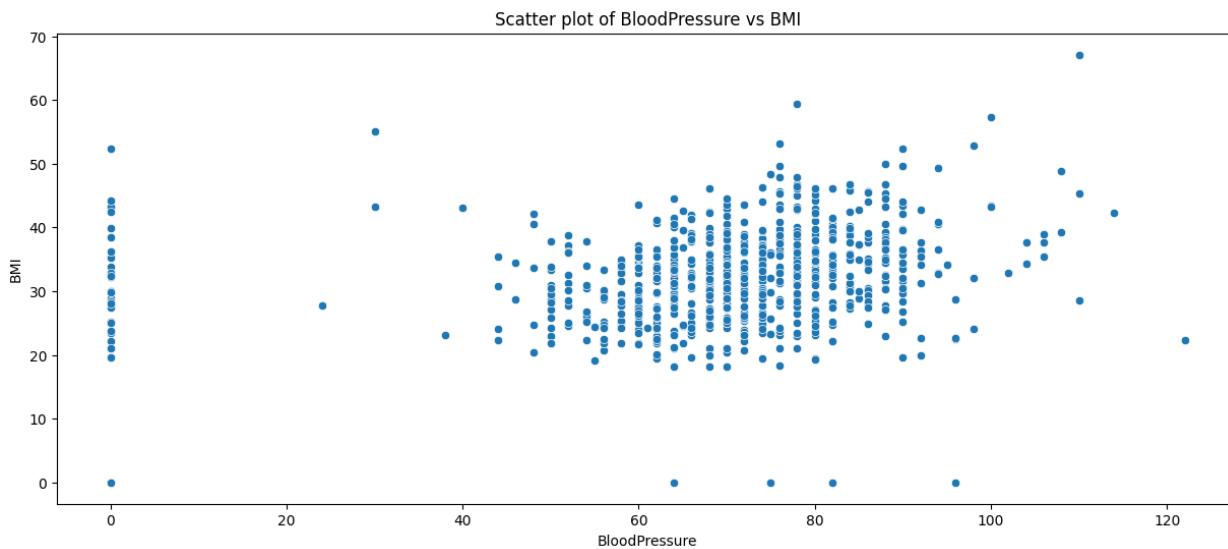


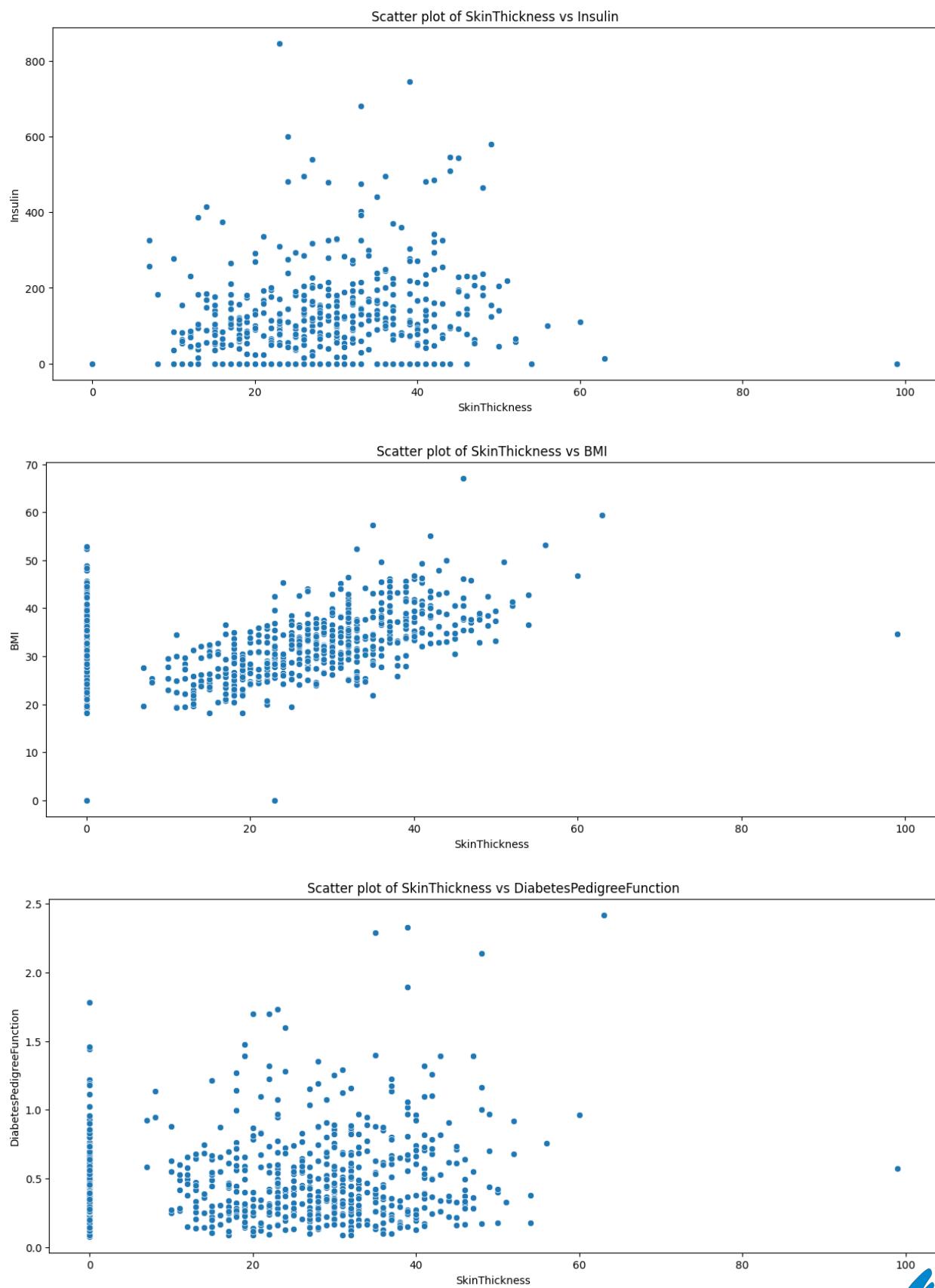
Scatter plot of Glucose vs SkinThickness

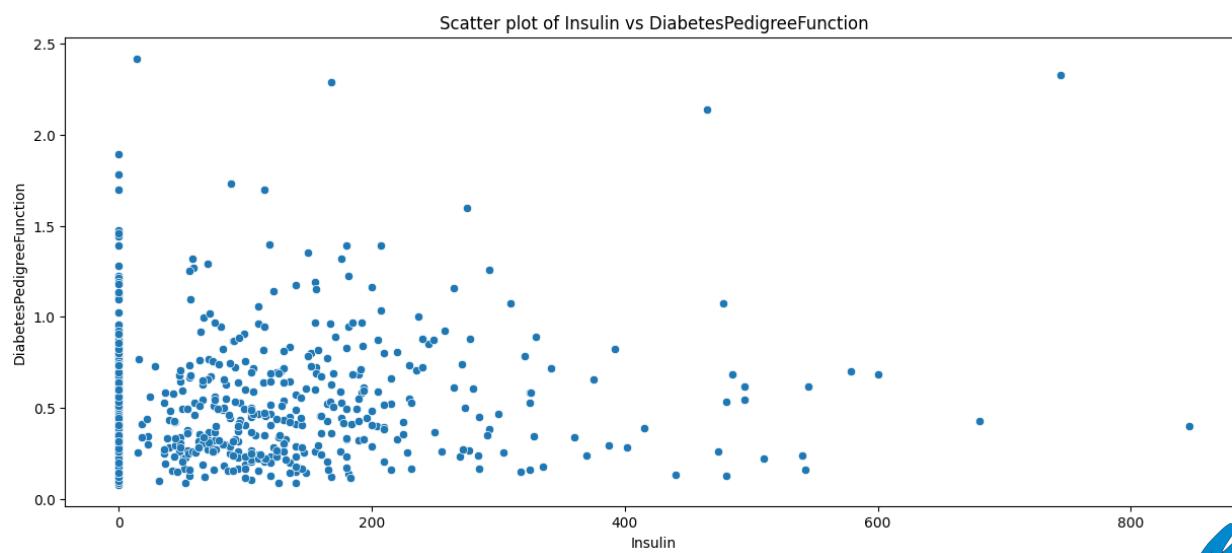
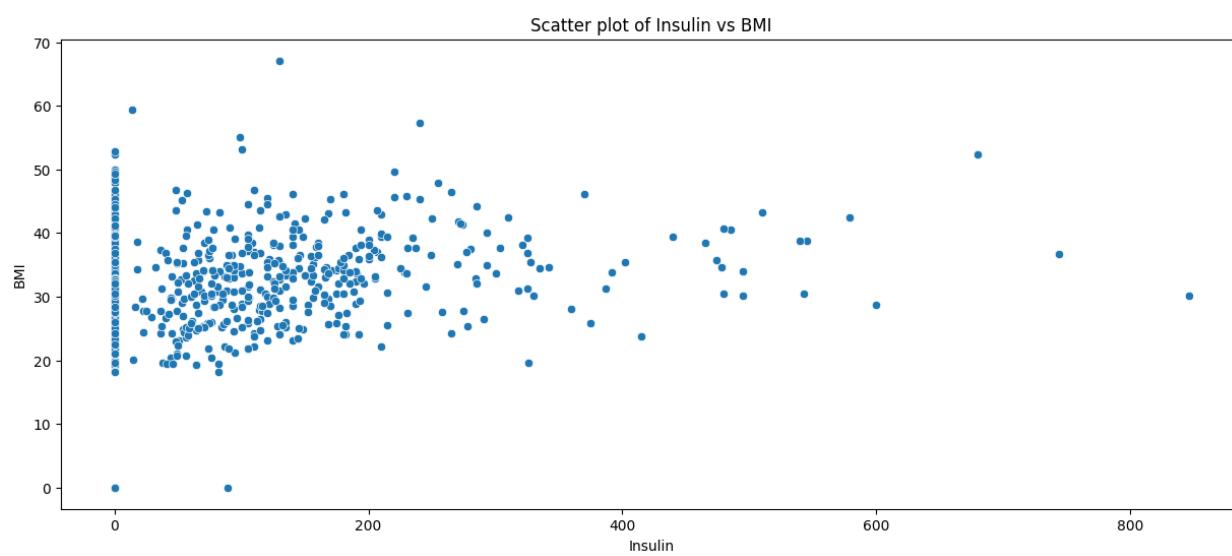
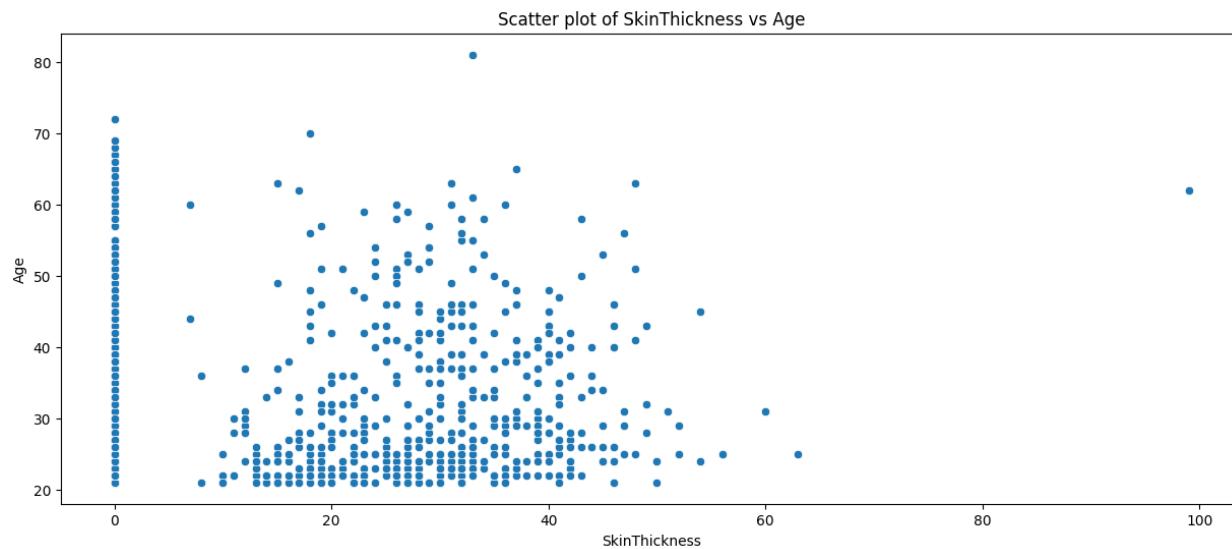




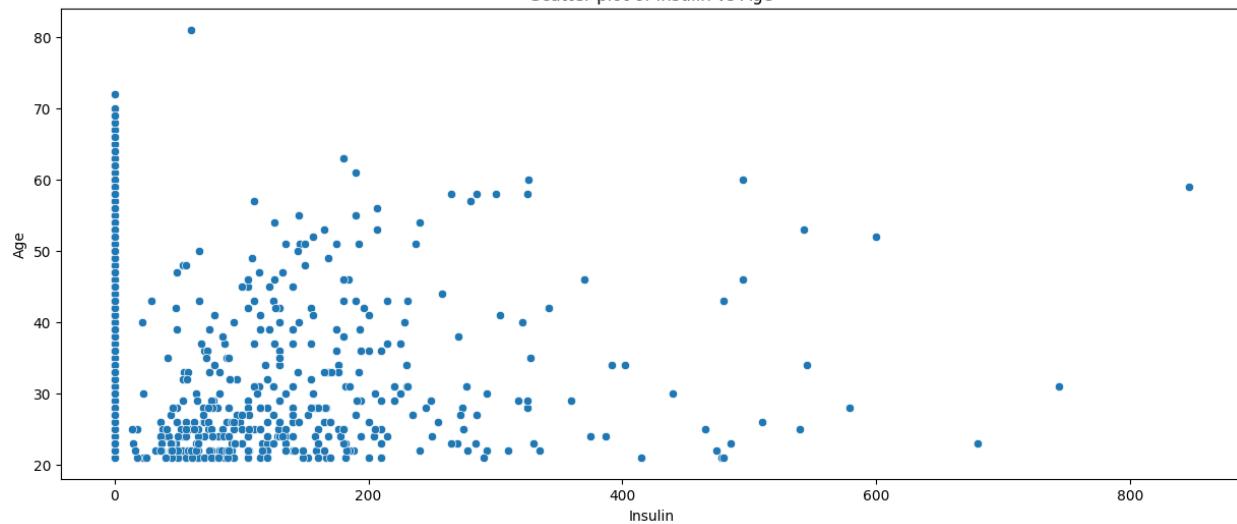




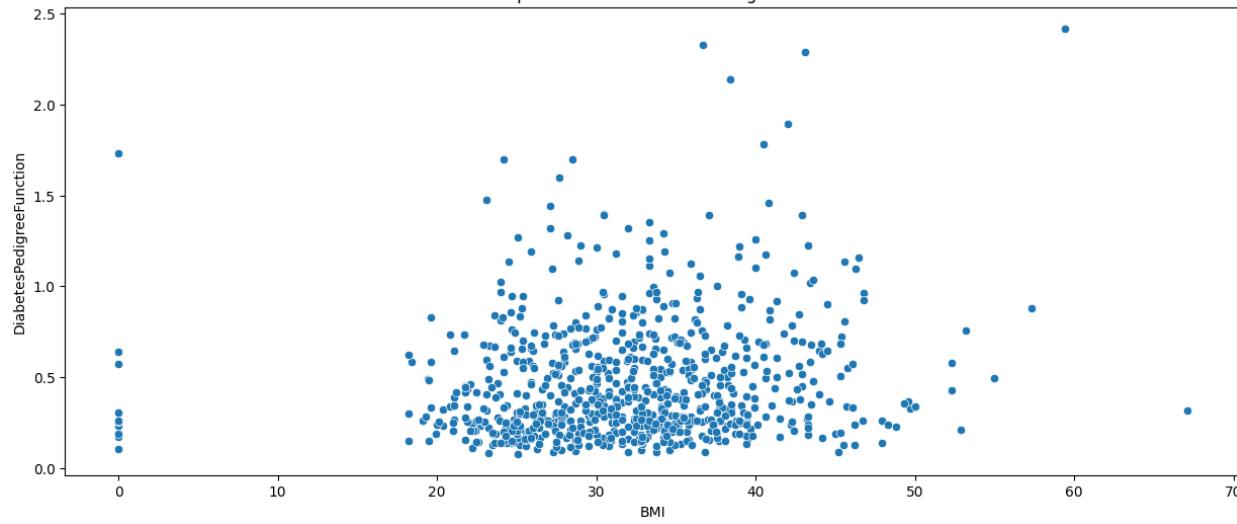




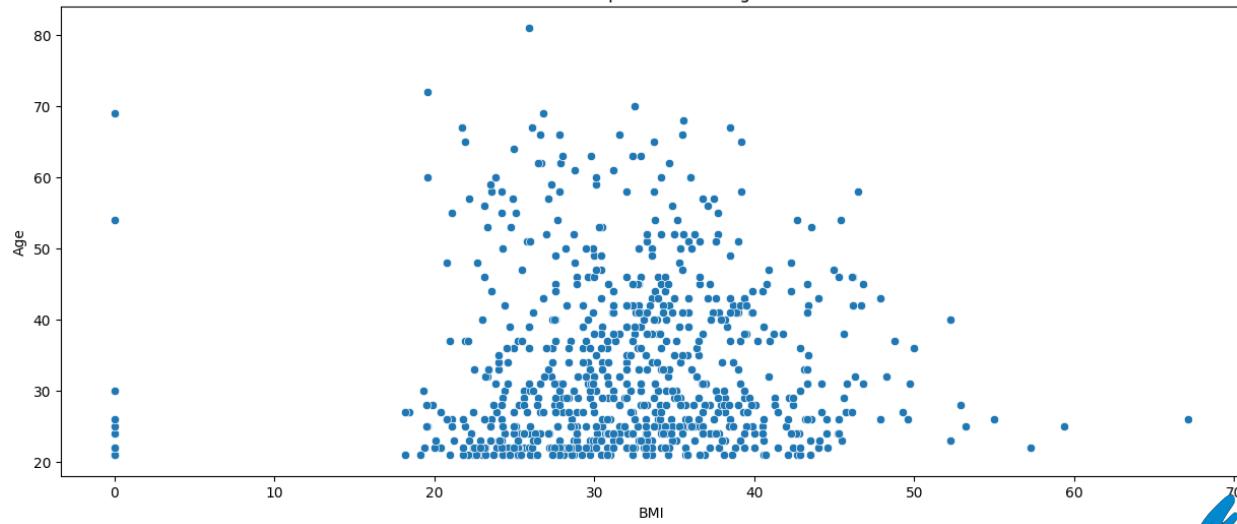
Scatter plot of Insulin vs Age

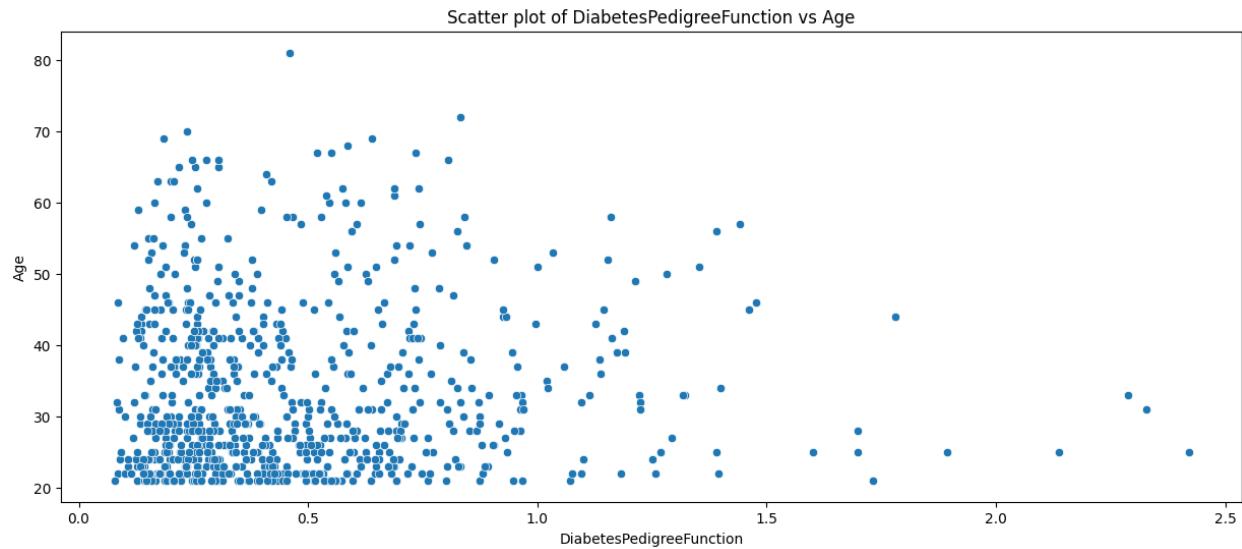


Scatter plot of BMI vs DiabetesPedigreeFunction



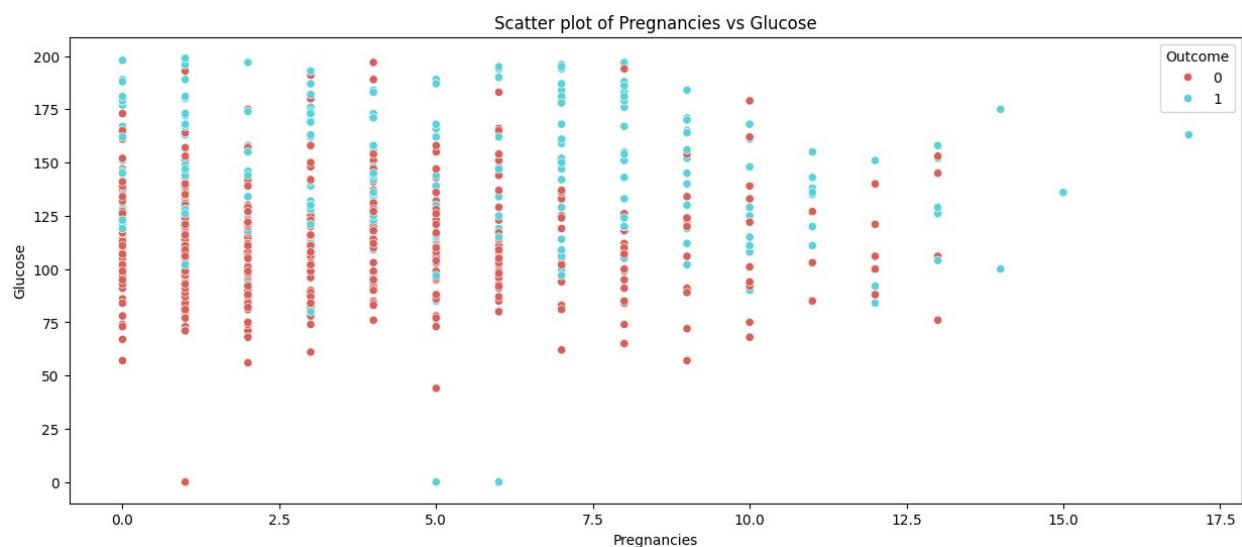
Scatter plot of BMI vs Age



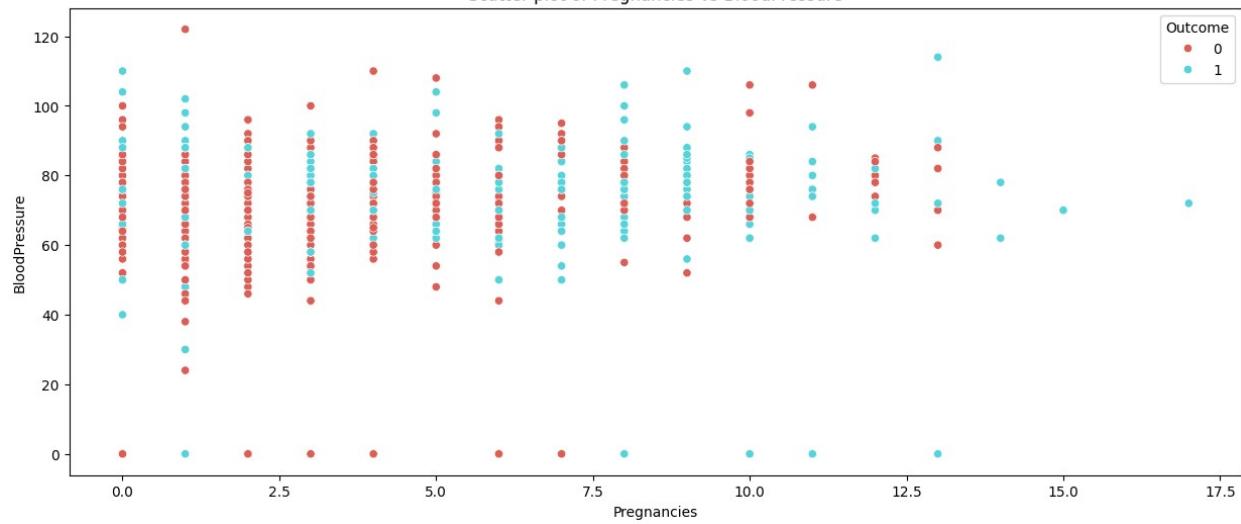


```

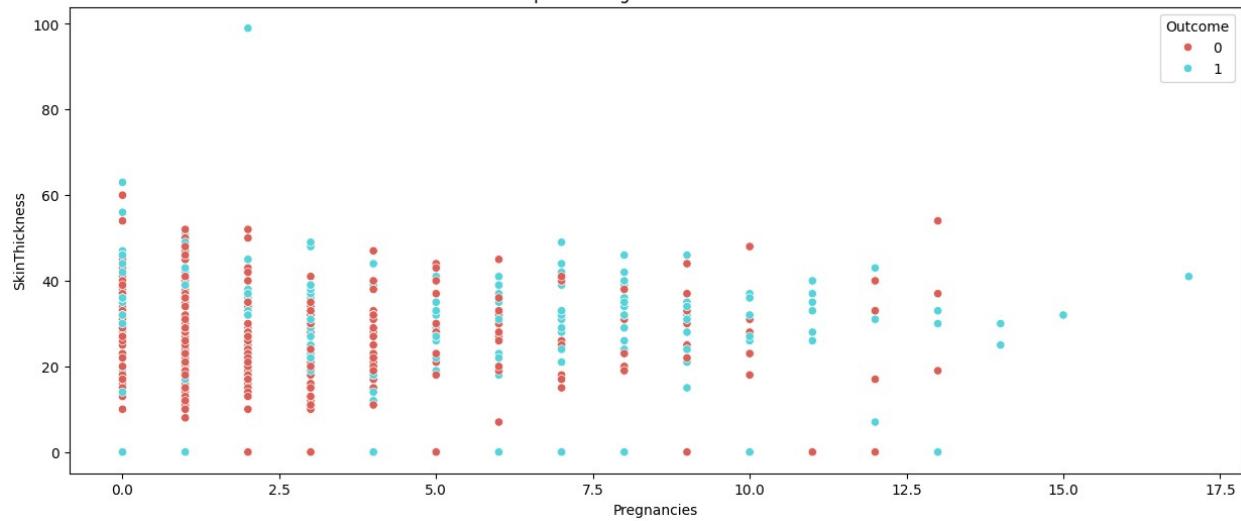
for i in range(len(continuous)):
    for j in range(i + 1, len(continuous)):
        plt.figure(figsize=(15, 6))
        sns.scatterplot(x=continuous[i], y=continuous[j], data=df, hue = 'Outcome', palette='hls')
        plt.title(f'Scatter plot of {continuous[i]} vs {continuous[j]}' )
        plt.show()
    
```



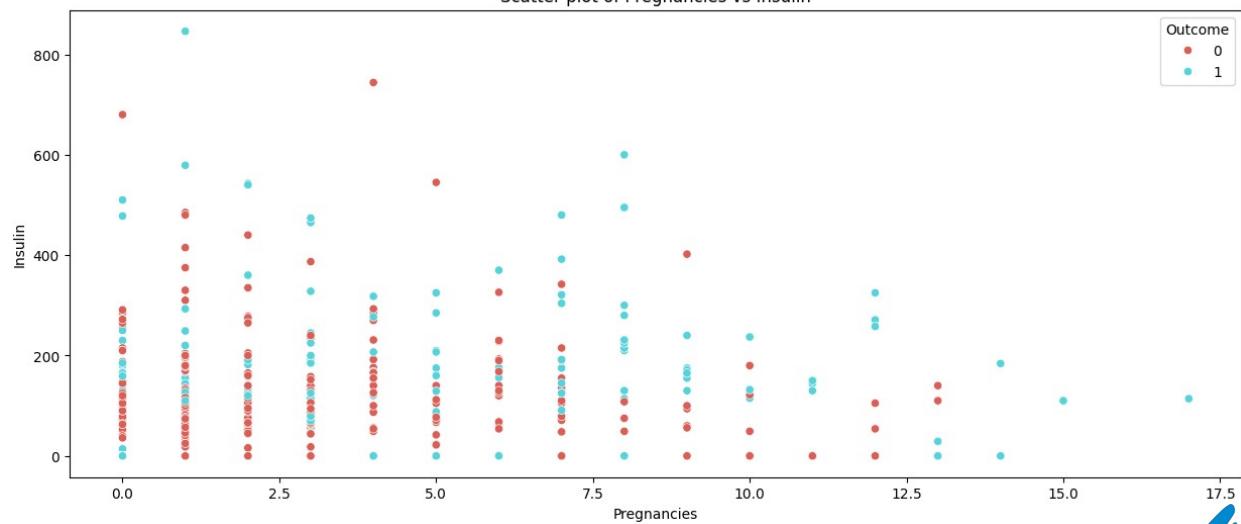
Scatter plot of Pregnancies vs BloodPressure

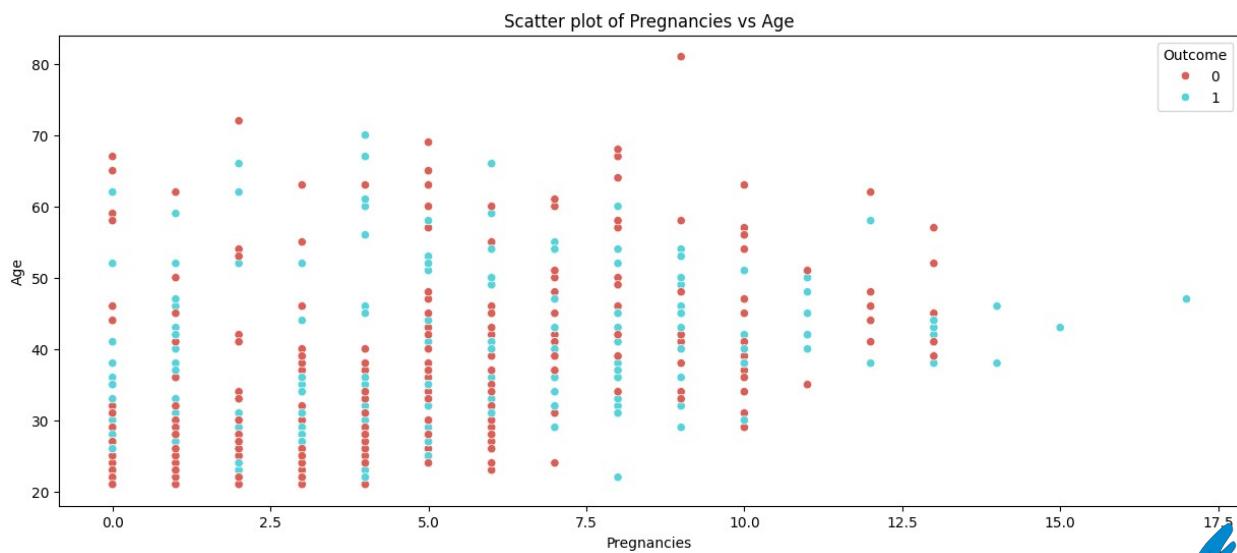
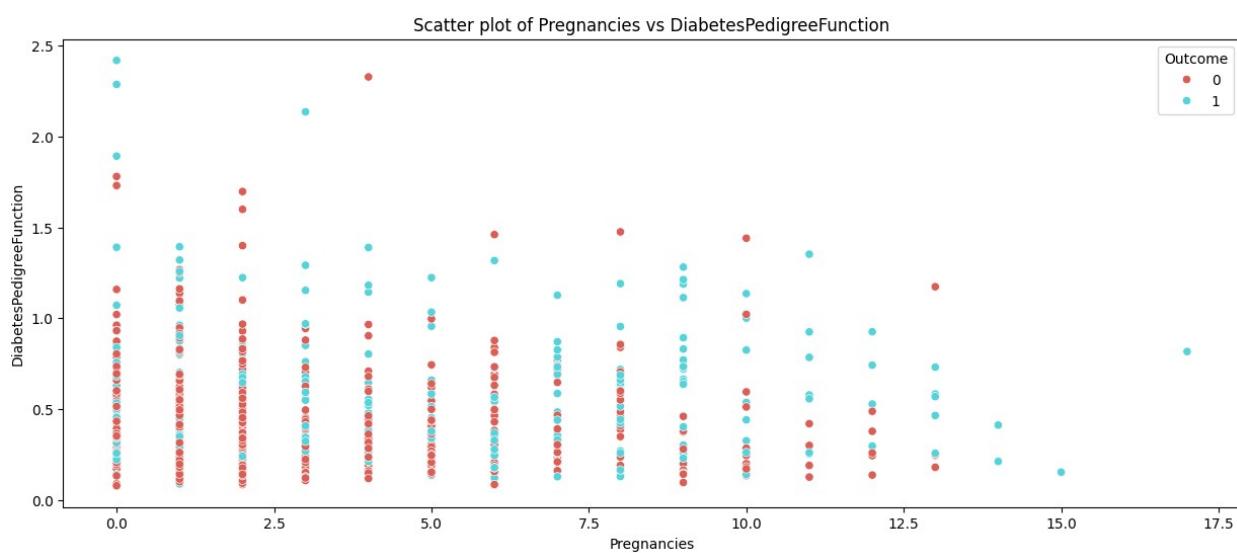
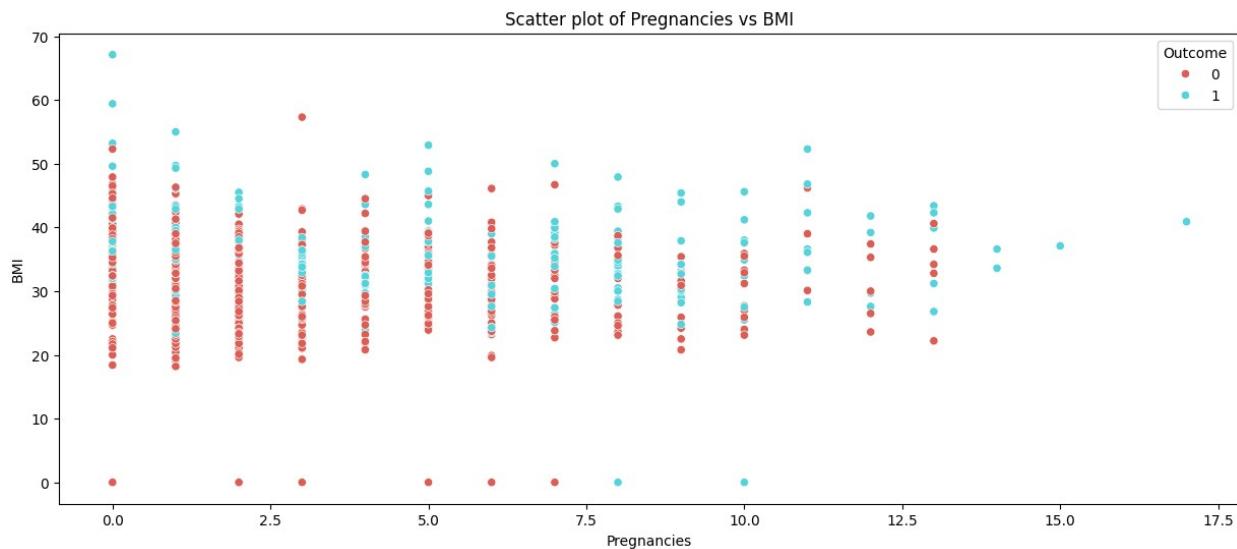


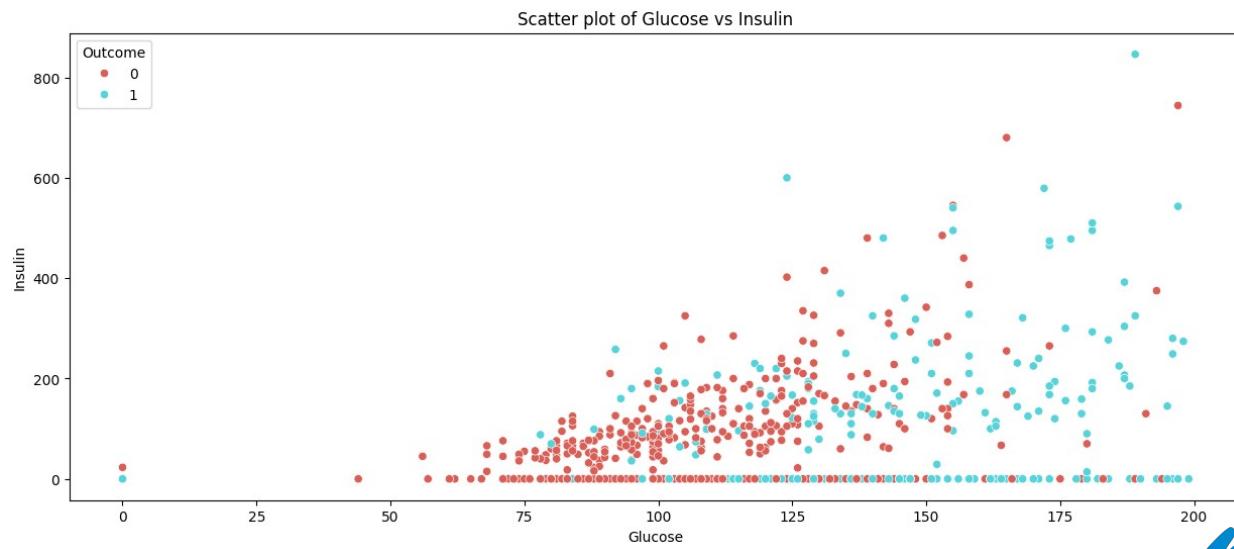
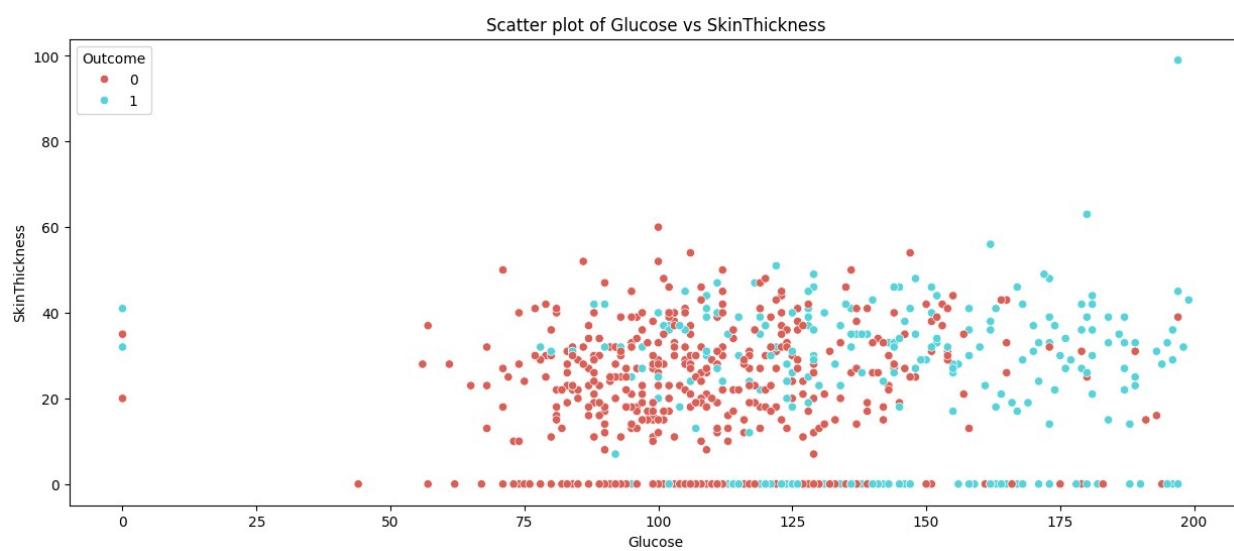
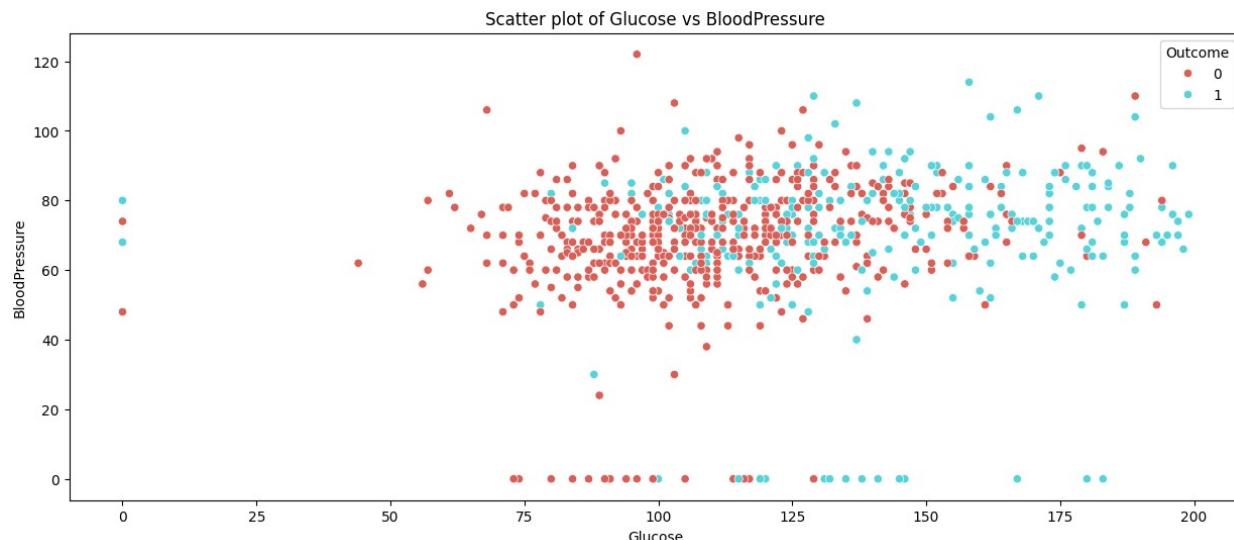
Scatter plot of Pregnancies vs SkinThickness

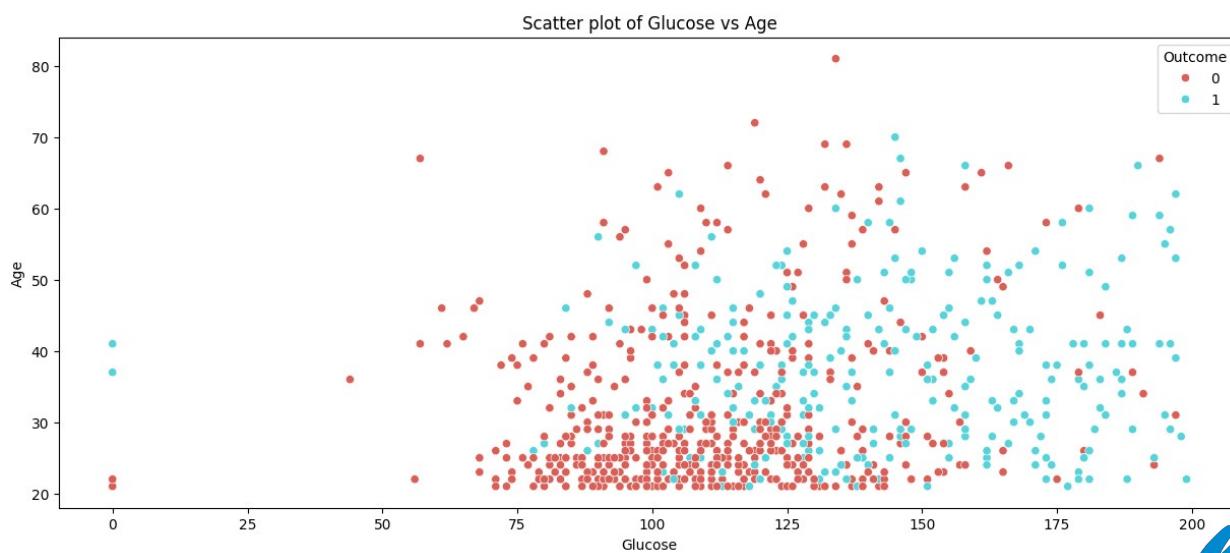
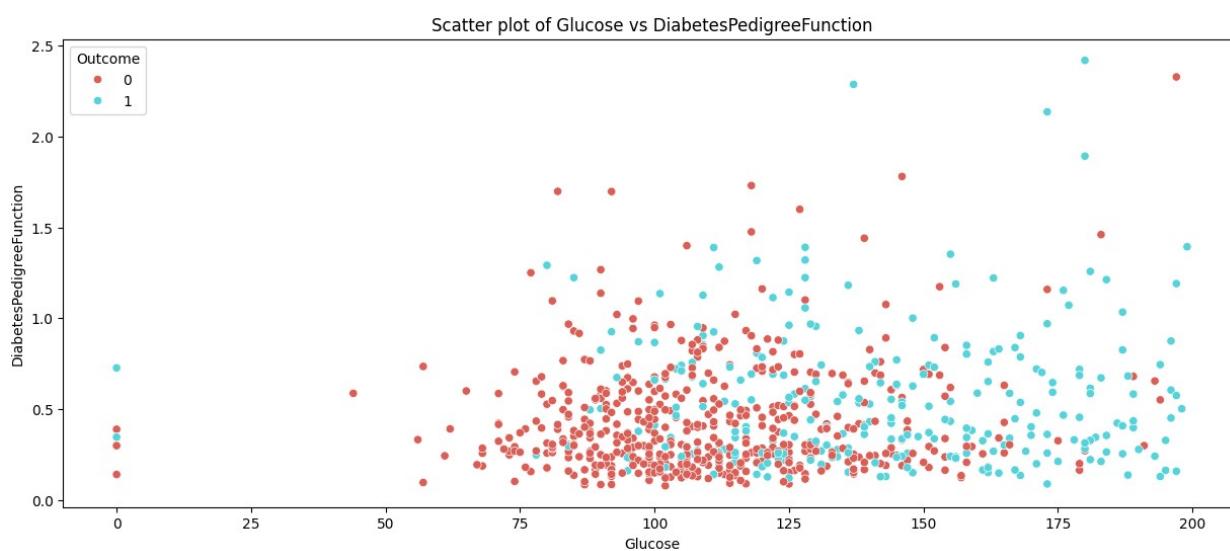
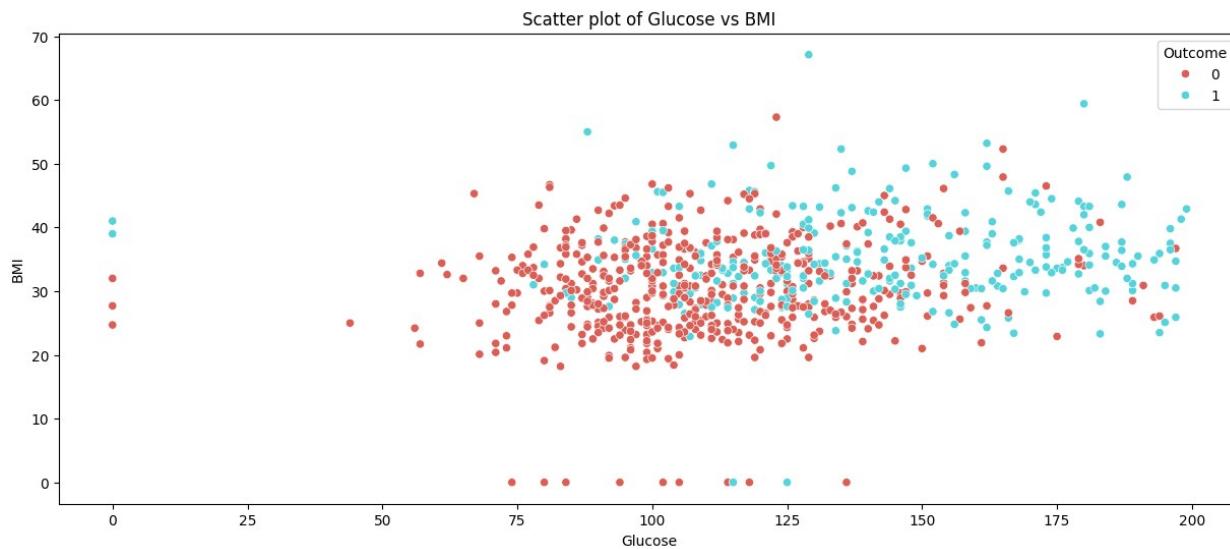


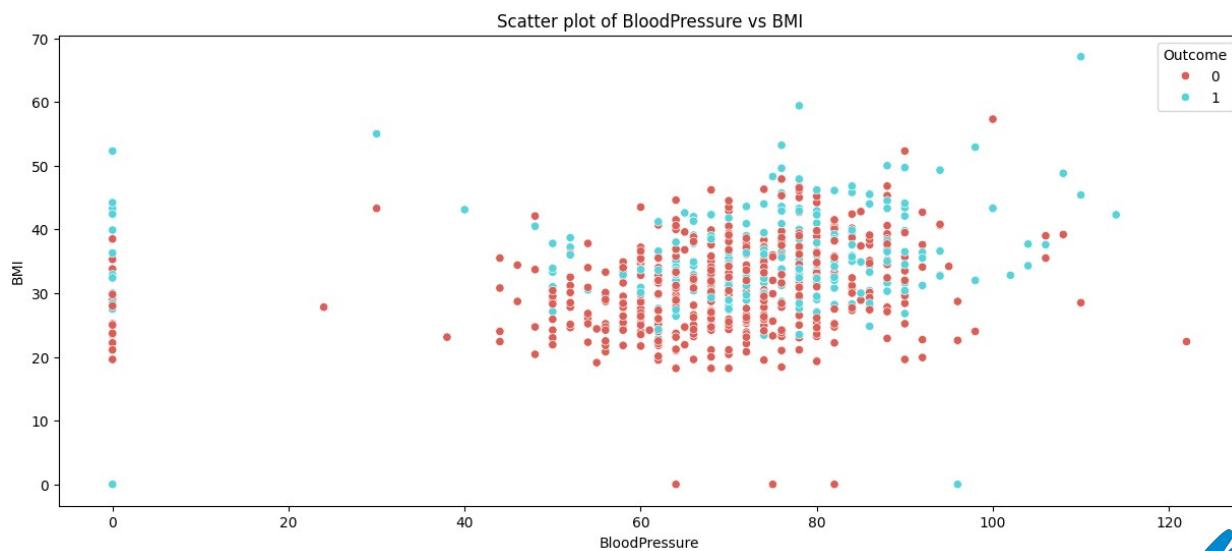
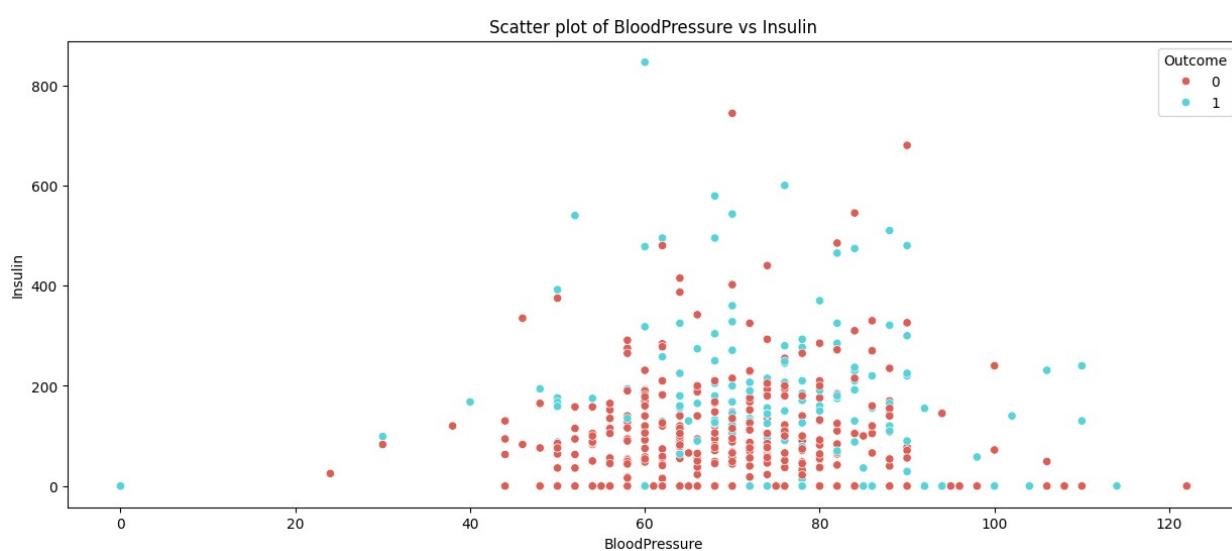
Scatter plot of Pregnancies vs Insulin

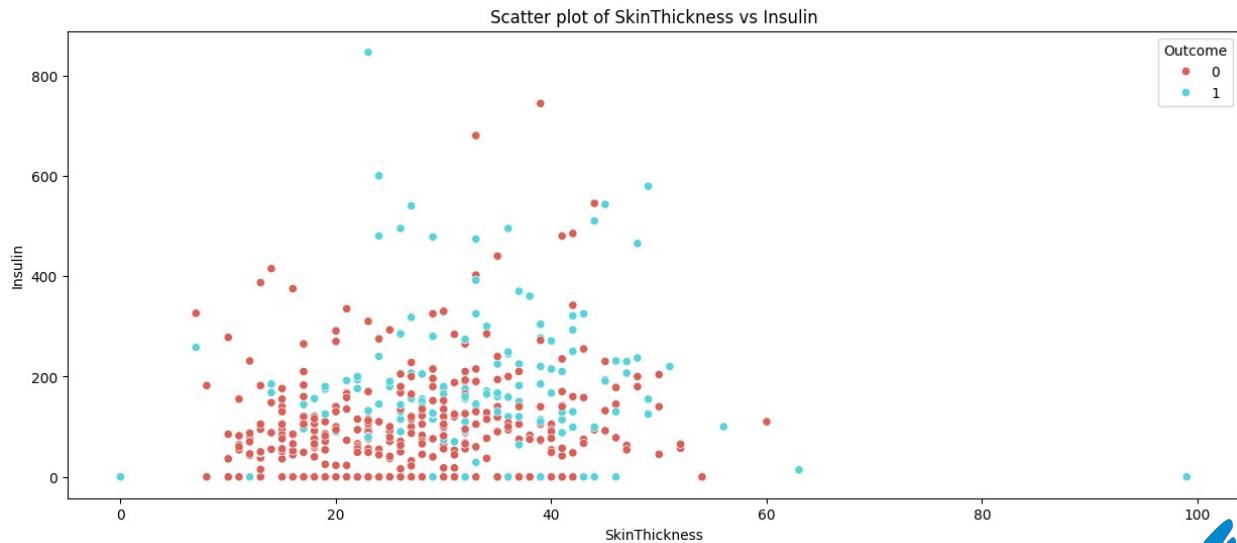
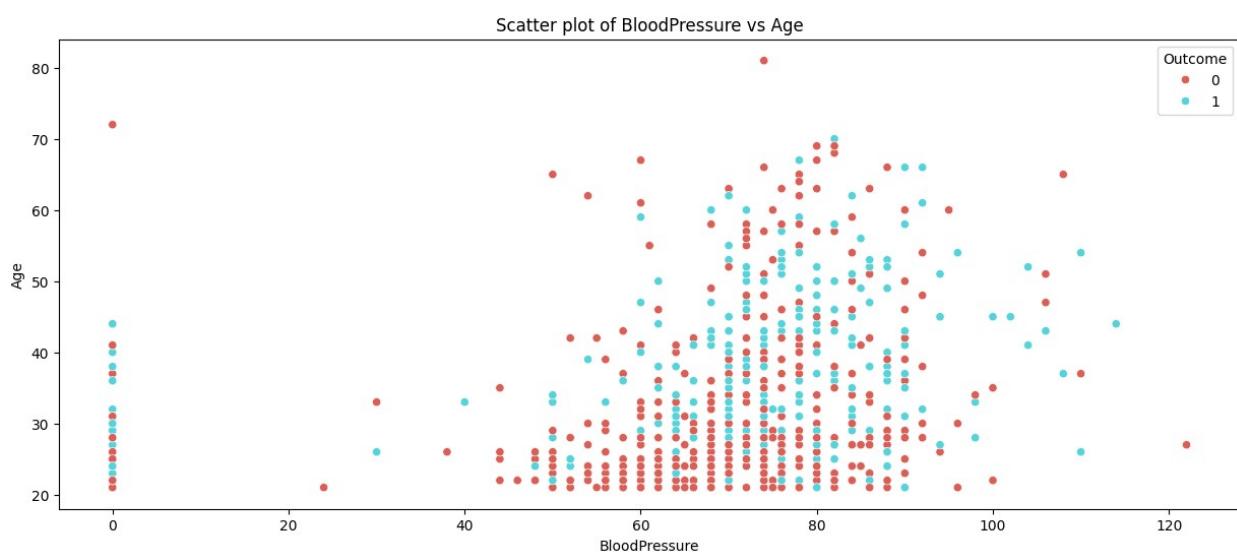
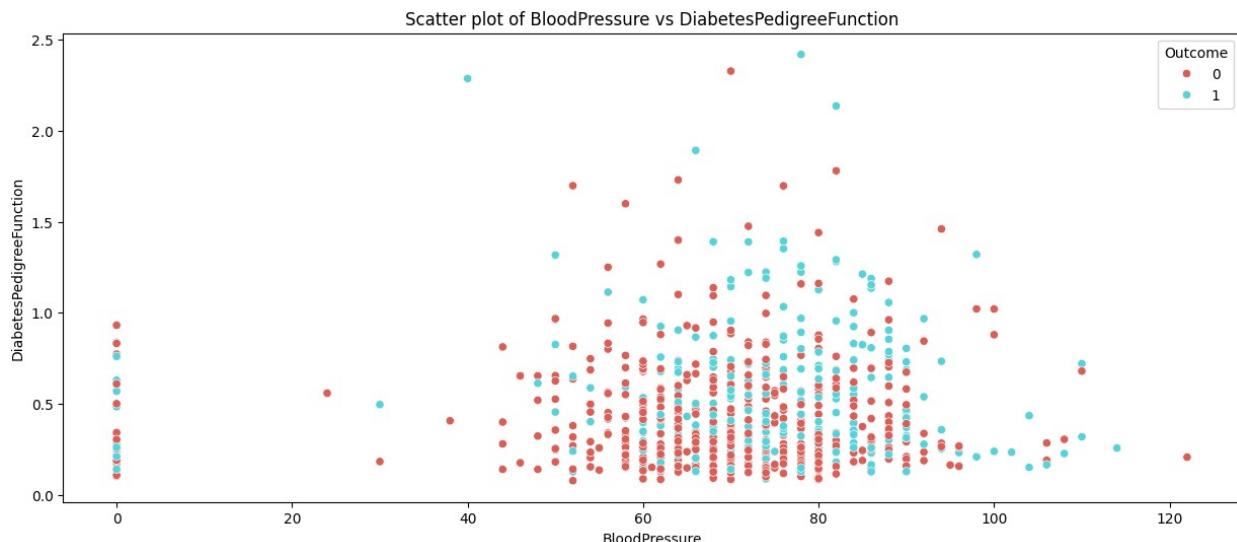


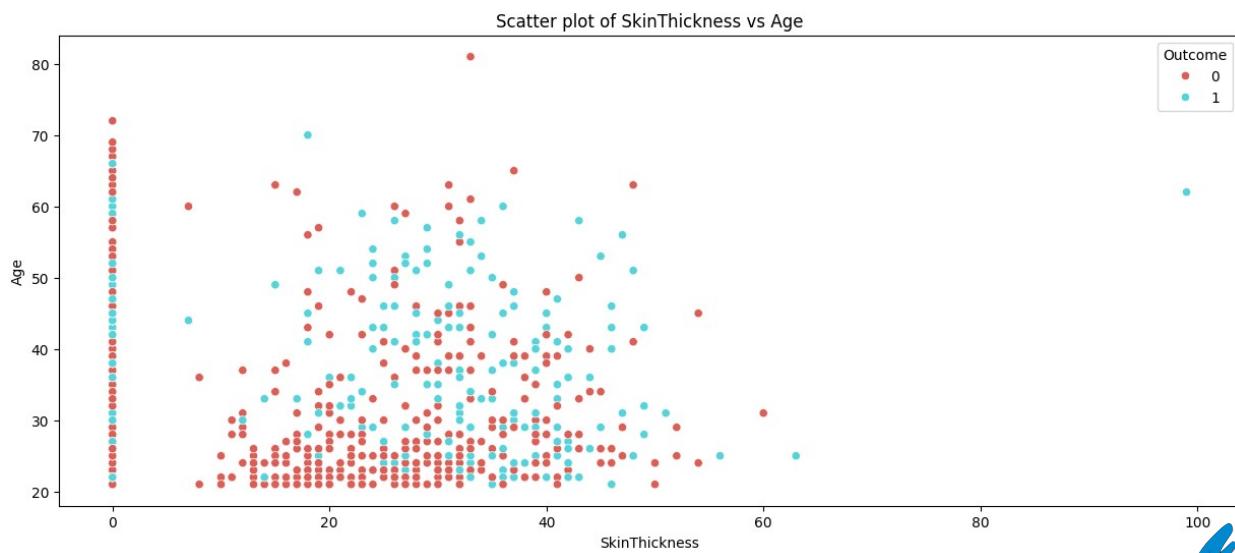
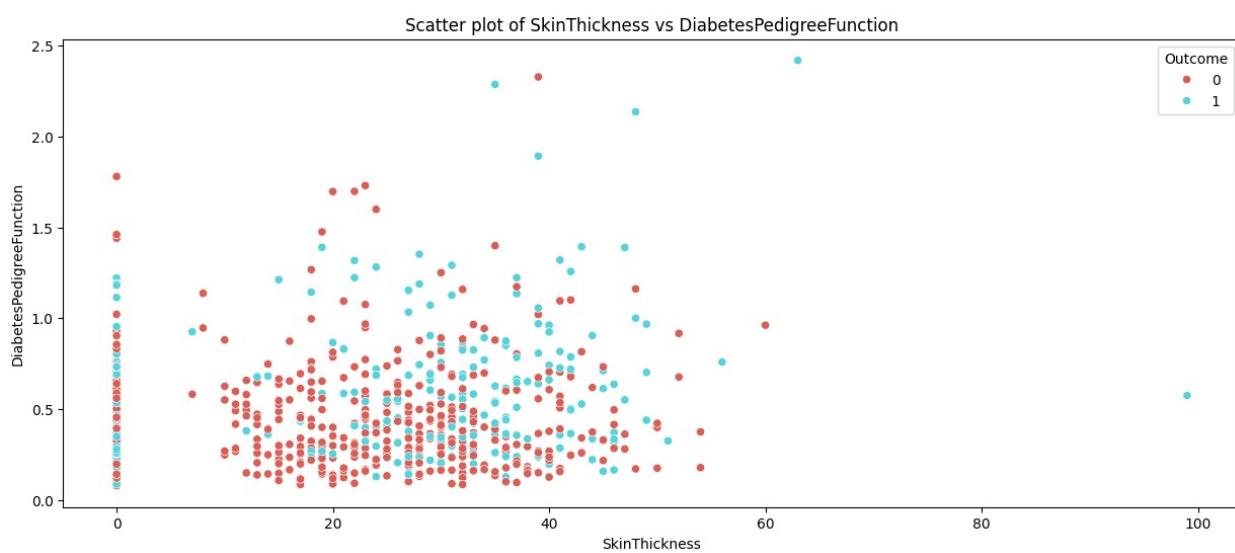
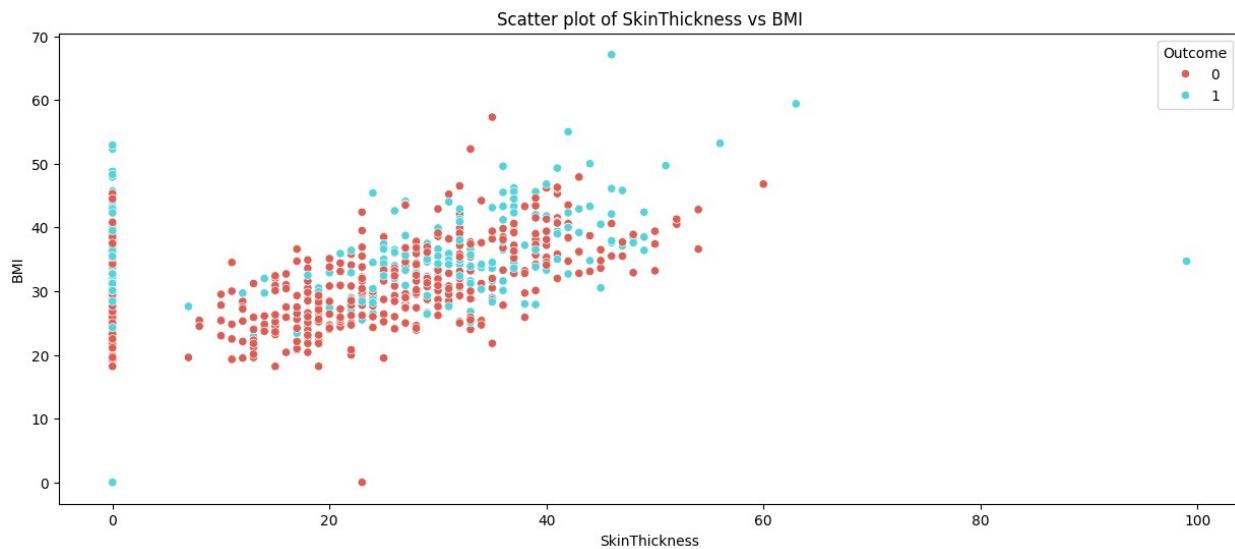


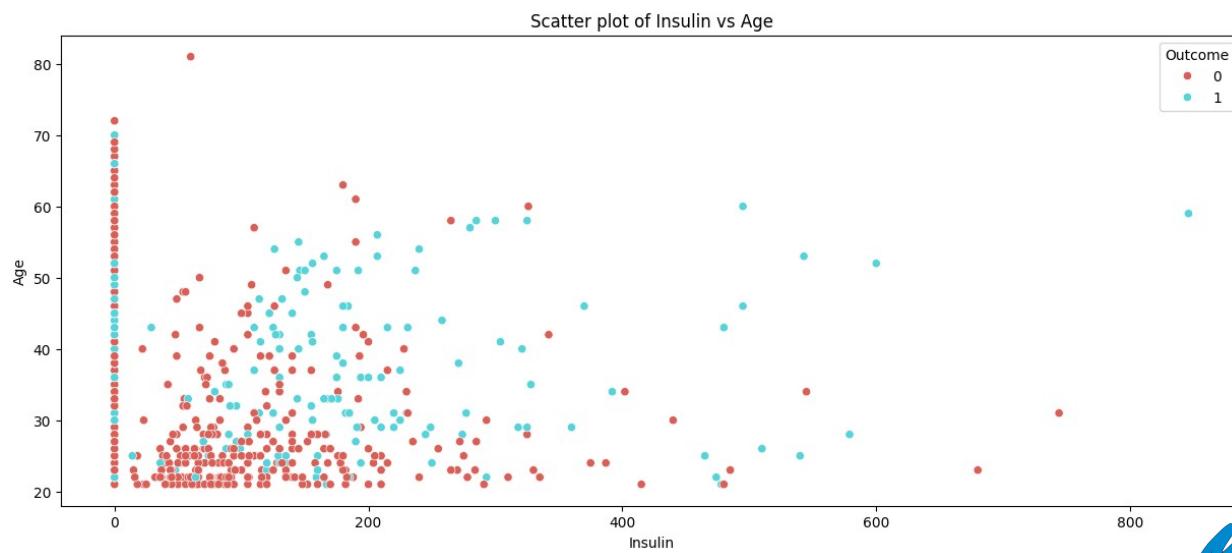
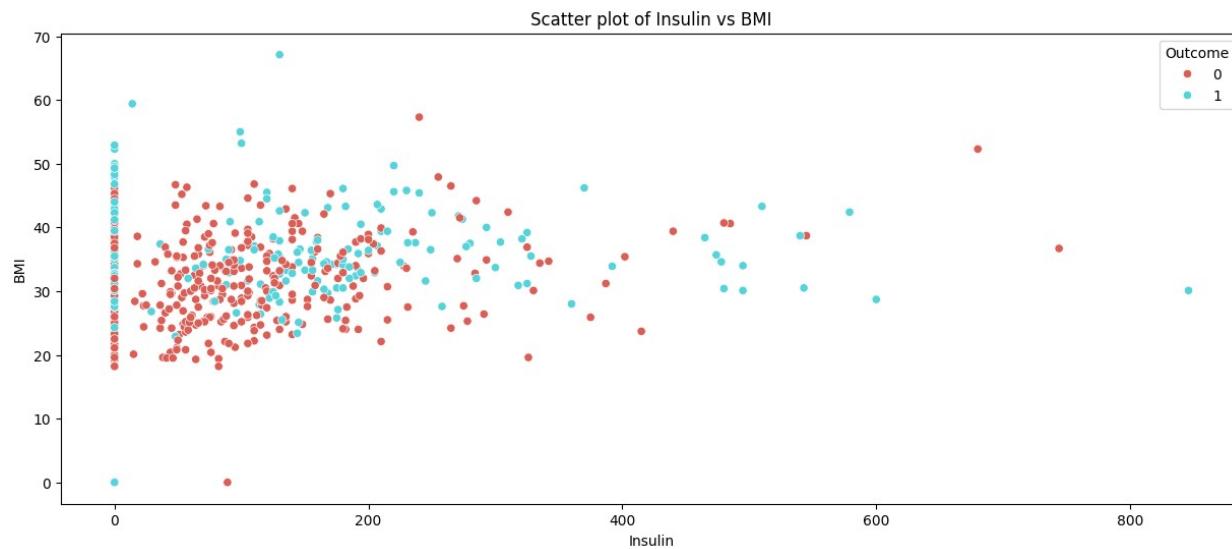


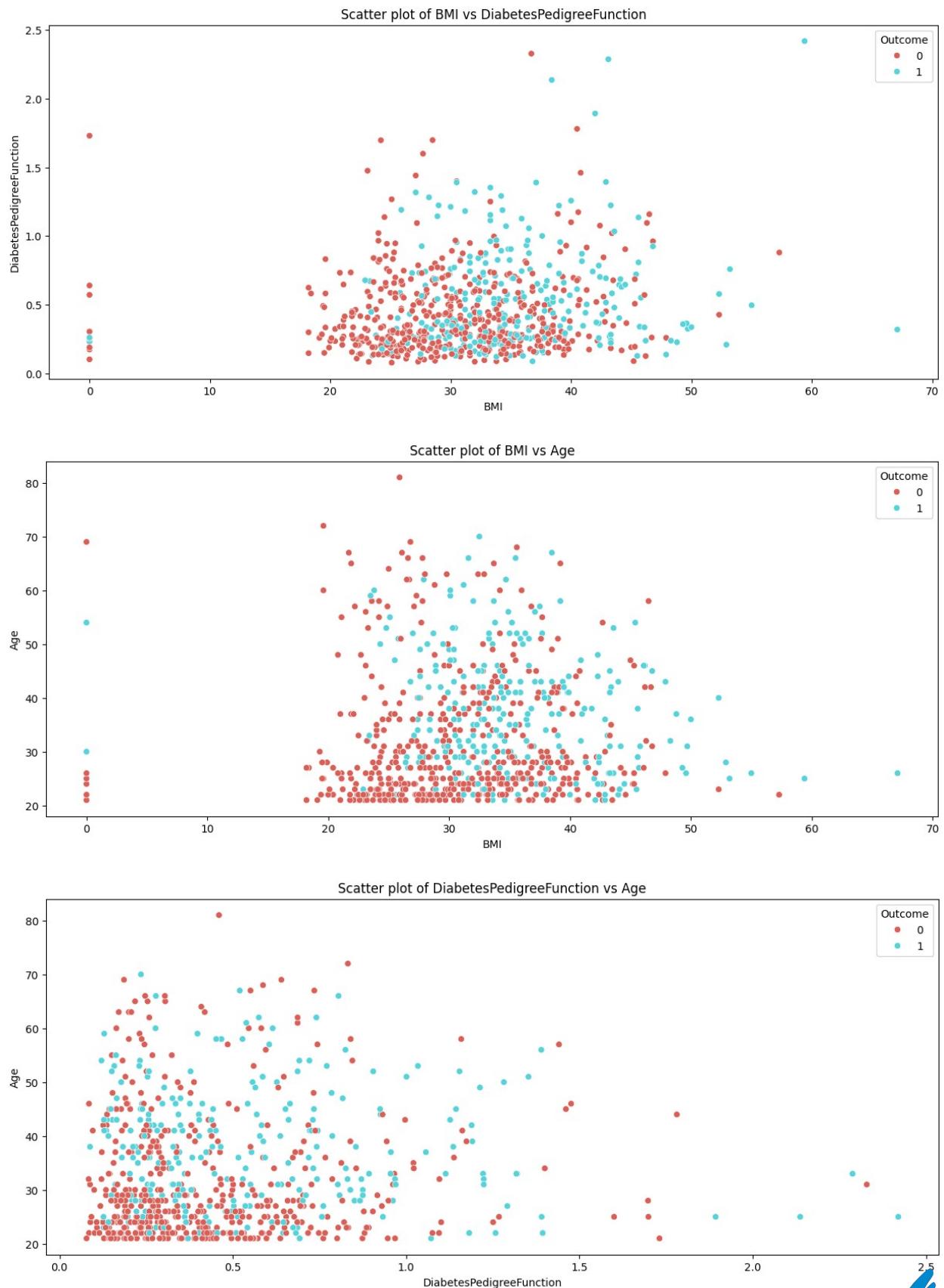








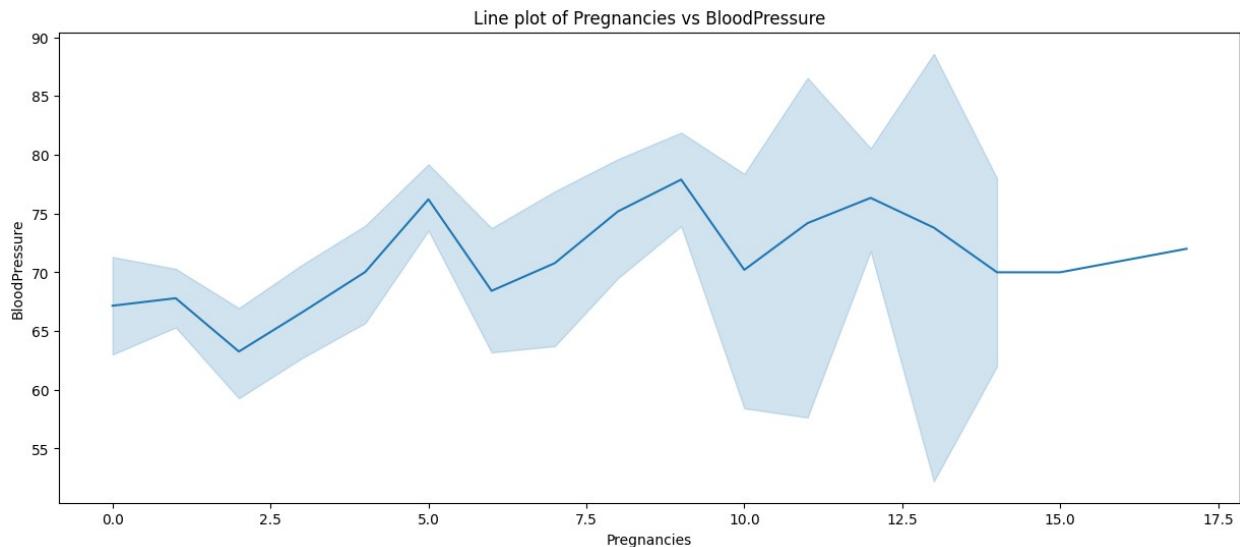
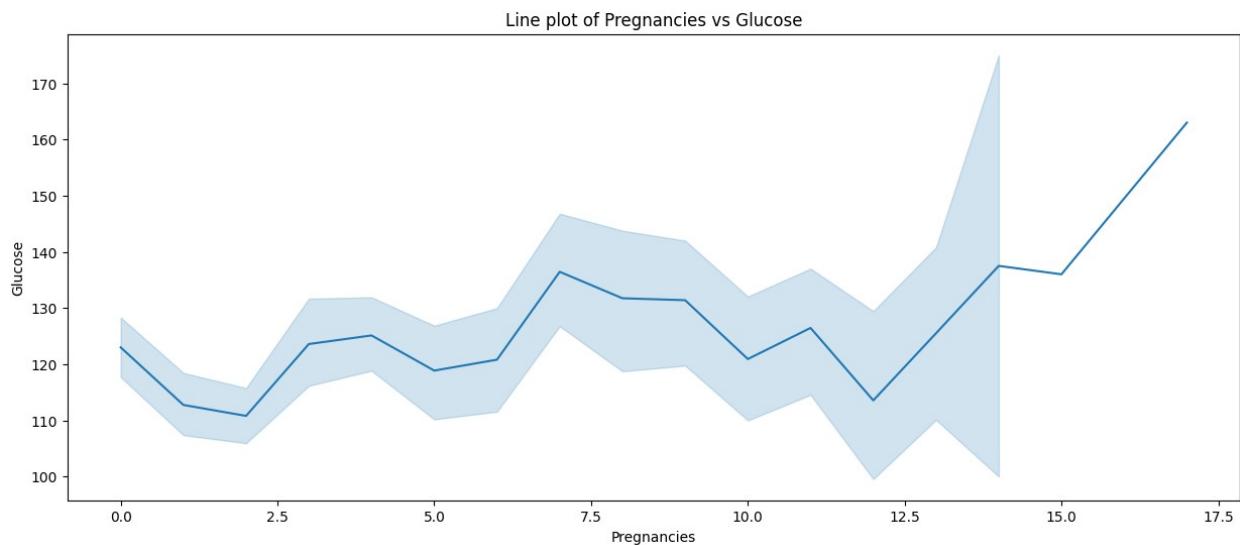




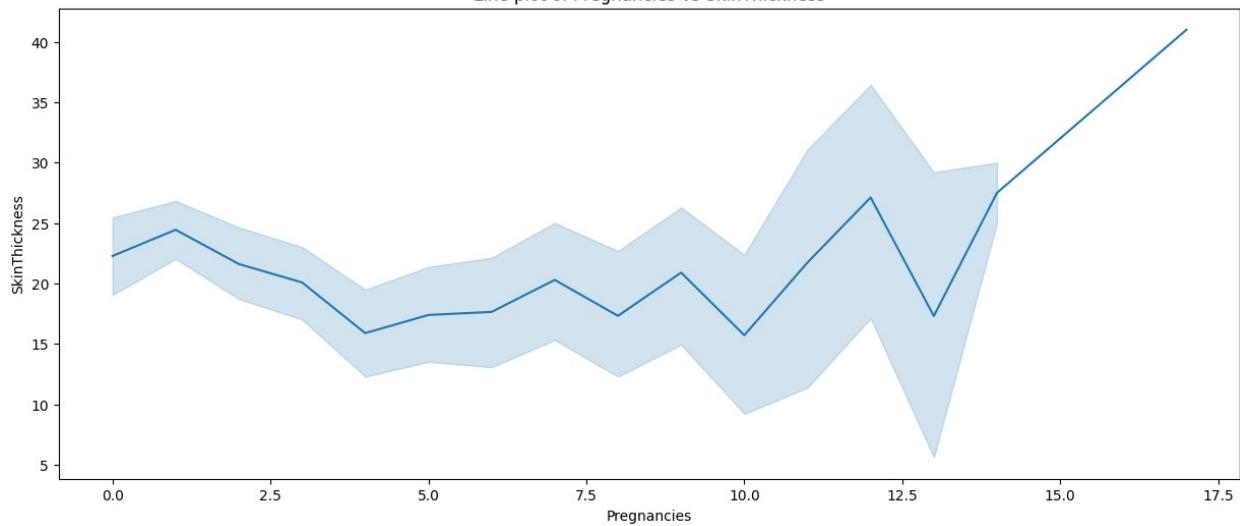
```

for i in range(len(continuous)):
    for j in range(i + 1, len(continuous)):
        plt.figure(figsize=(15, 6))
        sns.lineplot(x=continuous[i], y=continuous[j], data=df,
palettes='hls')
        plt.title(f'Line plot of {continuous[i]} vs {continuous[j]}')
        plt.show()

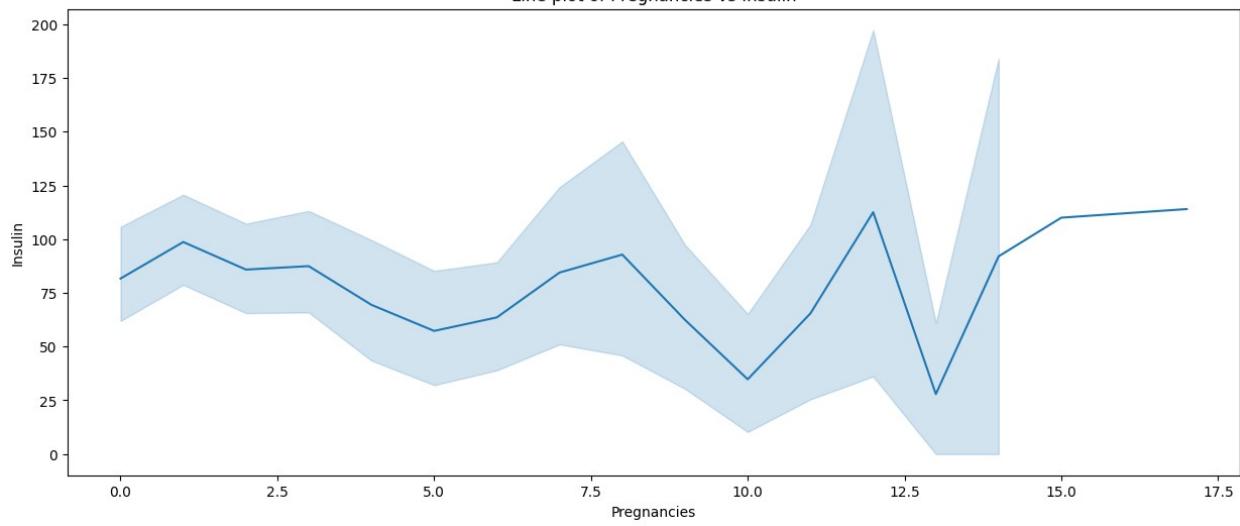
```



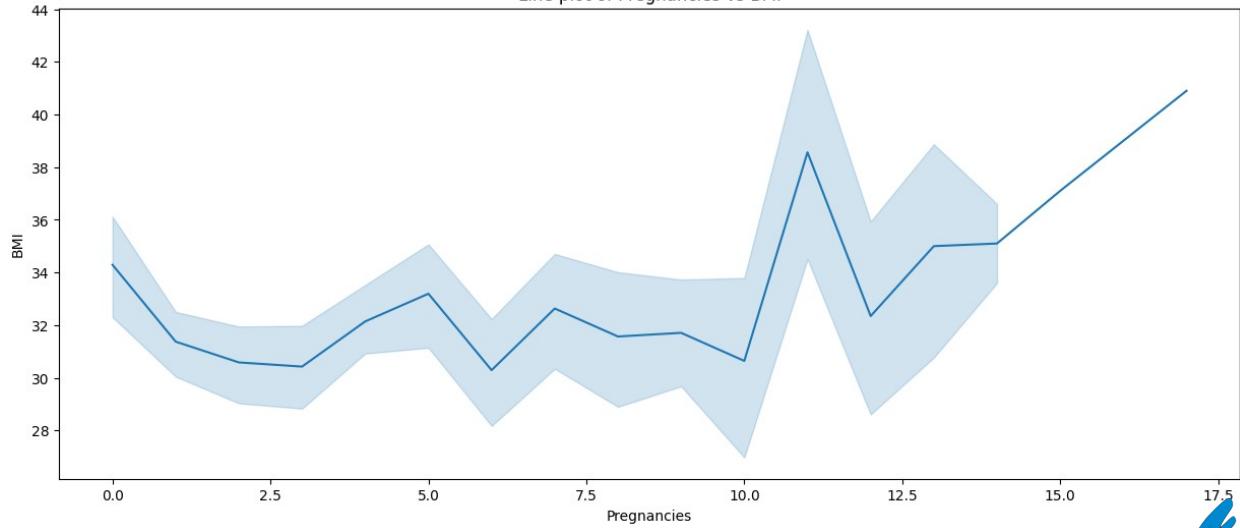
Line plot of Pregnancies vs SkinThickness

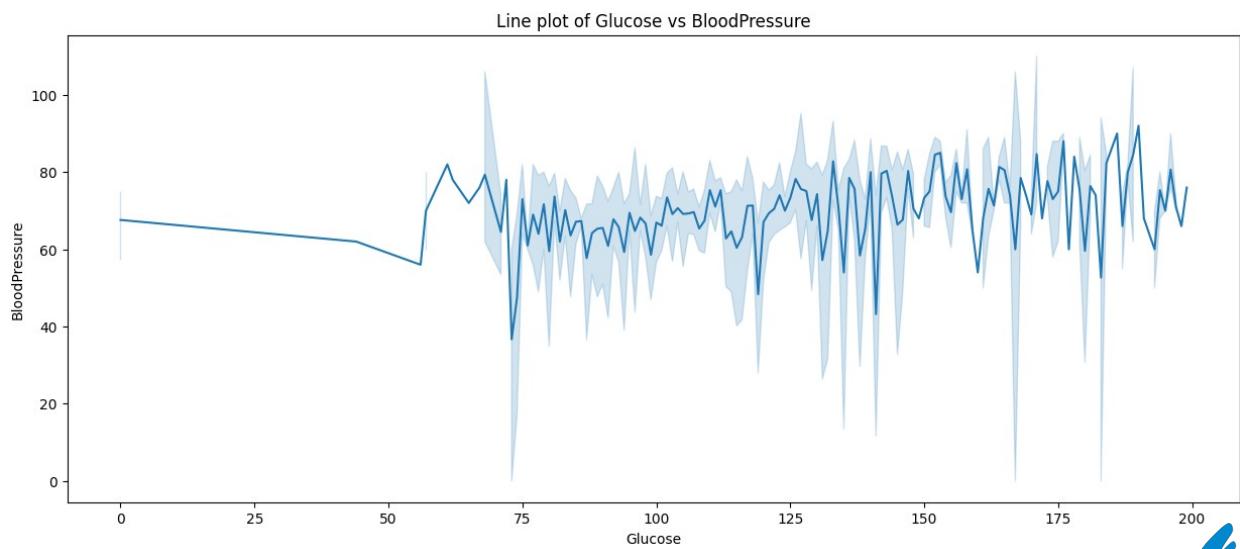
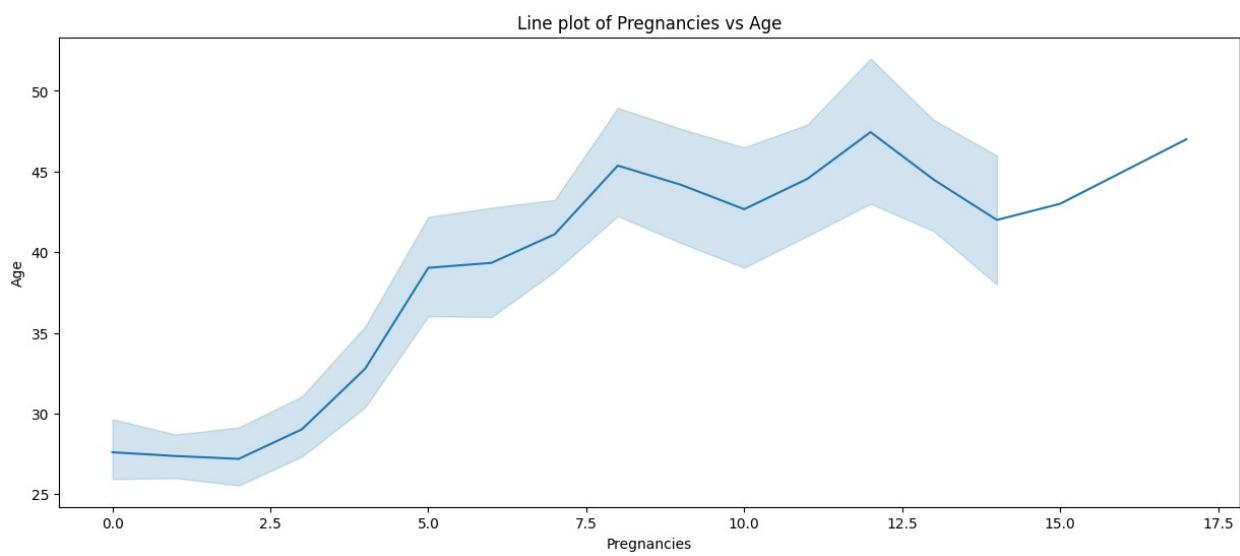
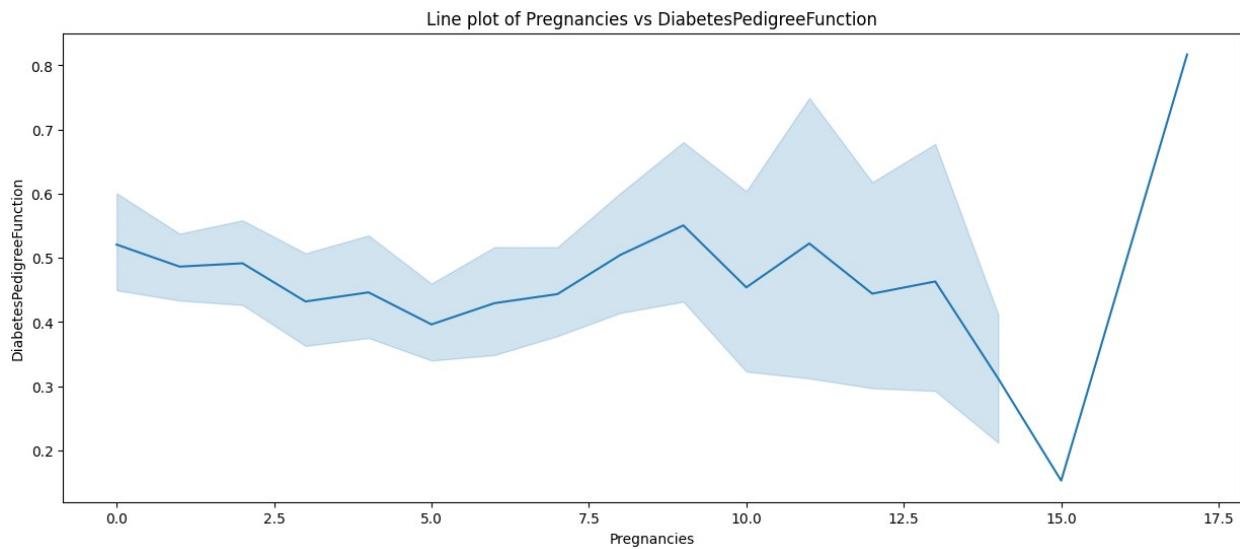


Line plot of Pregnancies vs Insulin

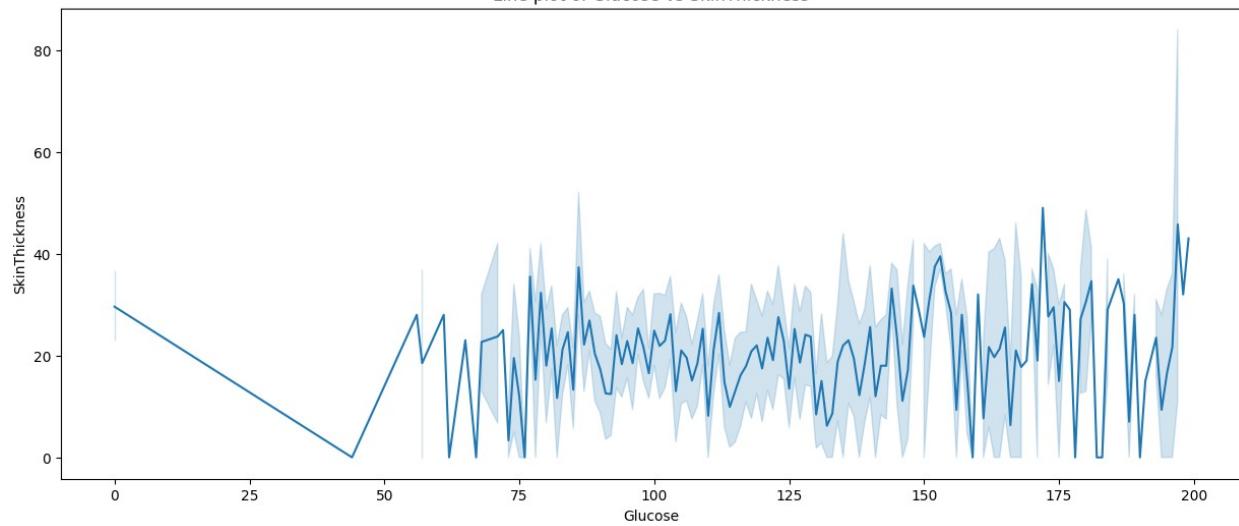


Line plot of Pregnancies vs BMI

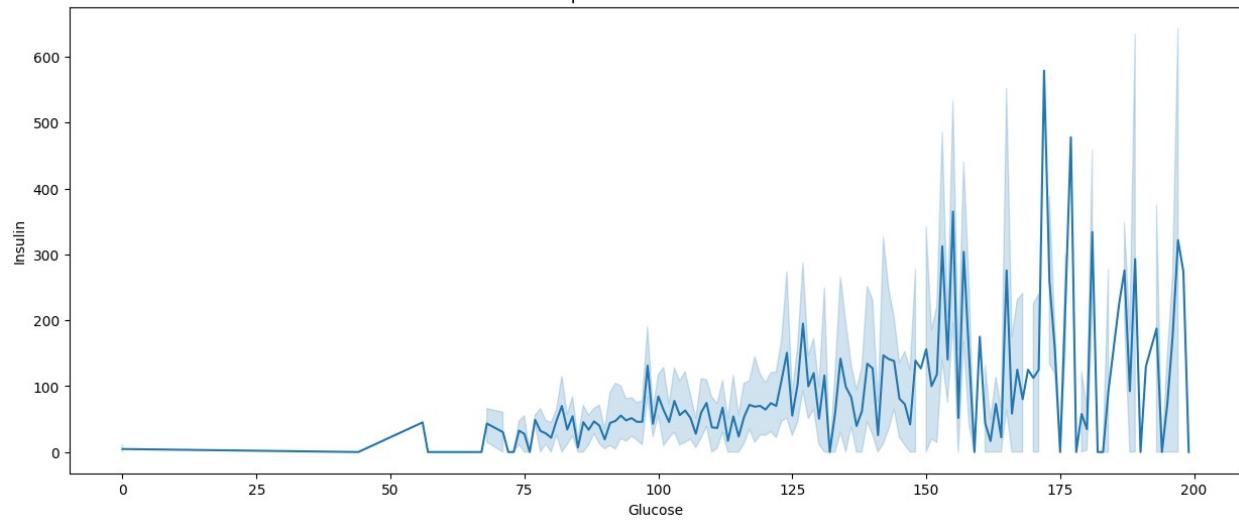




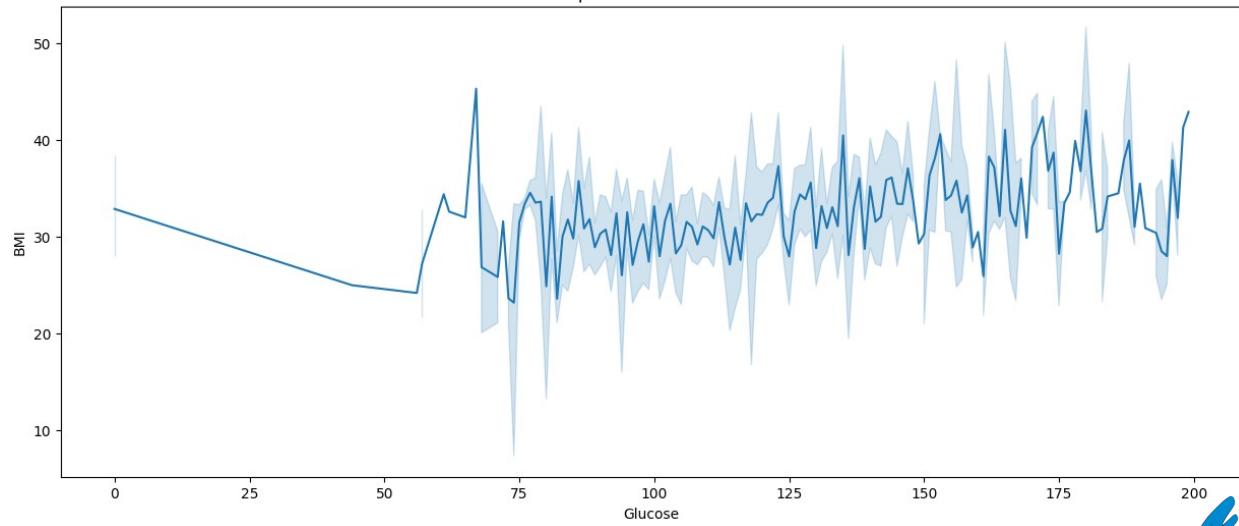
Line plot of Glucose vs SkinThickness

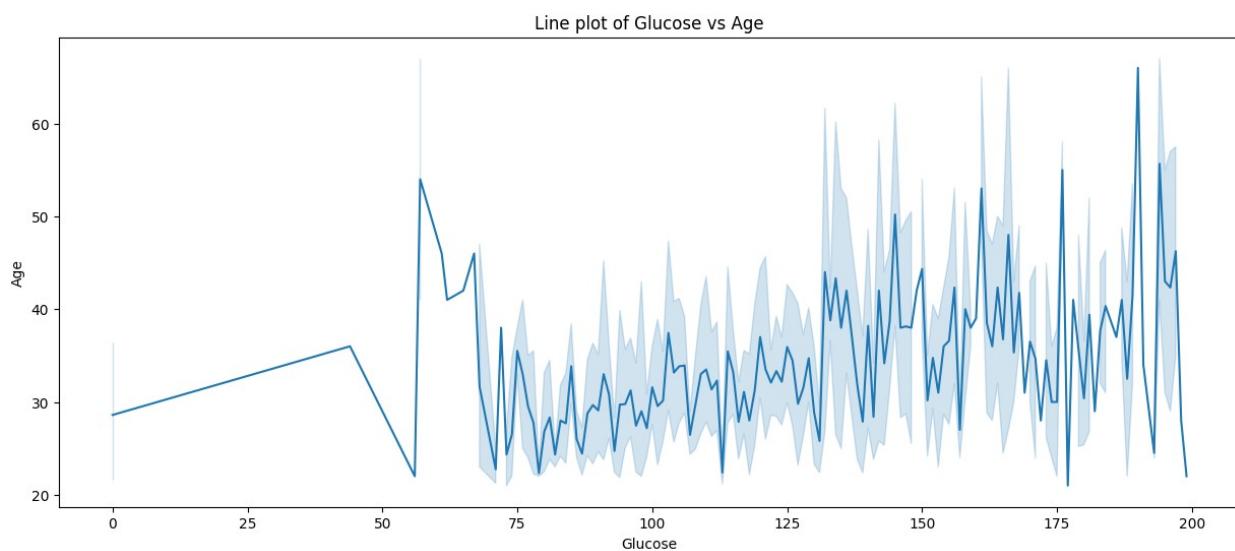
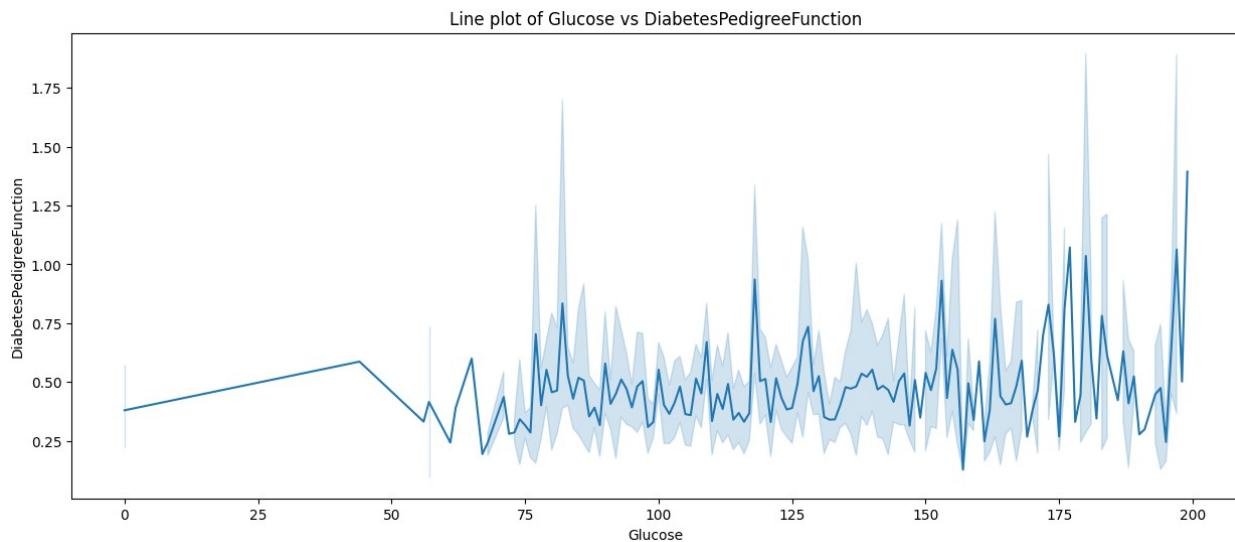


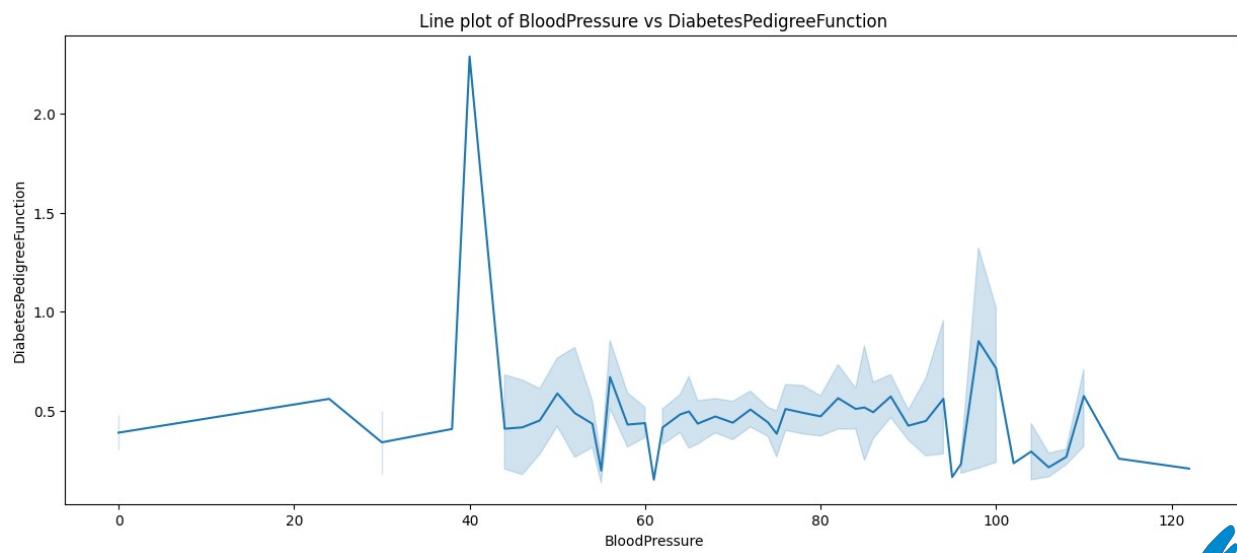
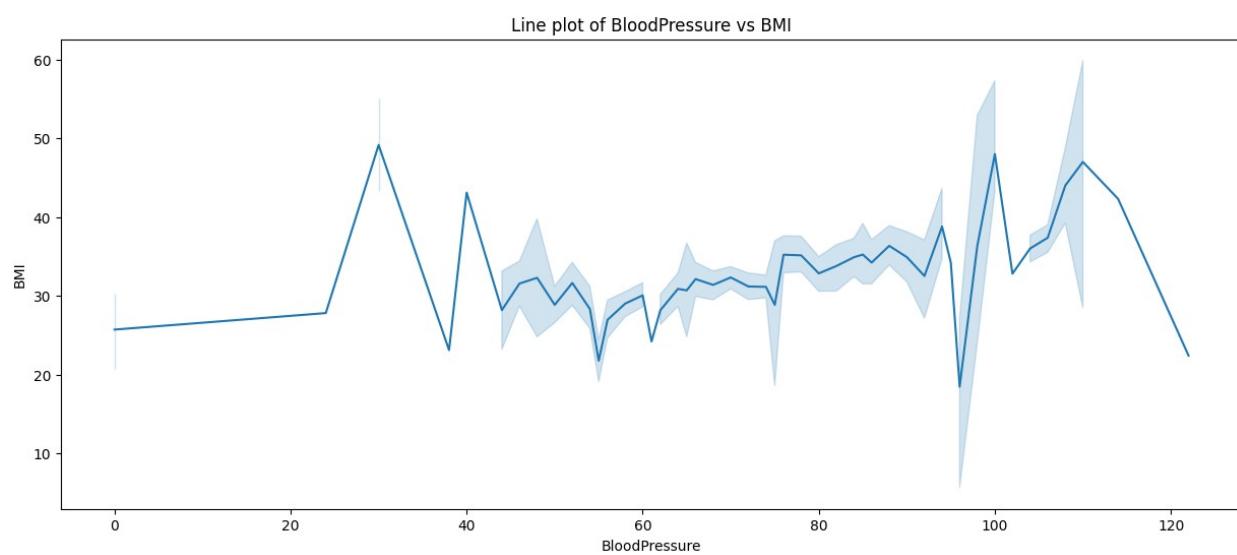
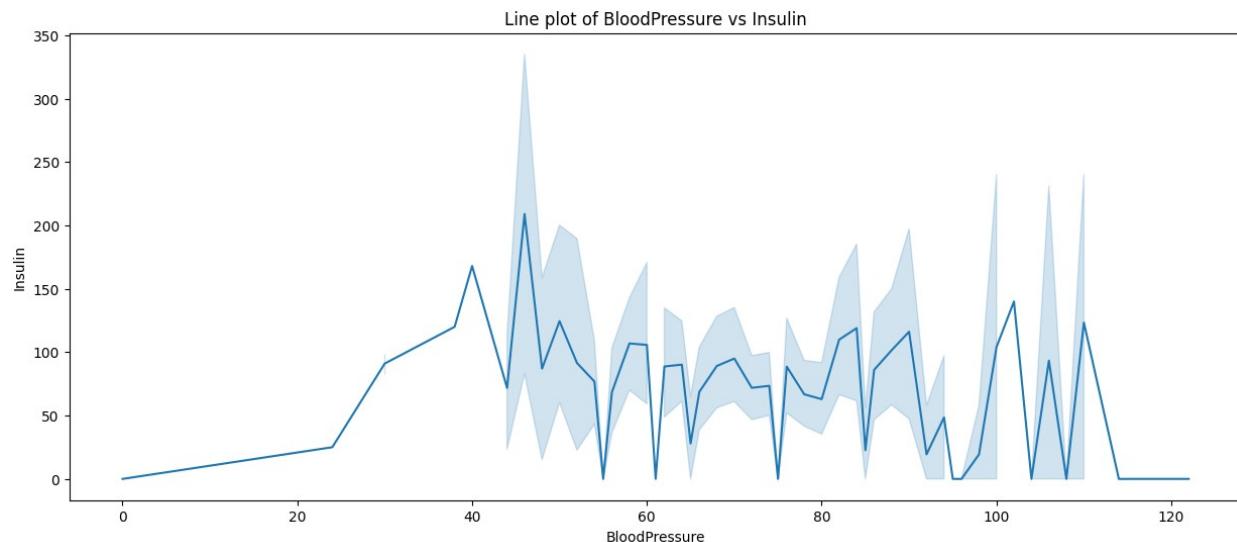
Line plot of Glucose vs Insulin



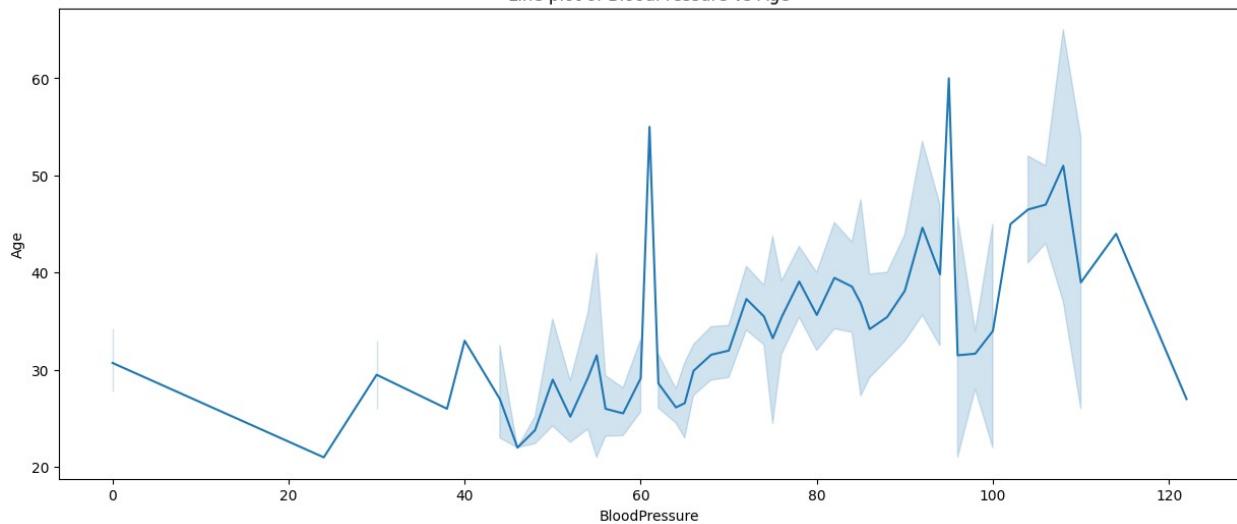
Line plot of Glucose vs BMI



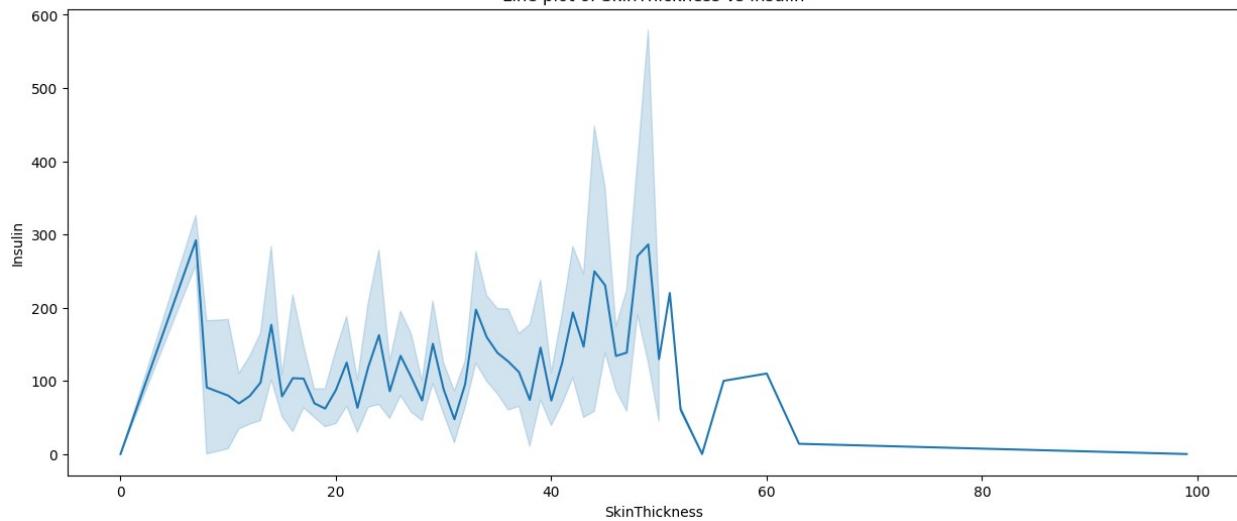




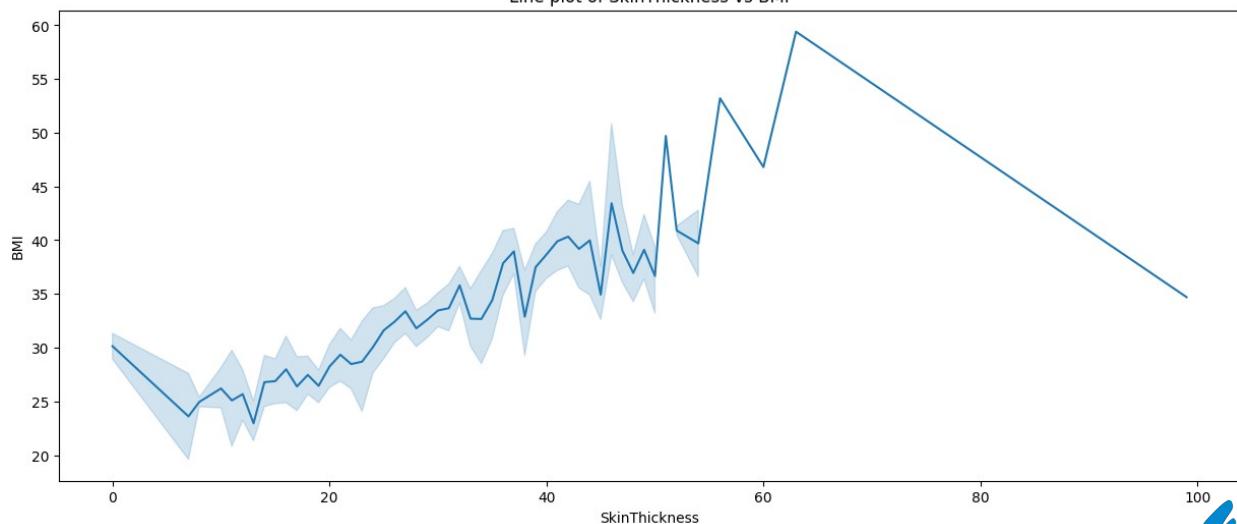
Line plot of BloodPressure vs Age

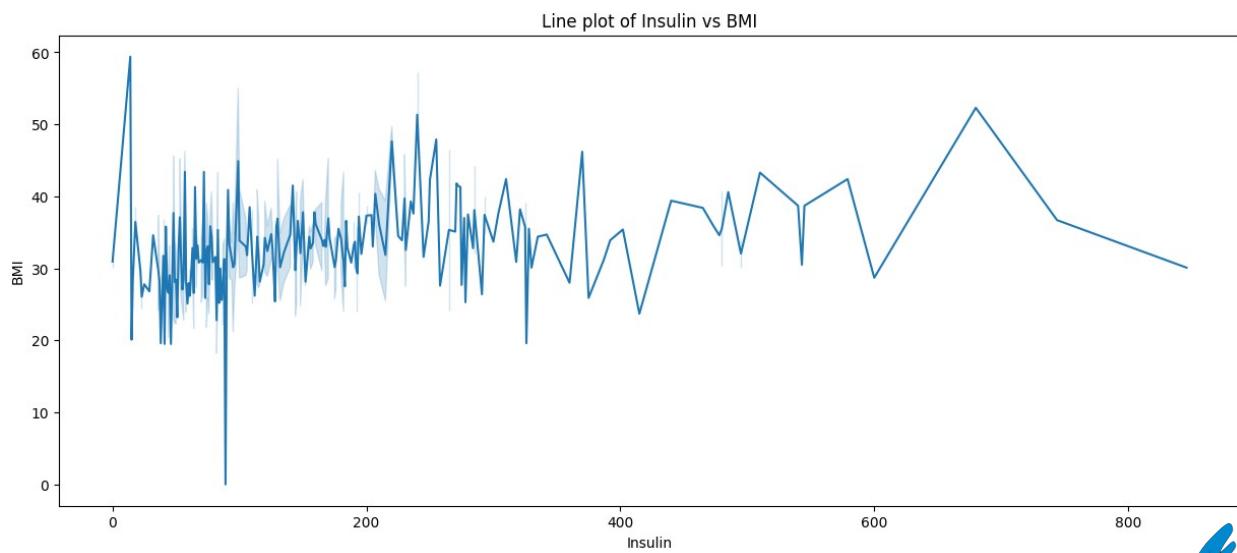
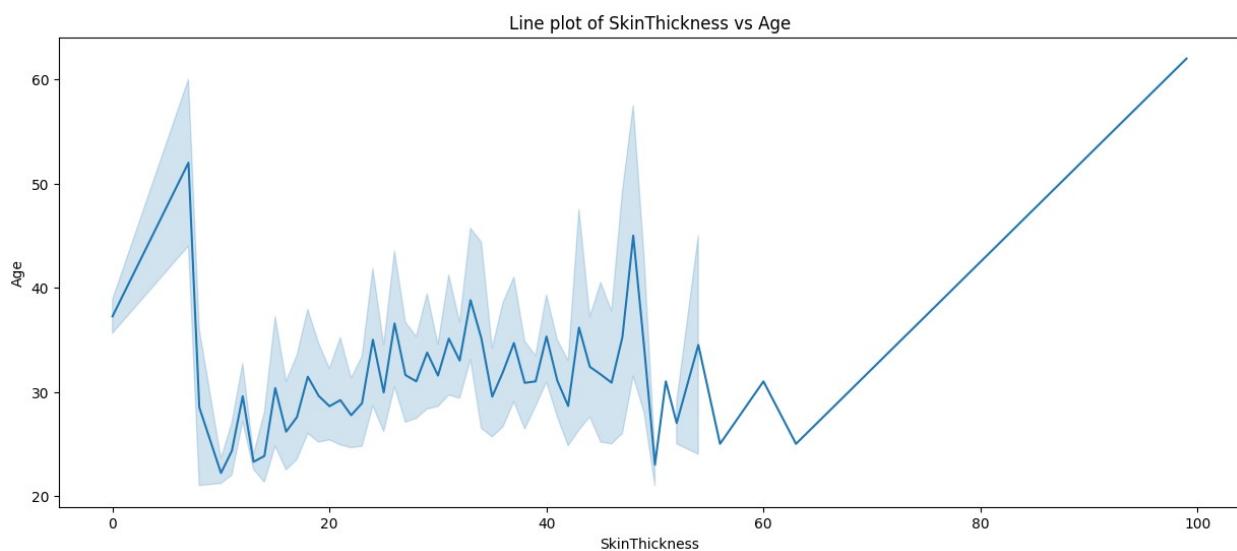
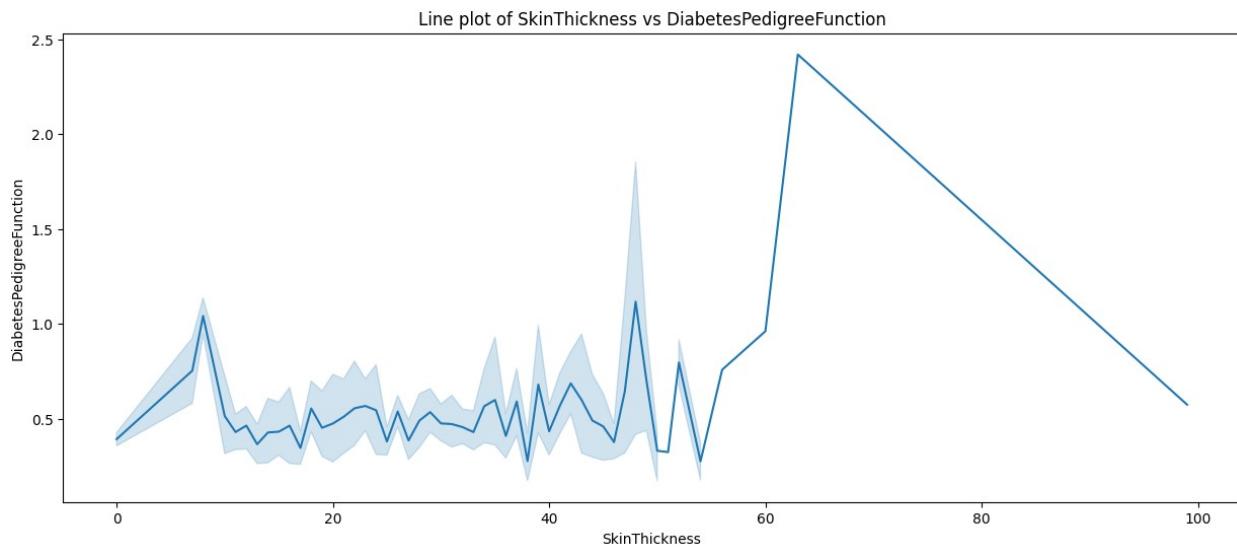


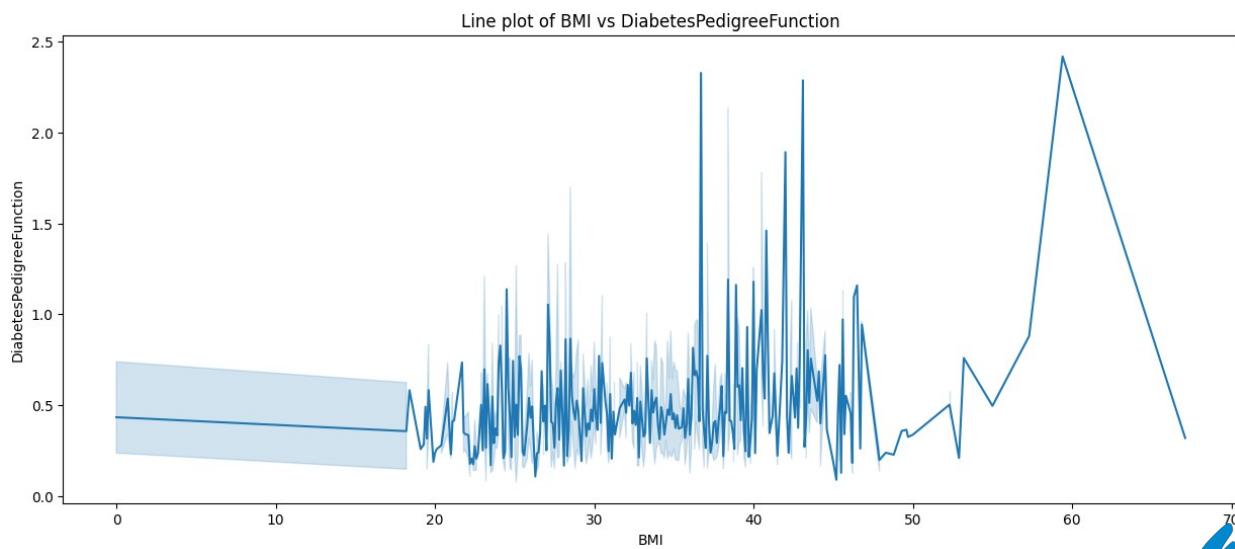
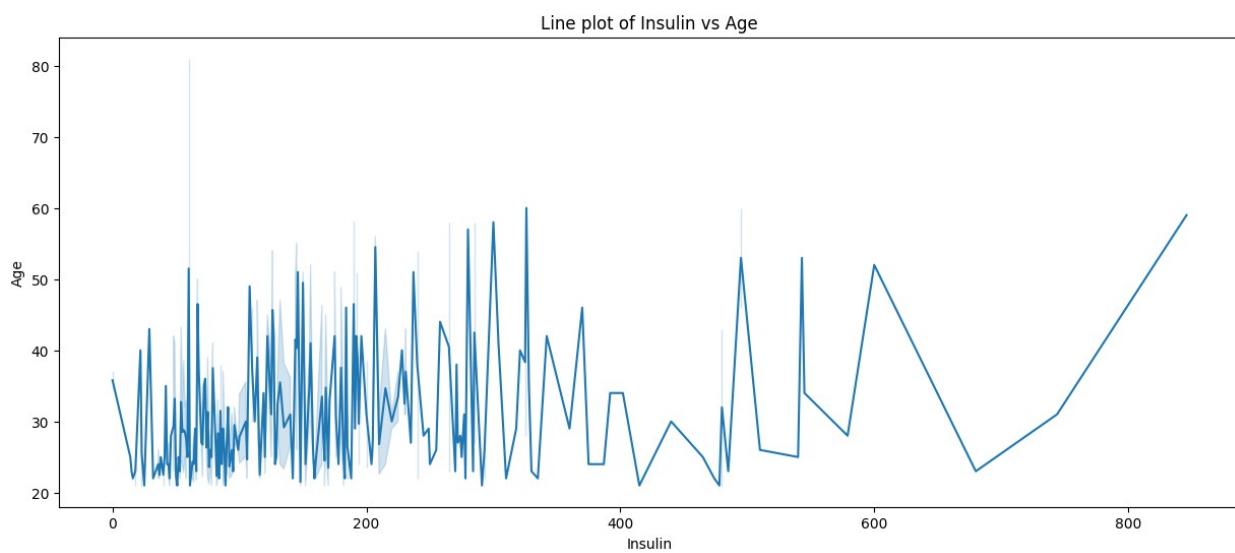
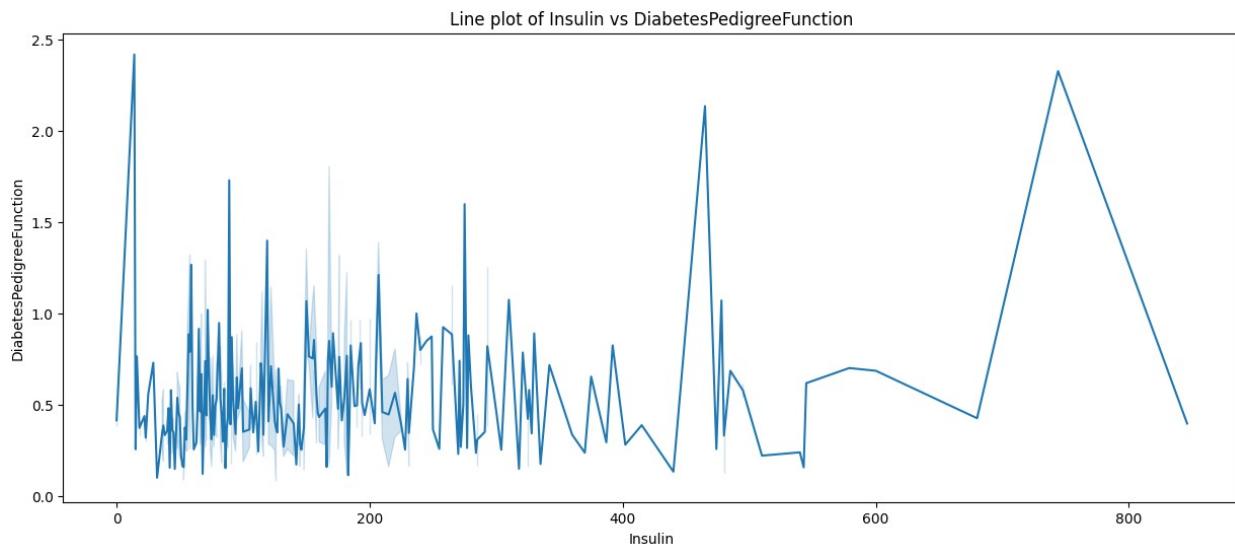
Line plot of SkinThickness vs Insulin

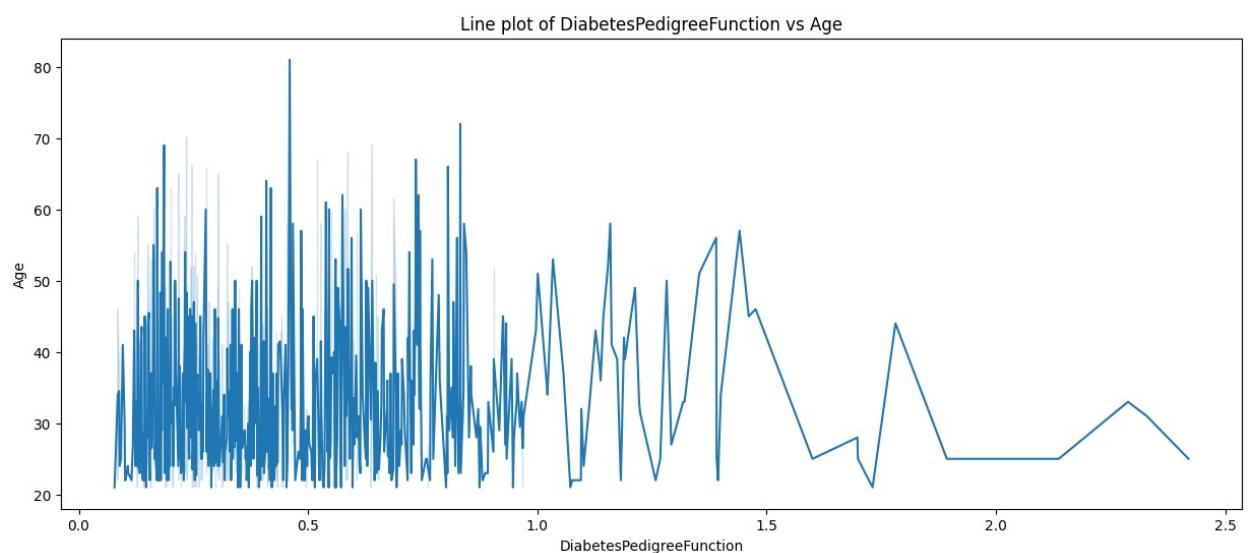
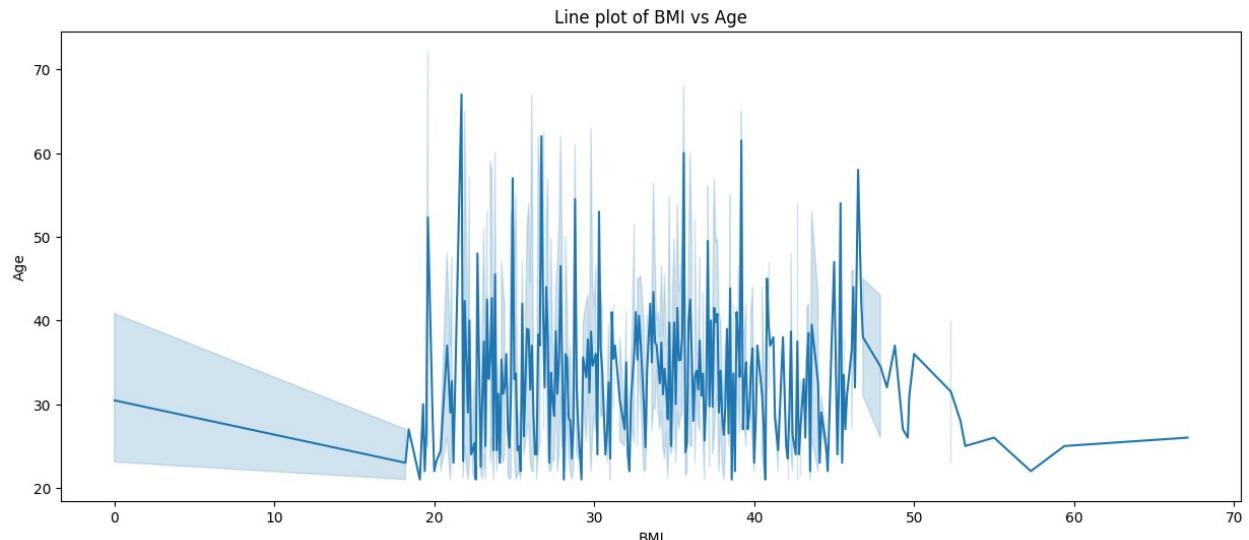


Line plot of SkinThickness vs BMI





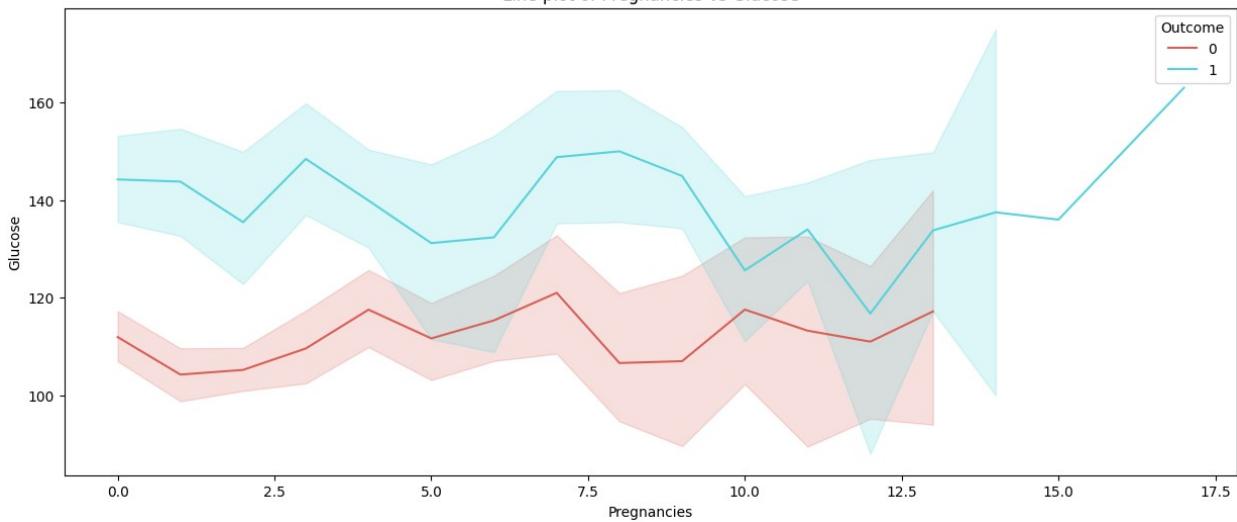




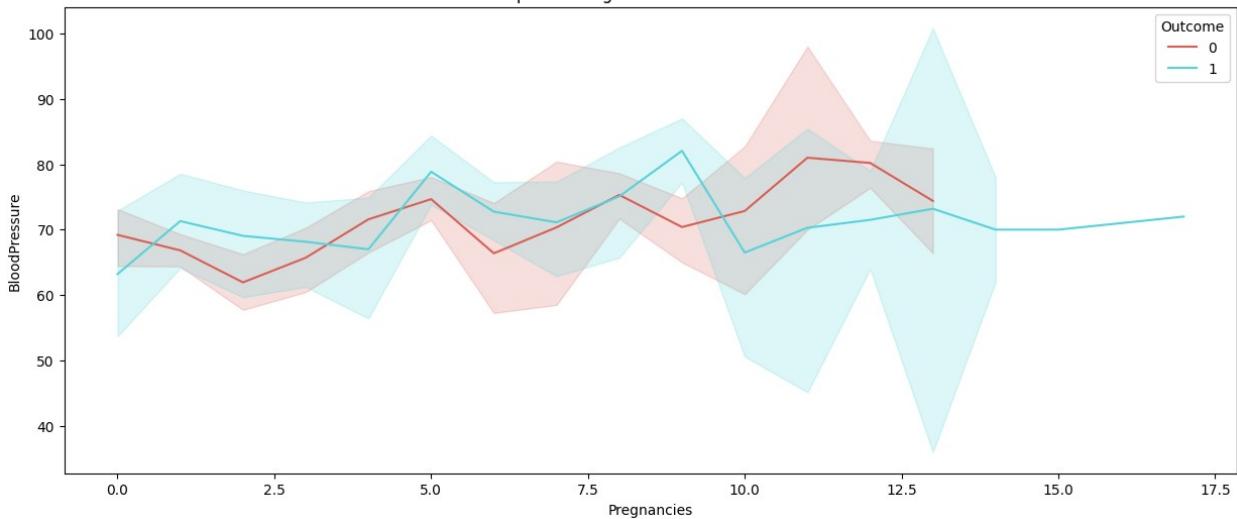
```

for i in range(len(continuous)):
    for j in range(i + 1, len(continuous)):
        plt.figure(figsize=(15, 6))
        sns.lineplot(x=continuous[i], y=continuous[j], data=df, hue = 'Outcome', palette='hls')
        plt.title(f'Line plot of {continuous[i]} vs {continuous[j]}')
        plt.show()
    
```

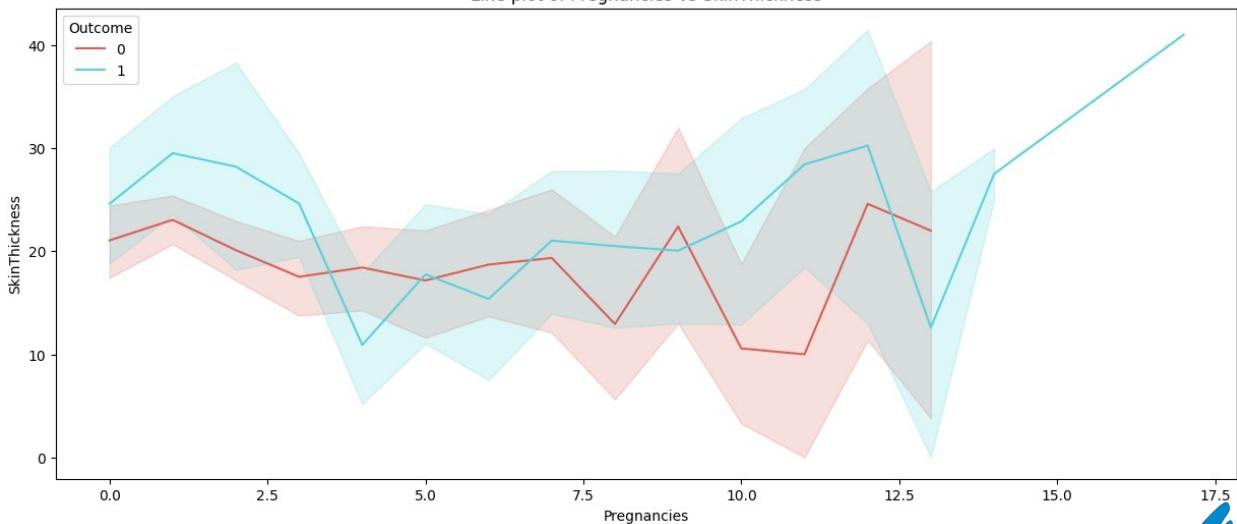
Line plot of Pregnancies vs Glucose

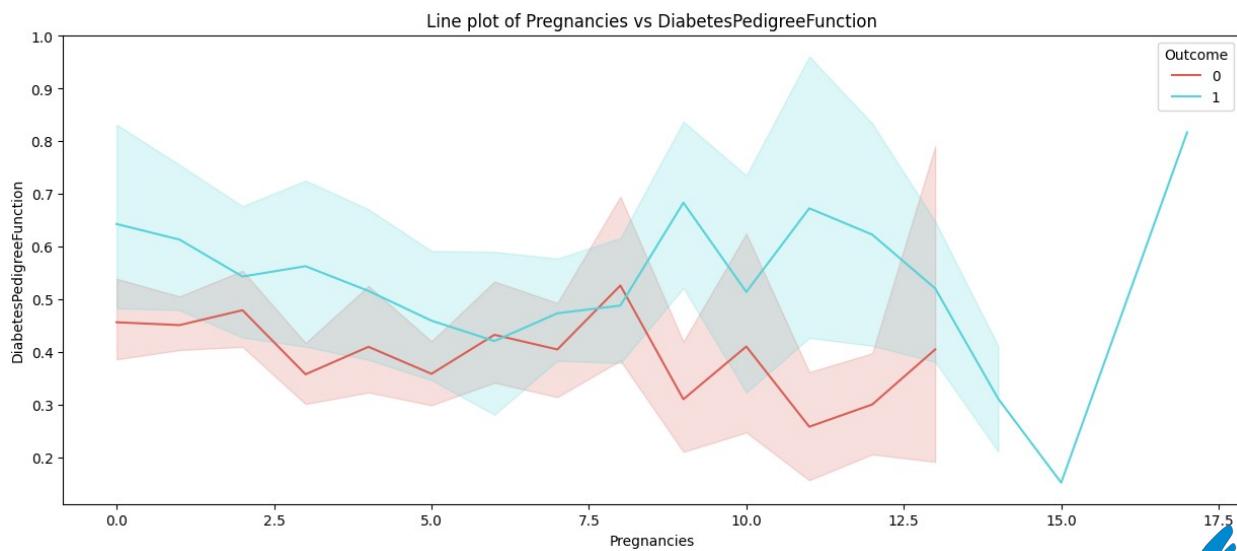
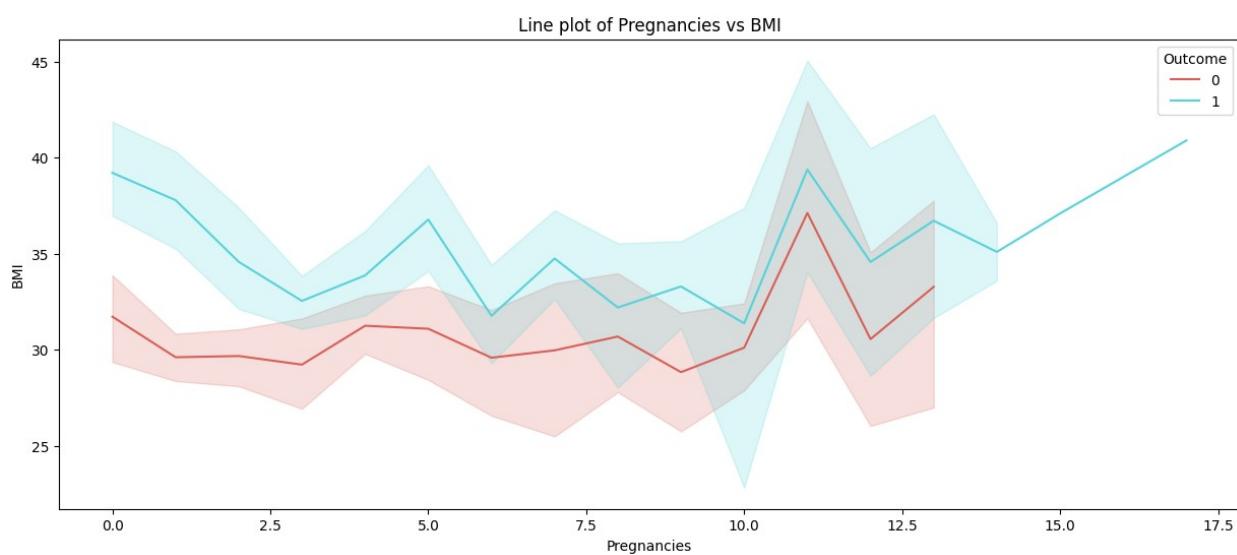
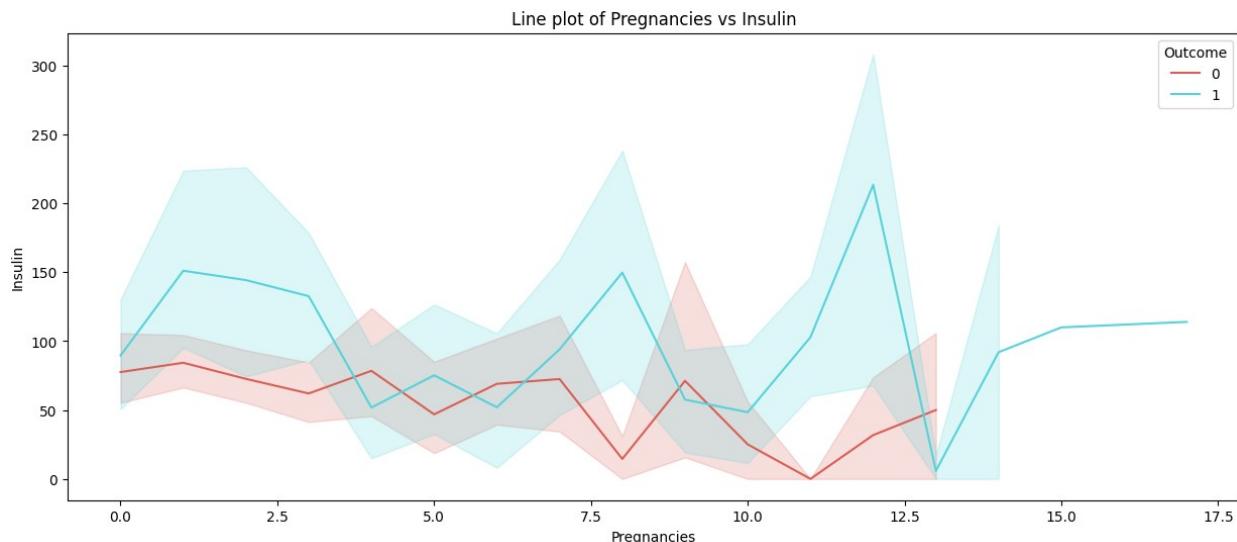


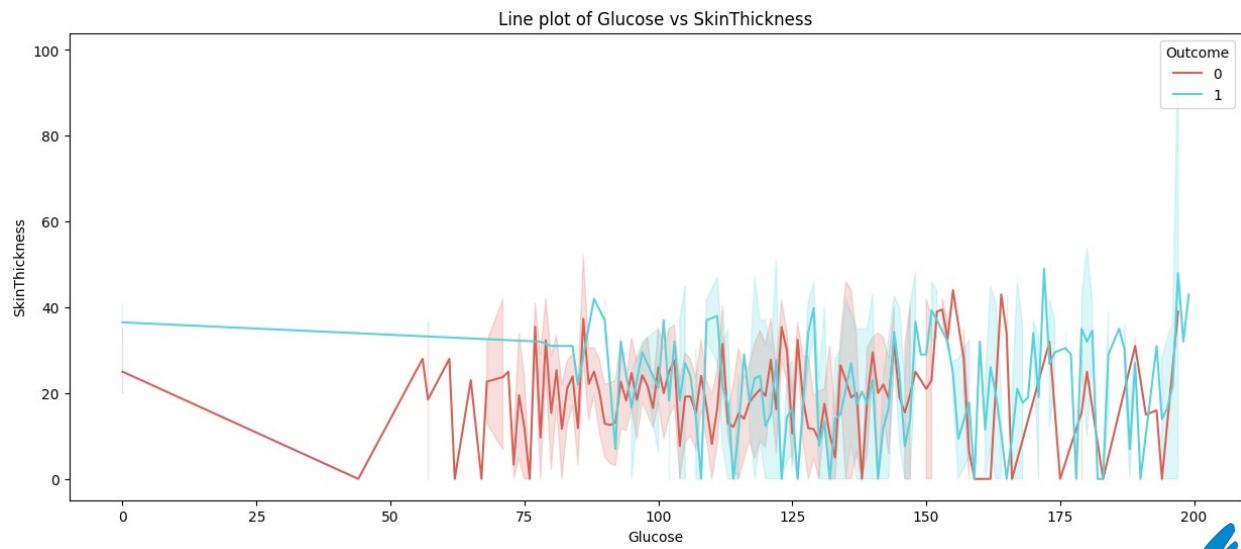
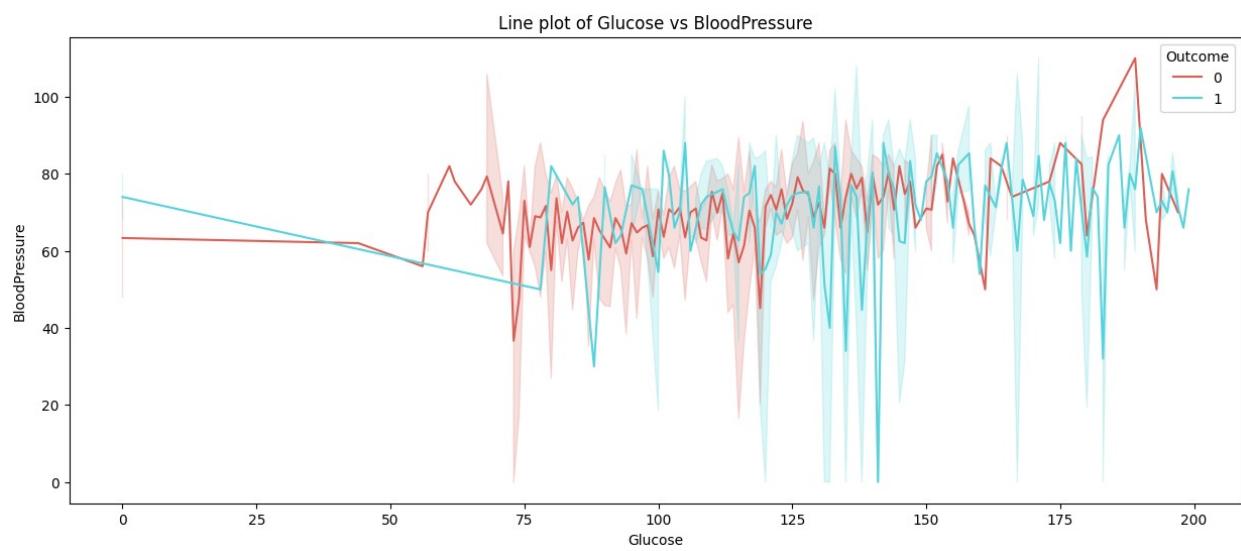
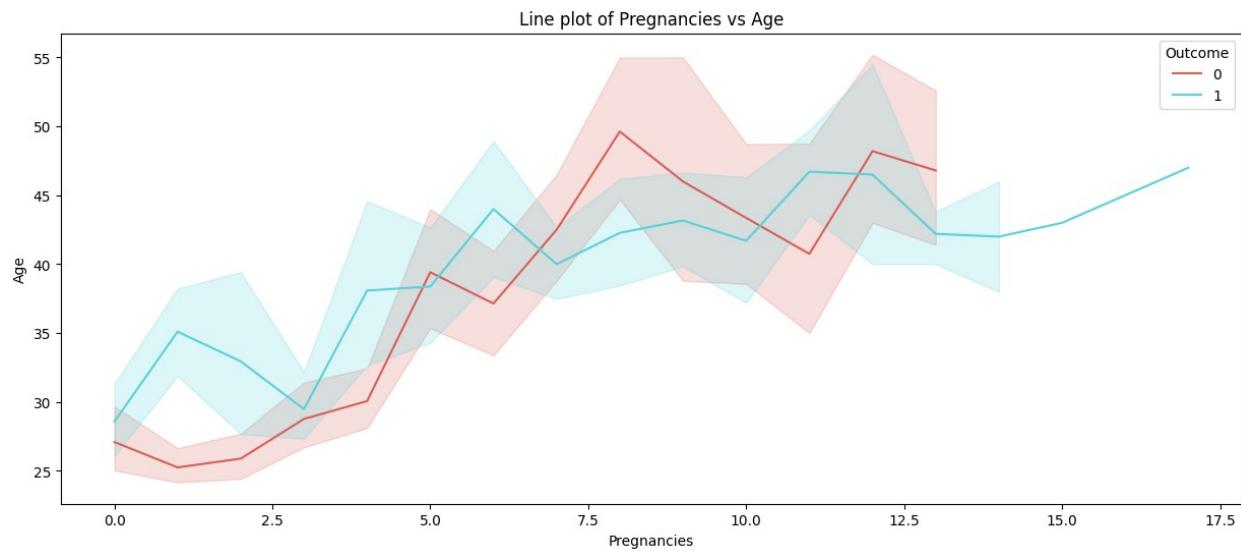
Line plot of Pregnancies vs BloodPressure



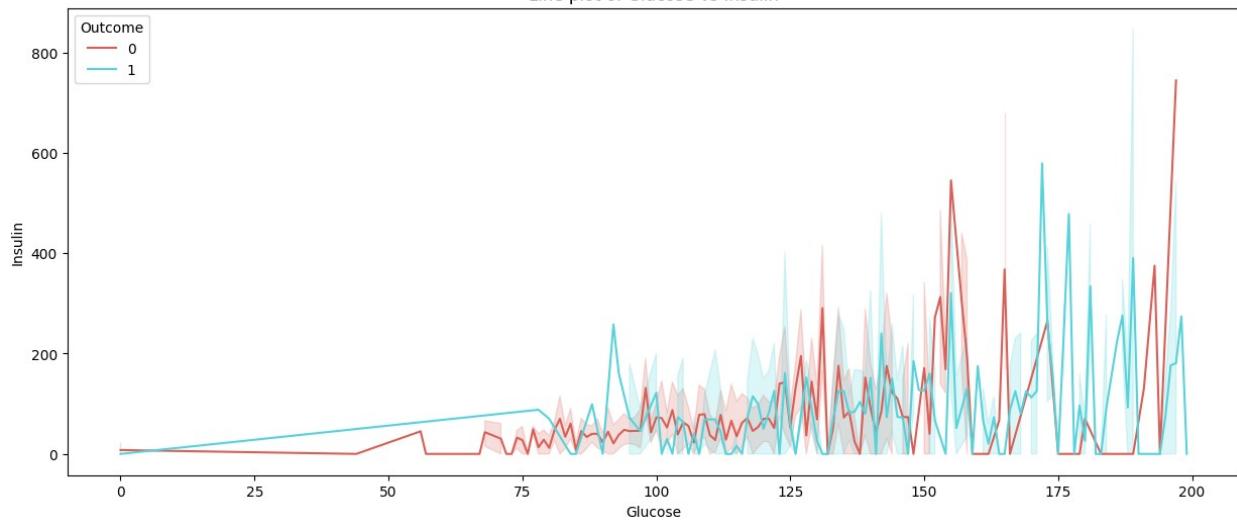
Line plot of Pregnancies vs SkinThickness



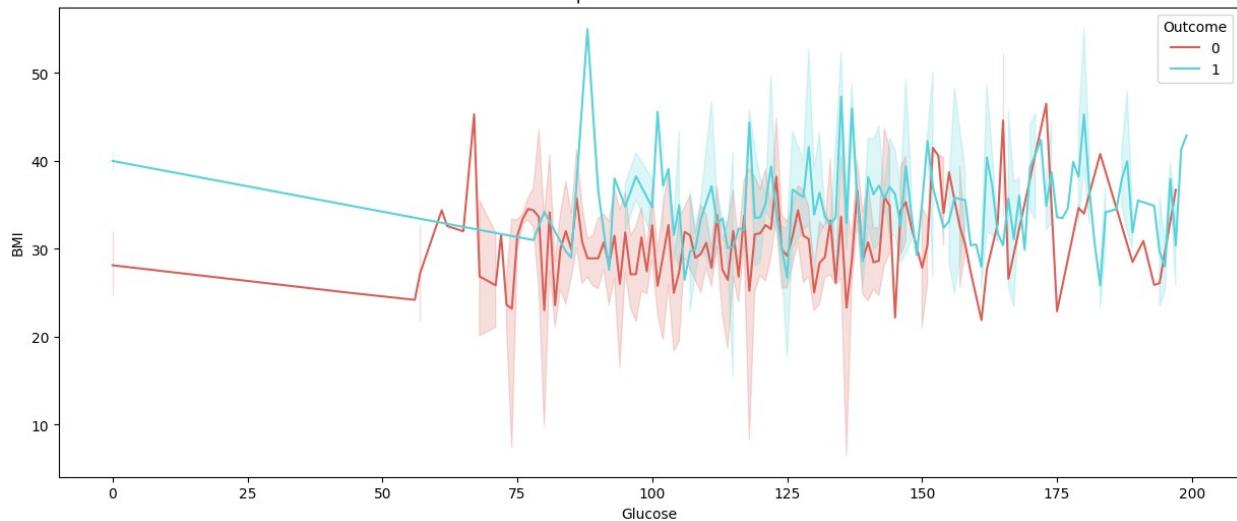




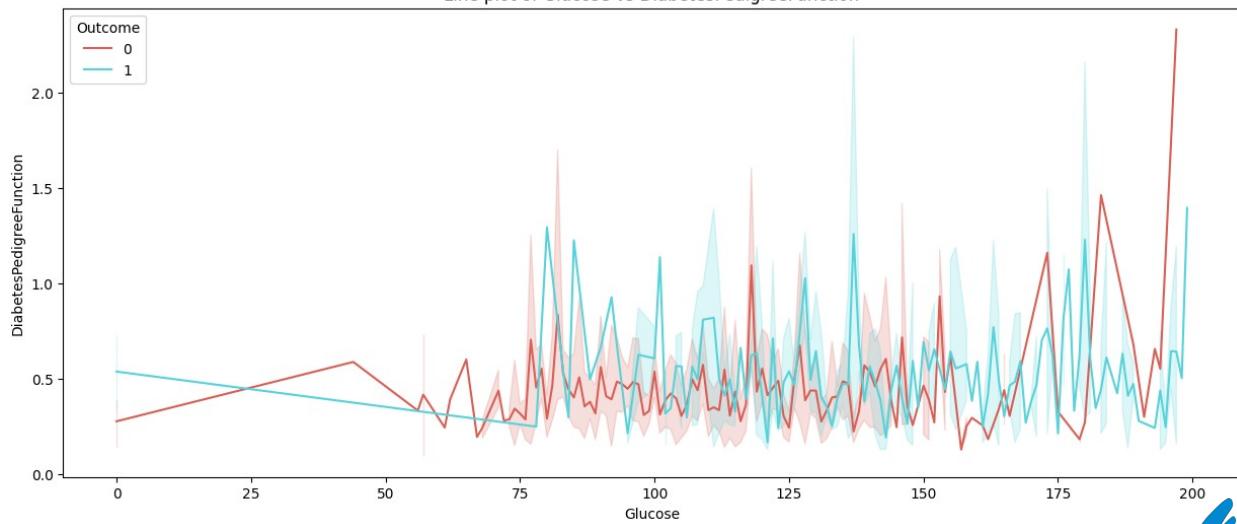
Line plot of Glucose vs Insulin

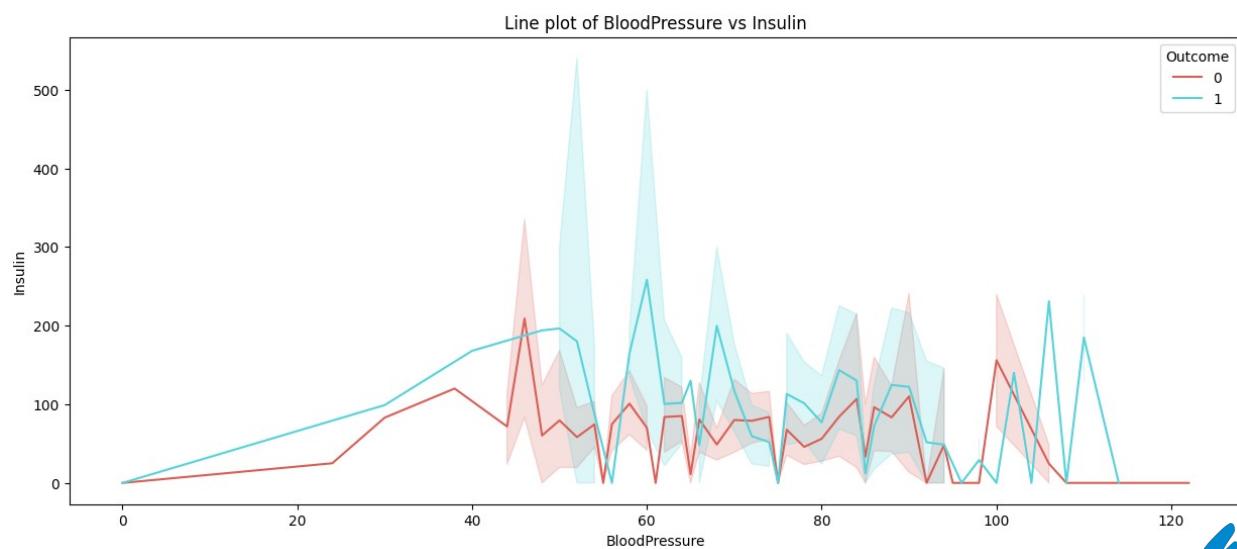
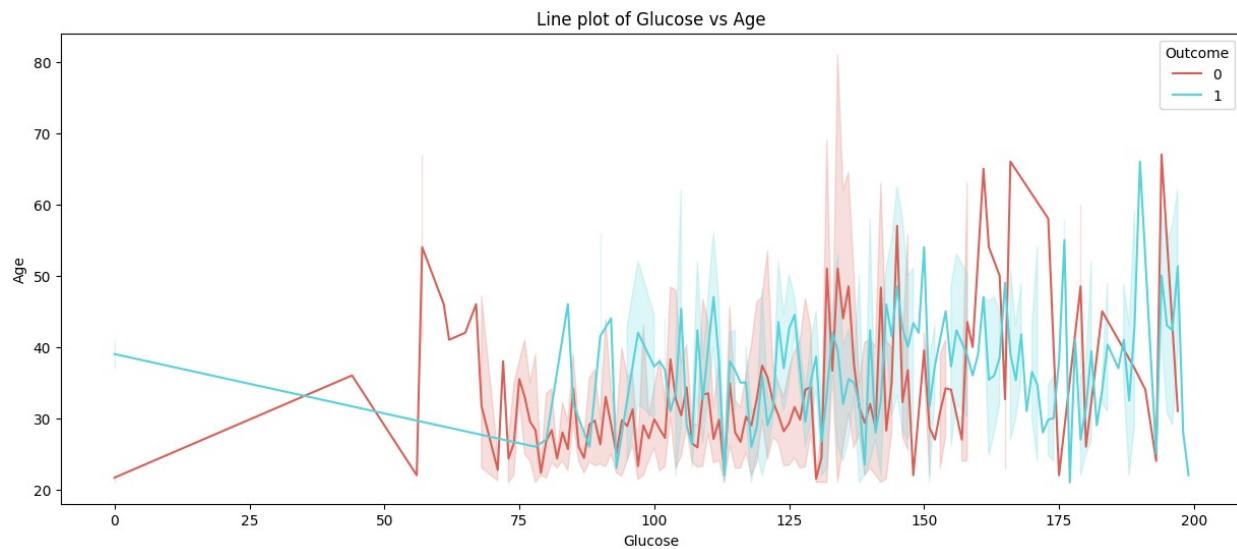


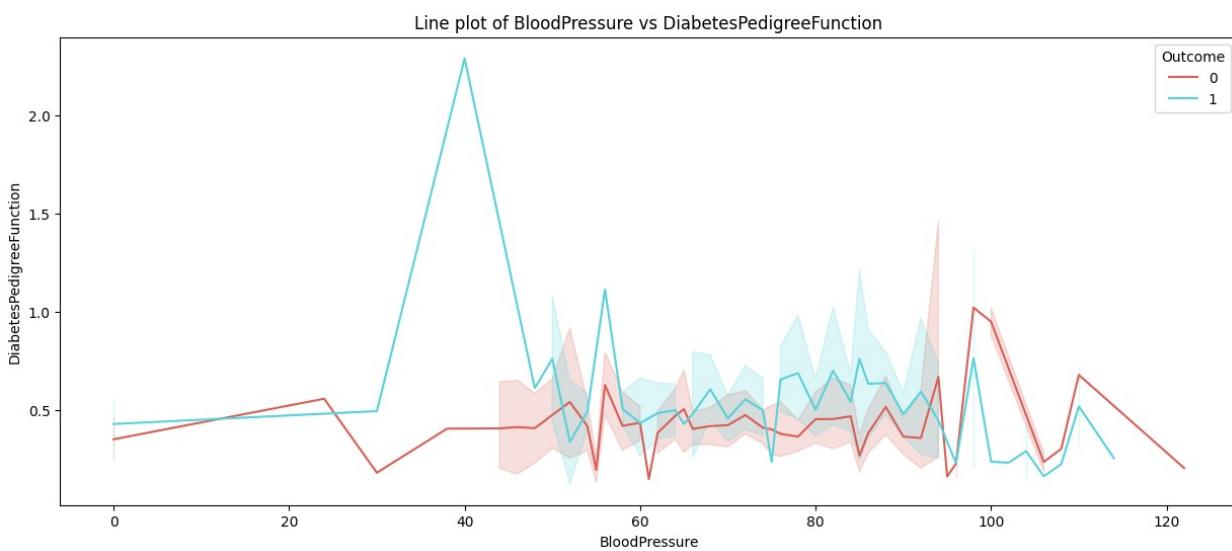
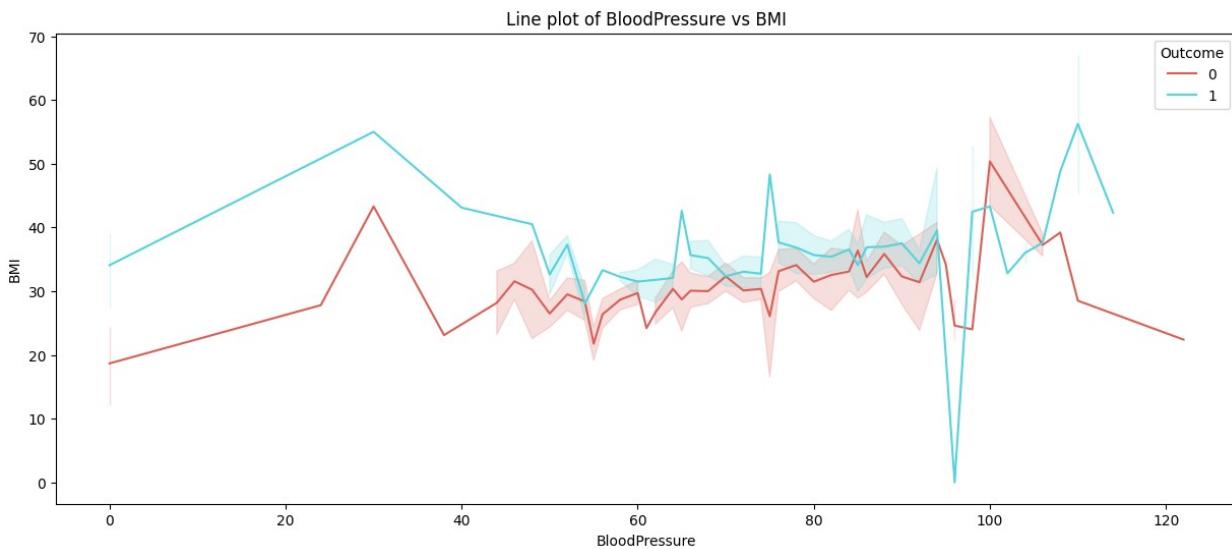
Line plot of Glucose vs BMI

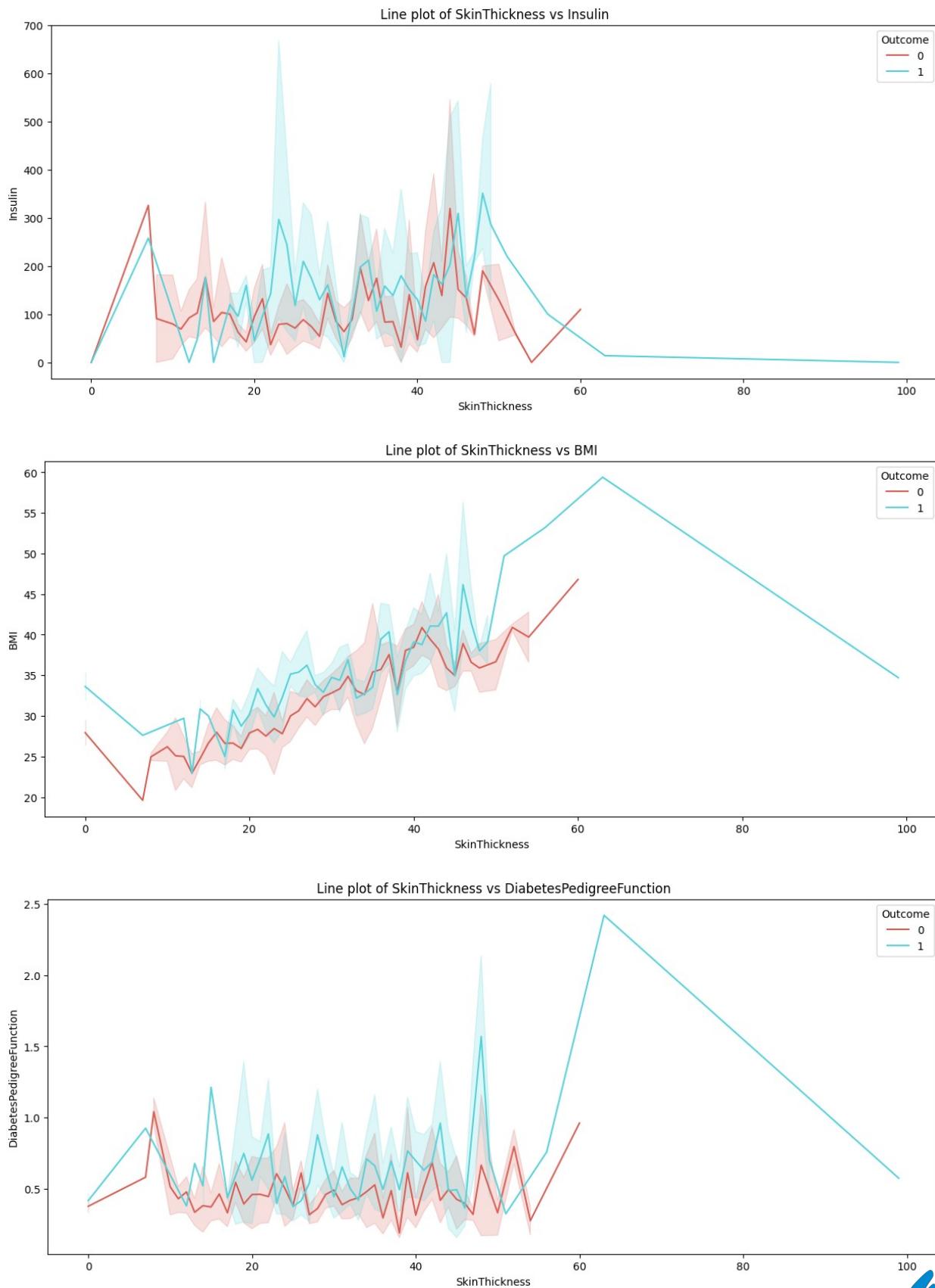


Line plot of Glucose vs DiabetesPedigreeFunction

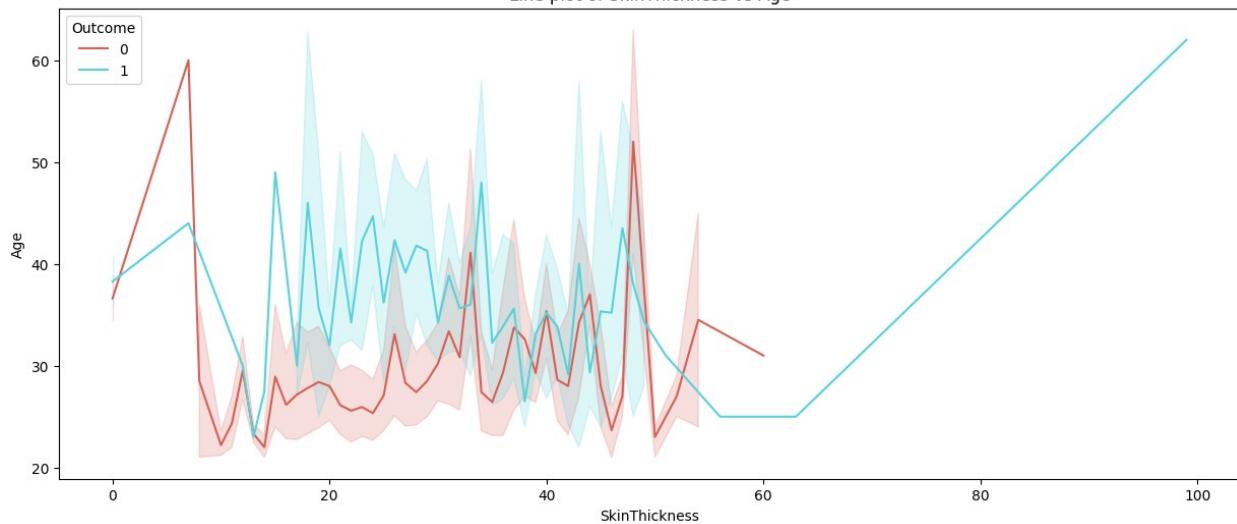




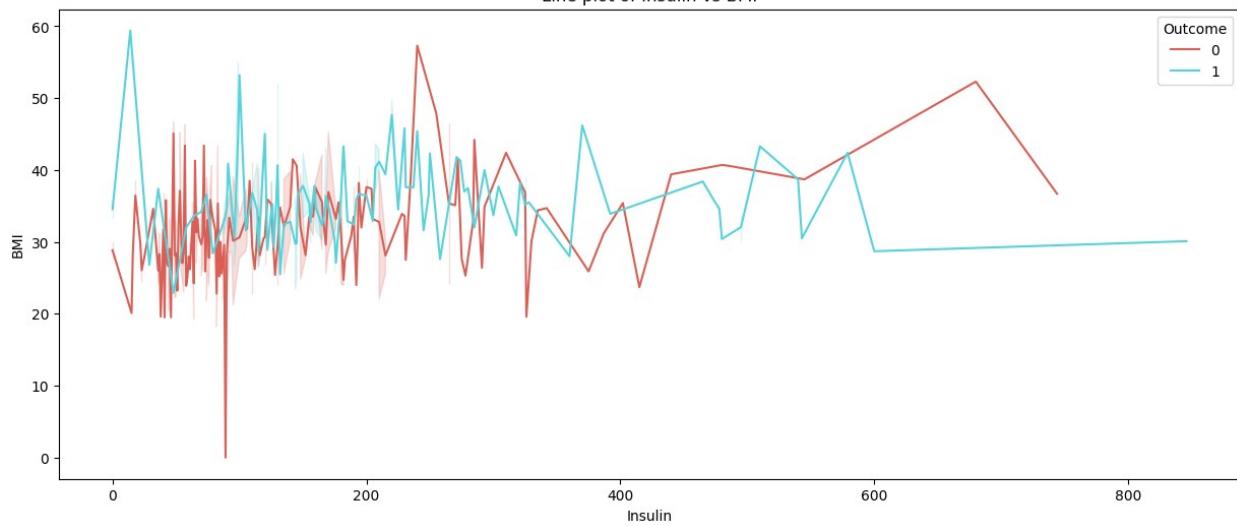




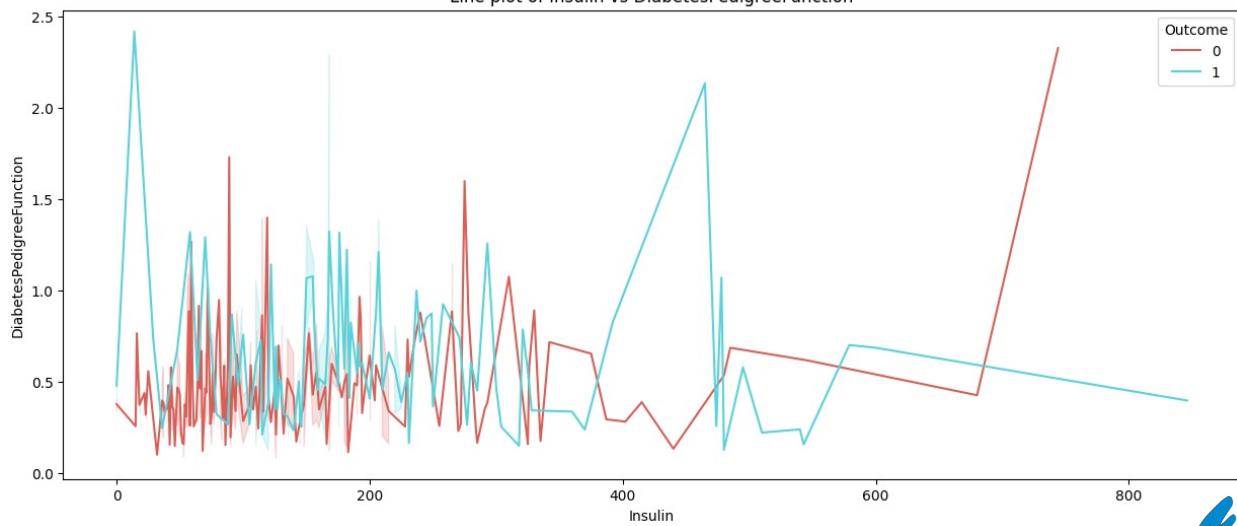
Line plot of SkinThickness vs Age



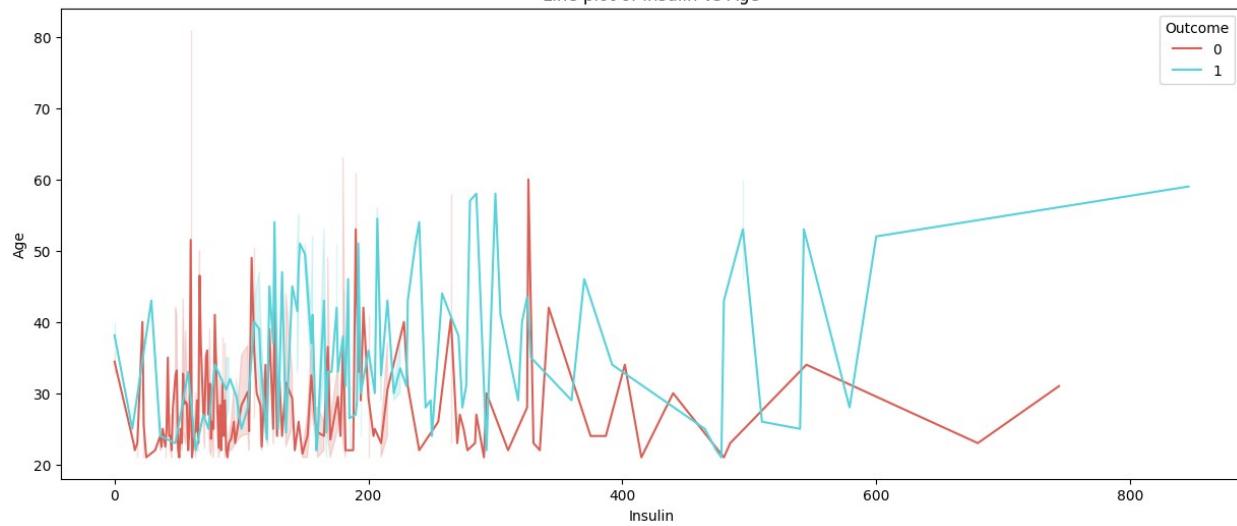
Line plot of Insulin vs BMI



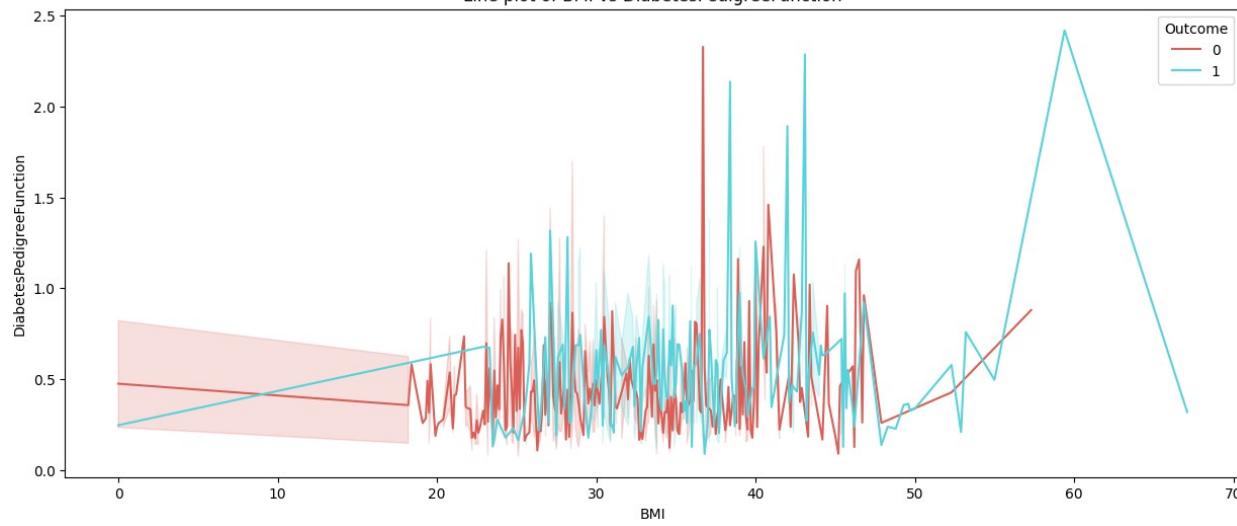
Line plot of Insulin vs DiabetesPedigreeFunction



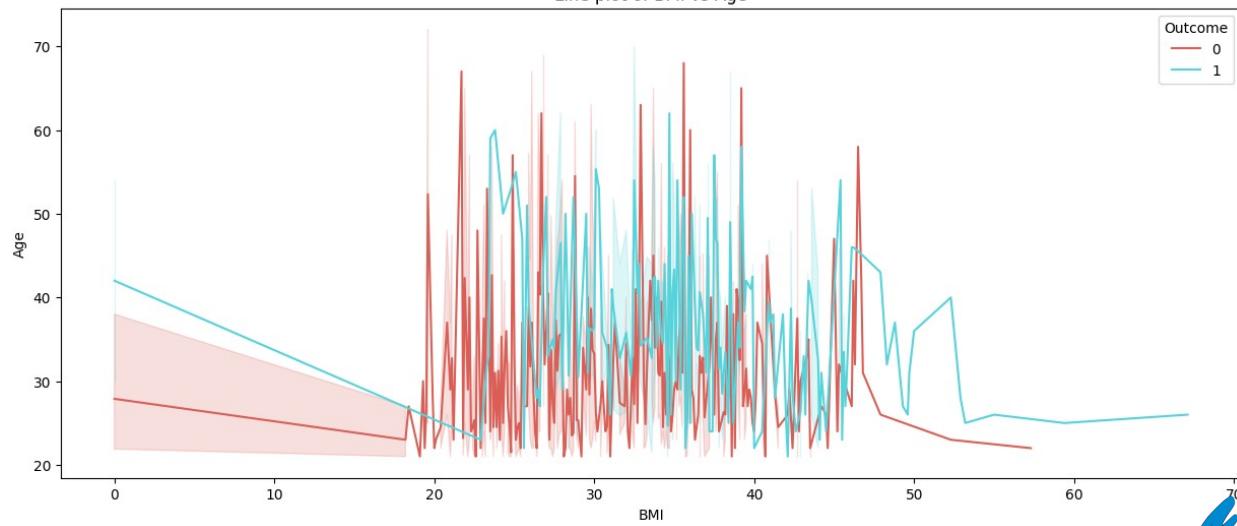
Line plot of Insulin vs Age

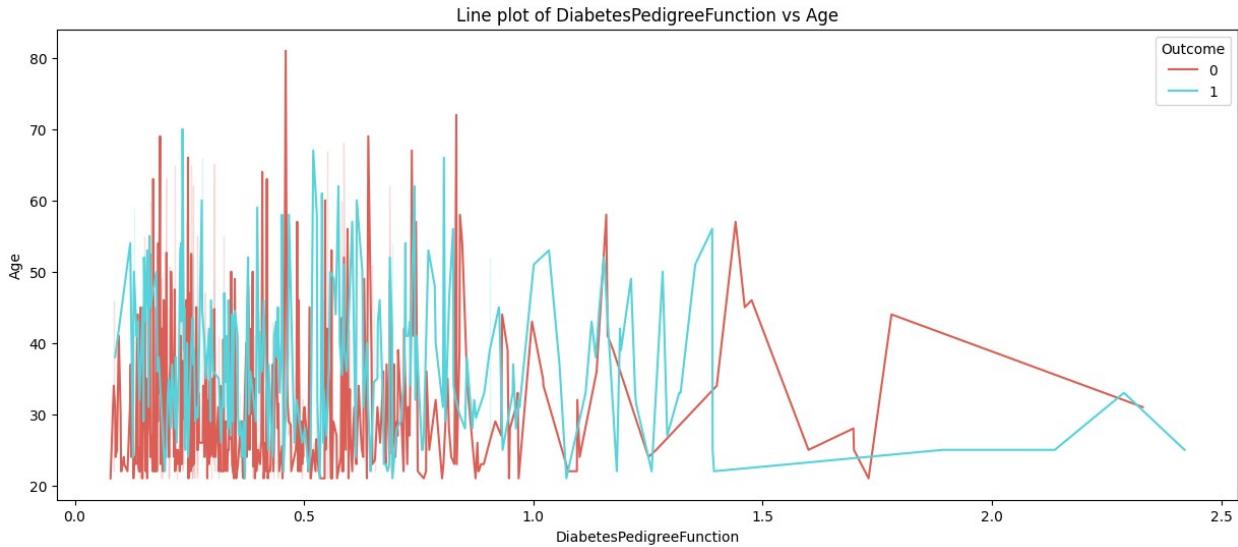


Line plot of BMI vs DiabetesPedigreeFunction



Line plot of BMI vs Age



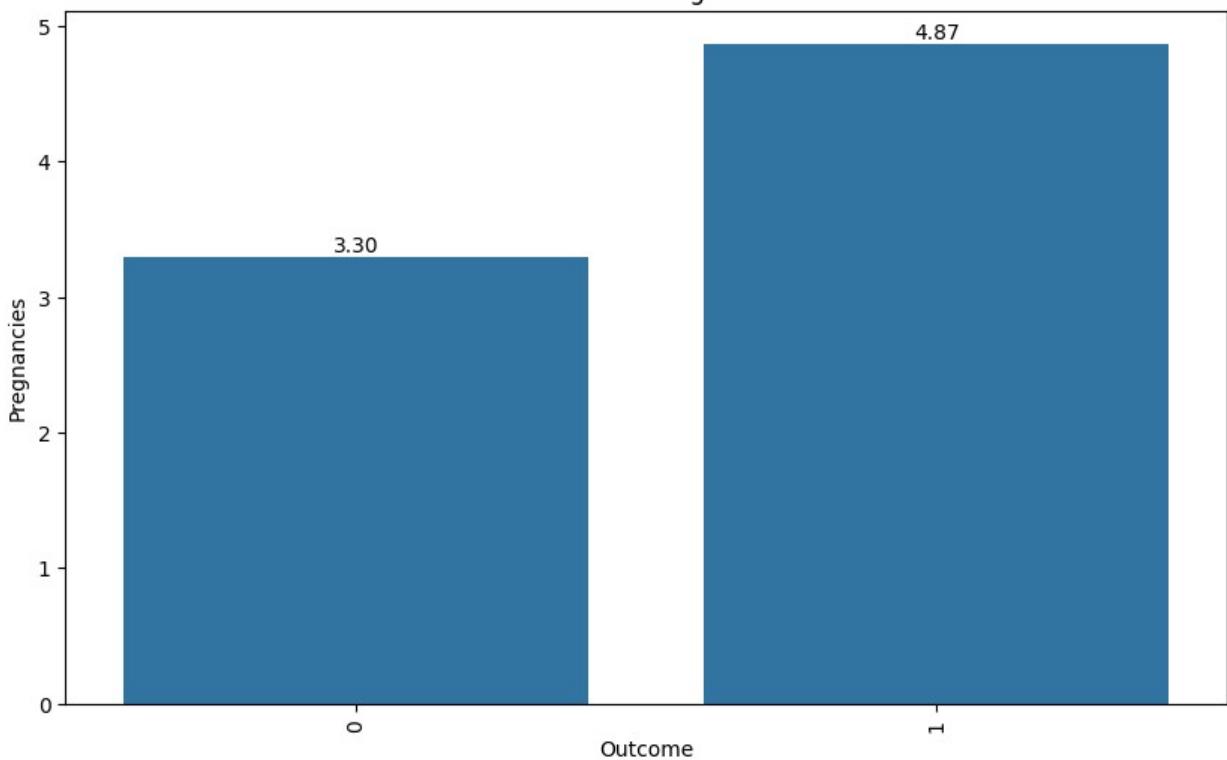


```

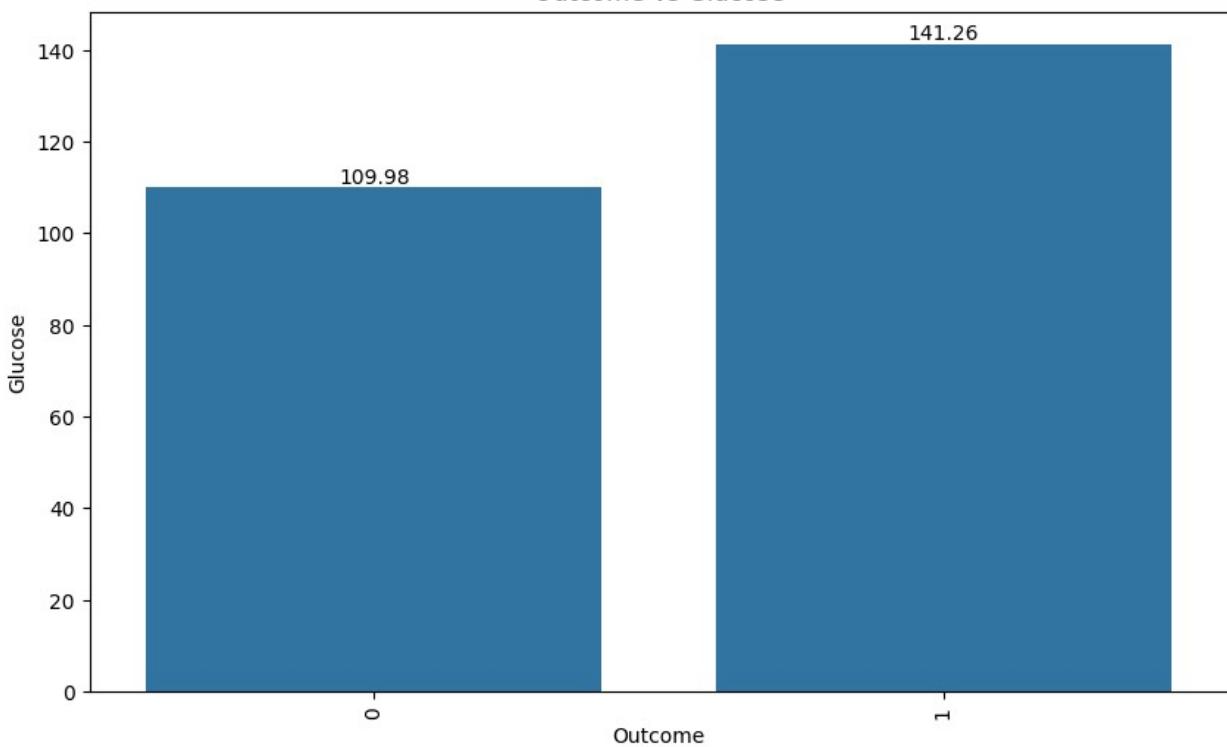
for dis in discrete:
    for cont in continuous:
        plt.figure(figsize=(10, 6))
        ax = sns.barplot(data=df, x=dis, y=cont, ci=None)
        plt.title(f'{dis} vs {cont}')

        for p in ax.patches:
            height = p.get_height()
            ax.annotate(f'{height:.2f}', (p.get_x() + p.get_width() / 2., height),
                        ha='center', va='bottom', fontsize=10,
            color='black', rotation=0)
        plt.xticks(rotation = 90)
        plt.show()
    
```

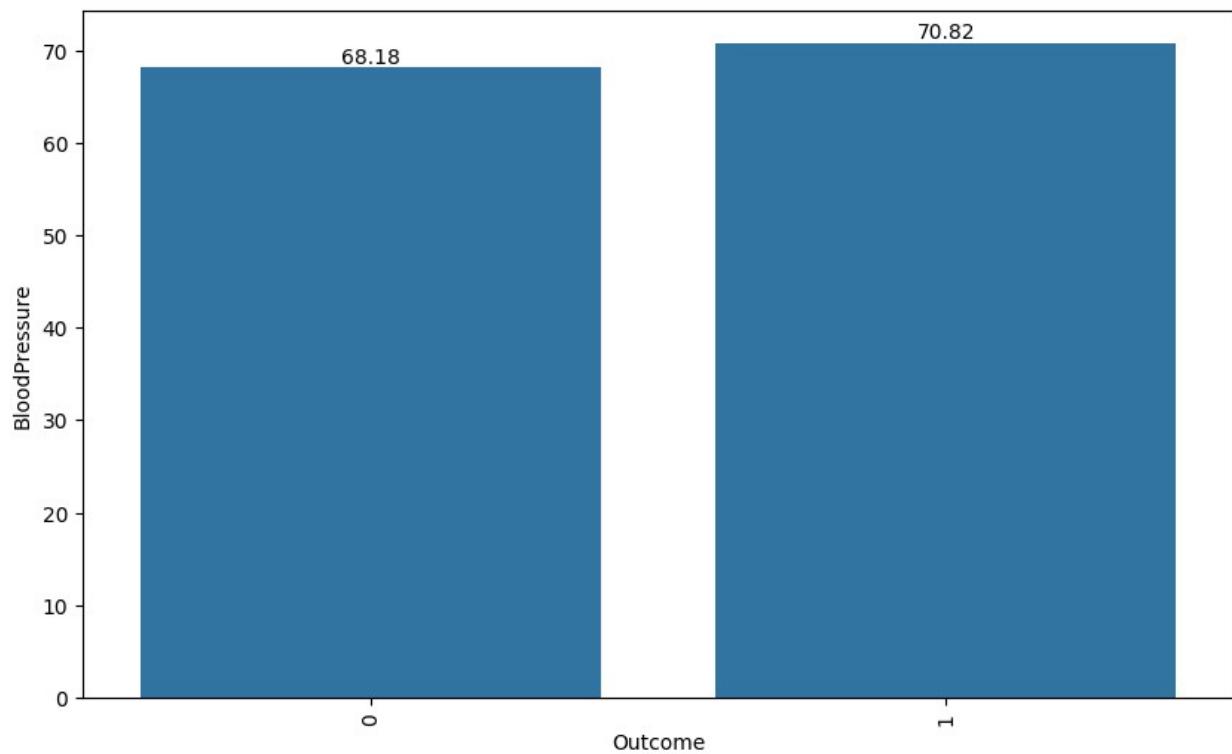
Outcome vs Pregnancies



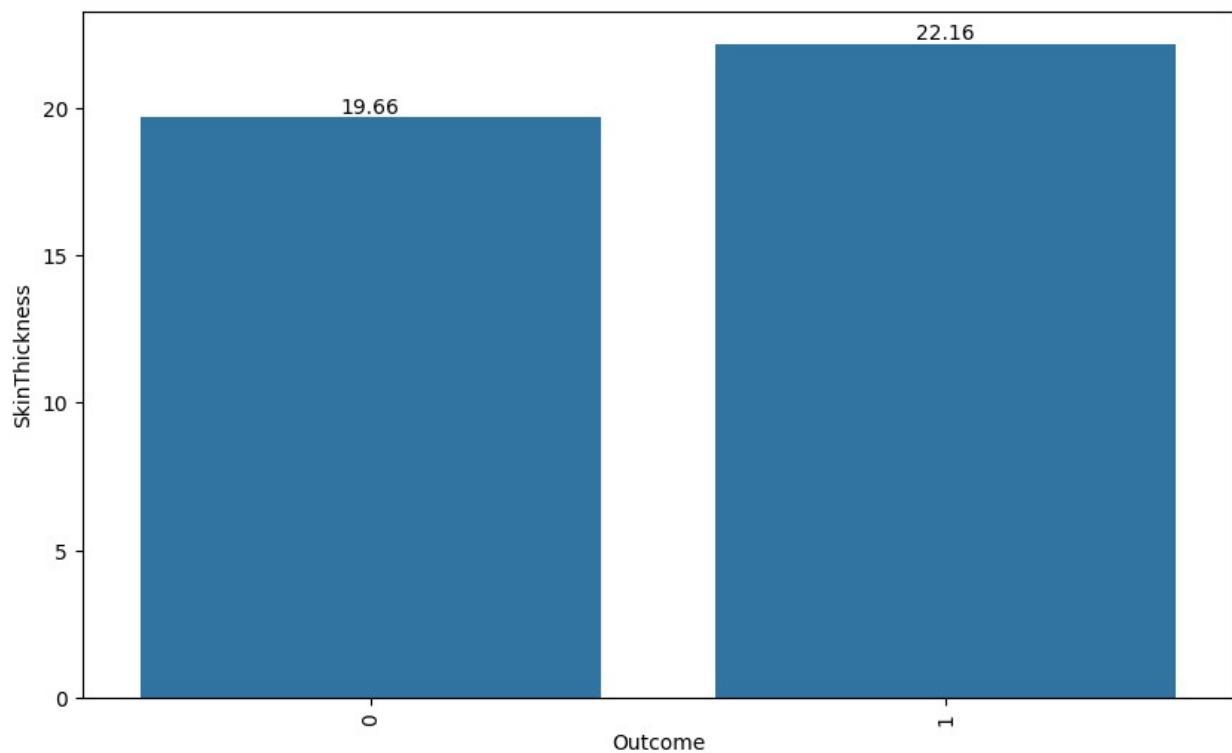
Outcome vs Glucose



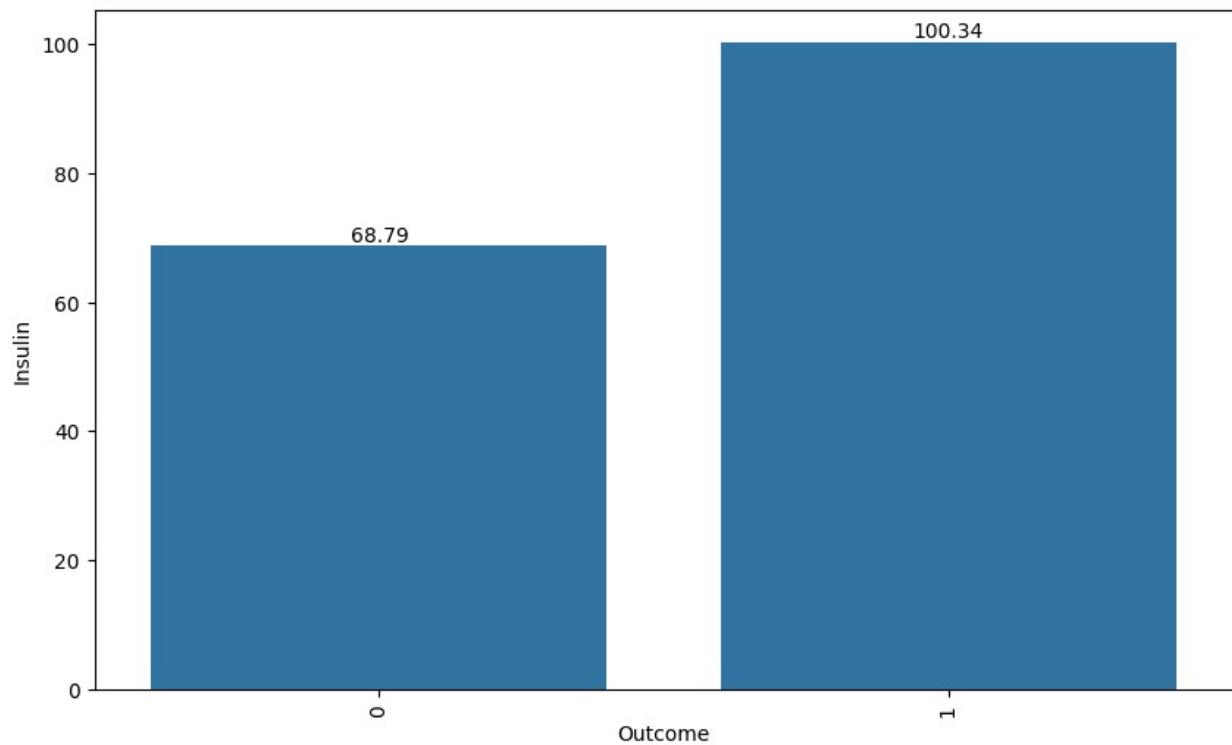
Outcome vs BloodPressure



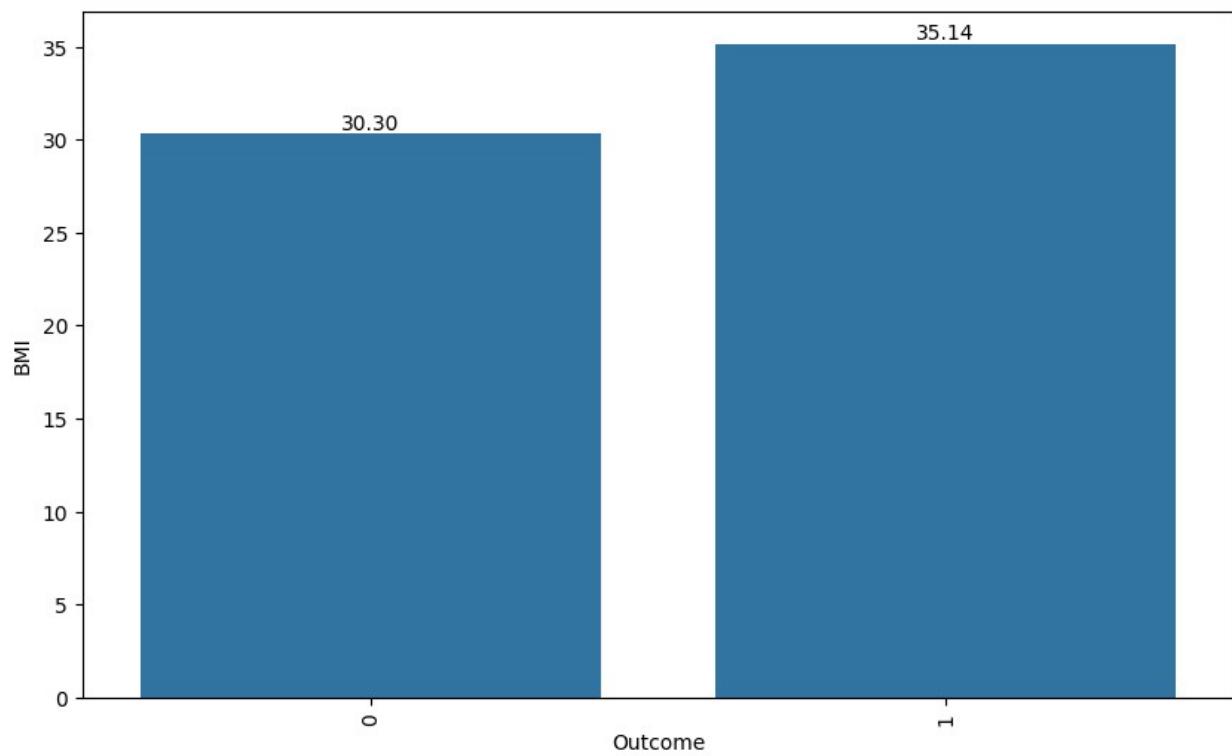
Outcome vs SkinThickness



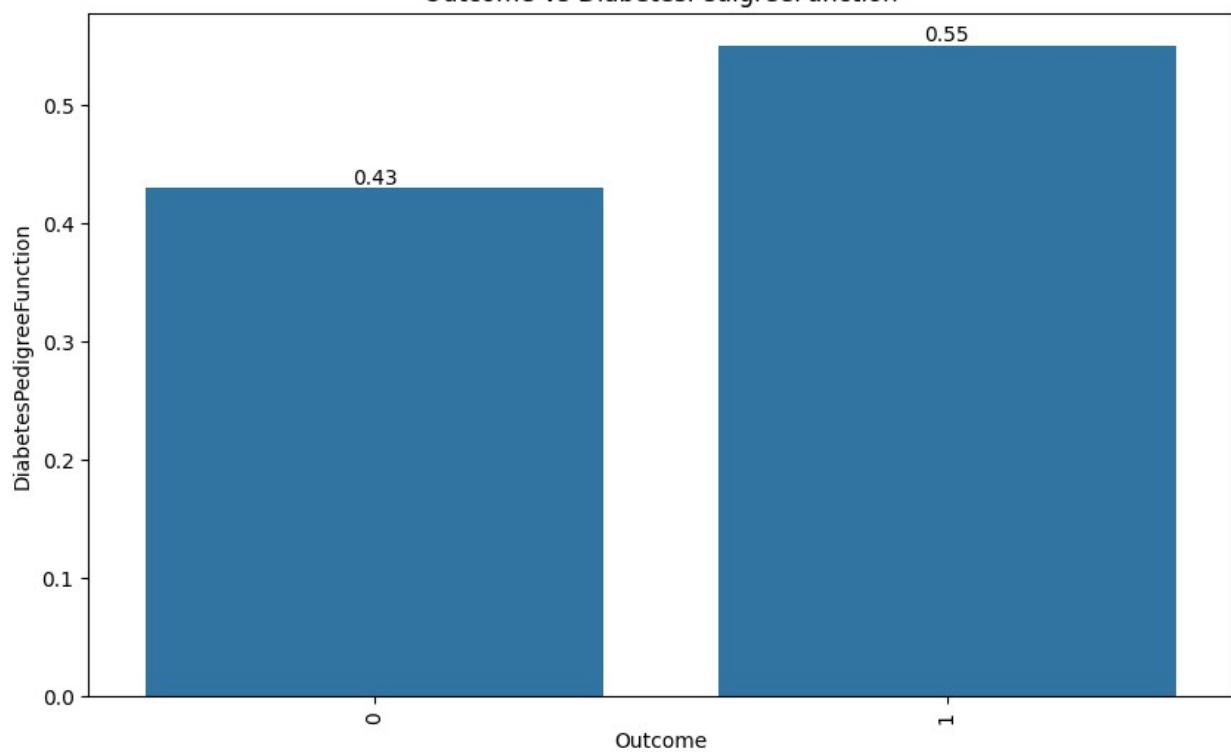
Outcome vs Insulin



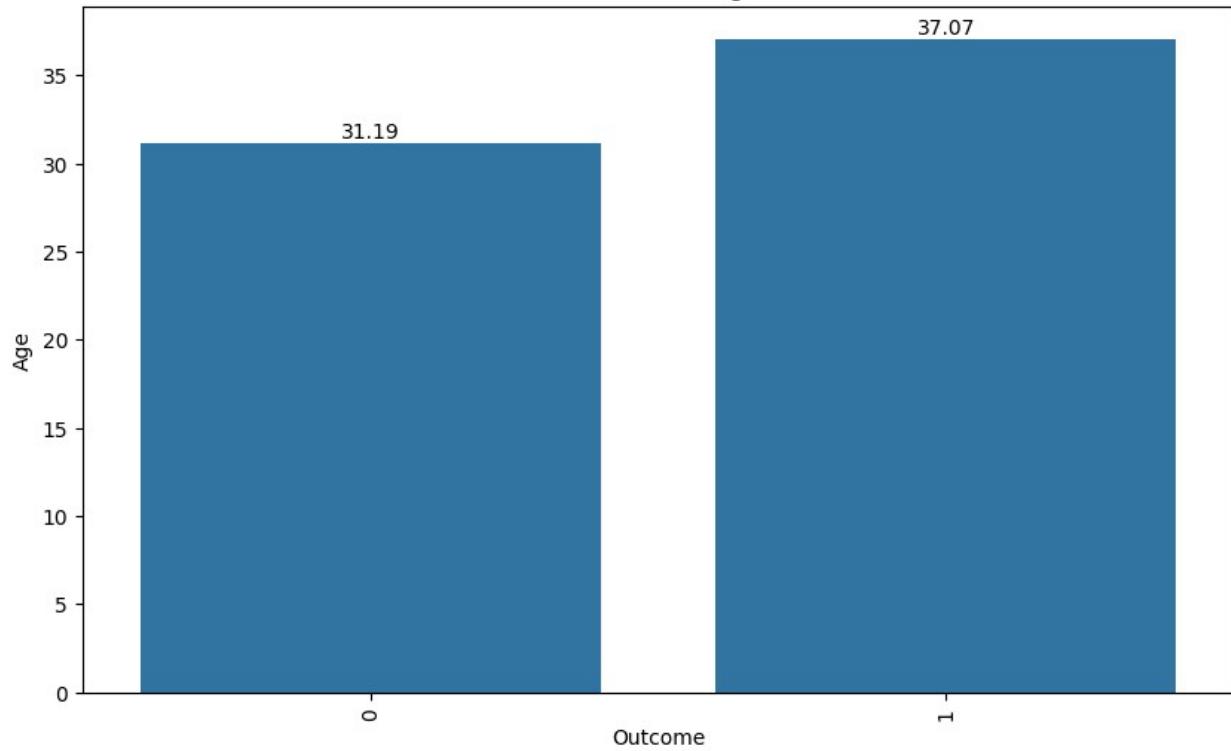
Outcome vs BMI



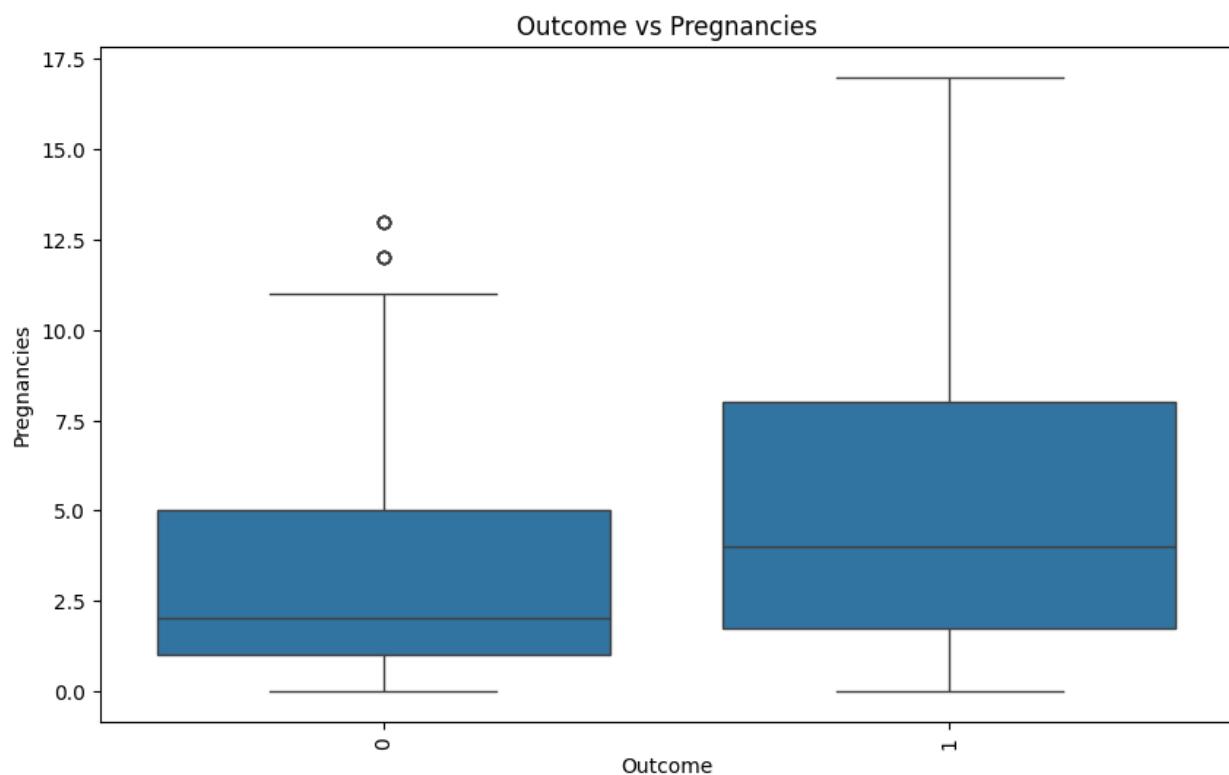
Outcome vs DiabetesPedigreeFunction



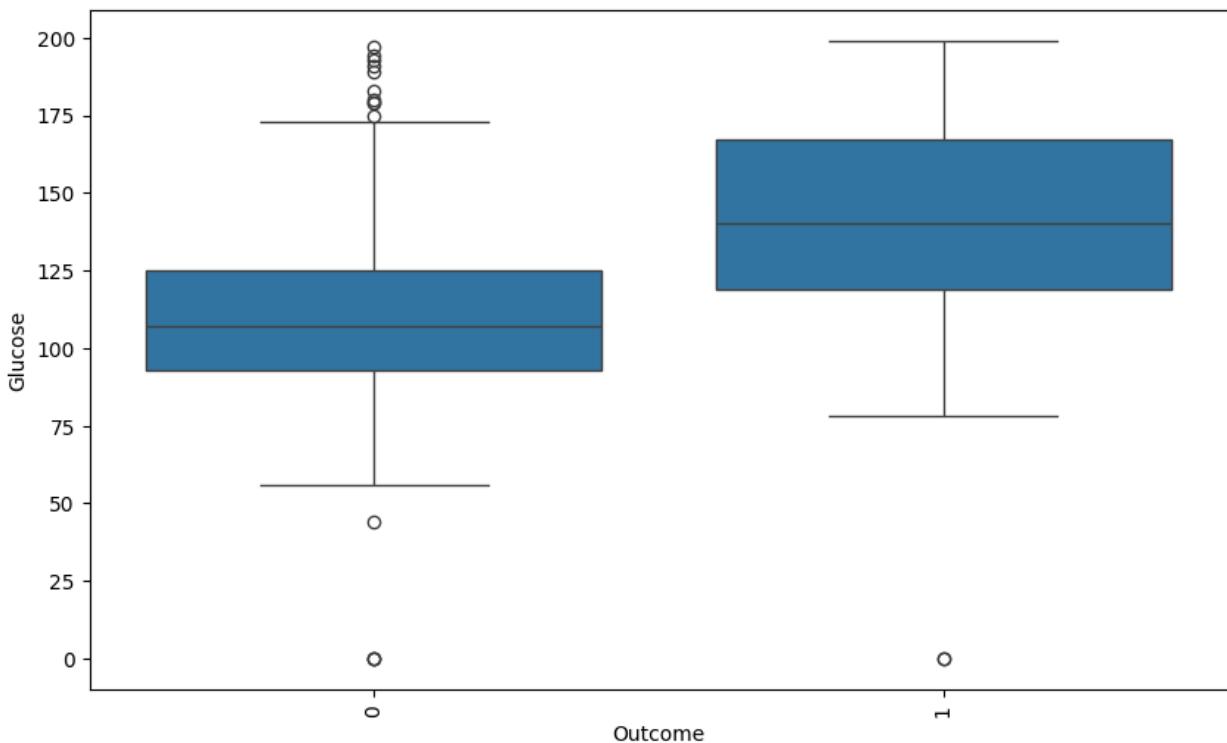
Outcome vs Age



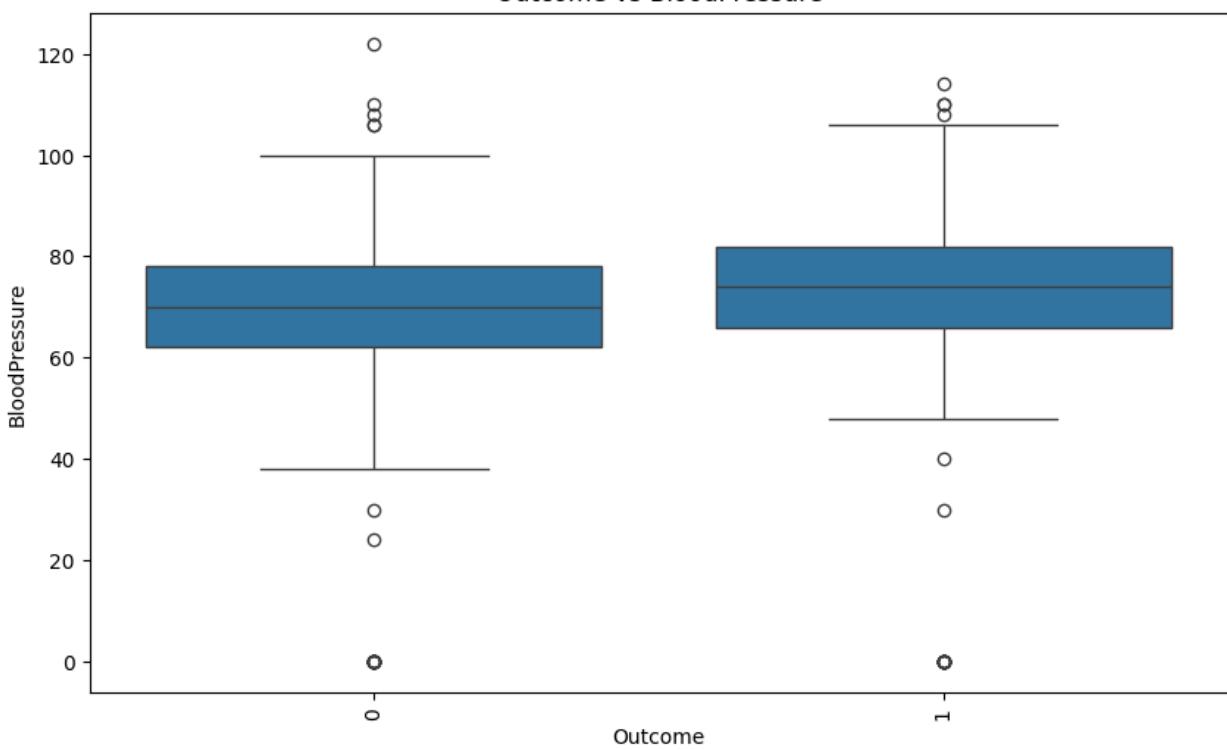
```
for dis in discrete:
    for cont in continuous:
        plt.figure(figsize=(10, 6))
        ax = sns.boxplot(data=df, x=dis, y=cont)
        plt.title(f'{dis} vs {cont}')
        plt.xticks(rotation = 90)
        plt.show()
```



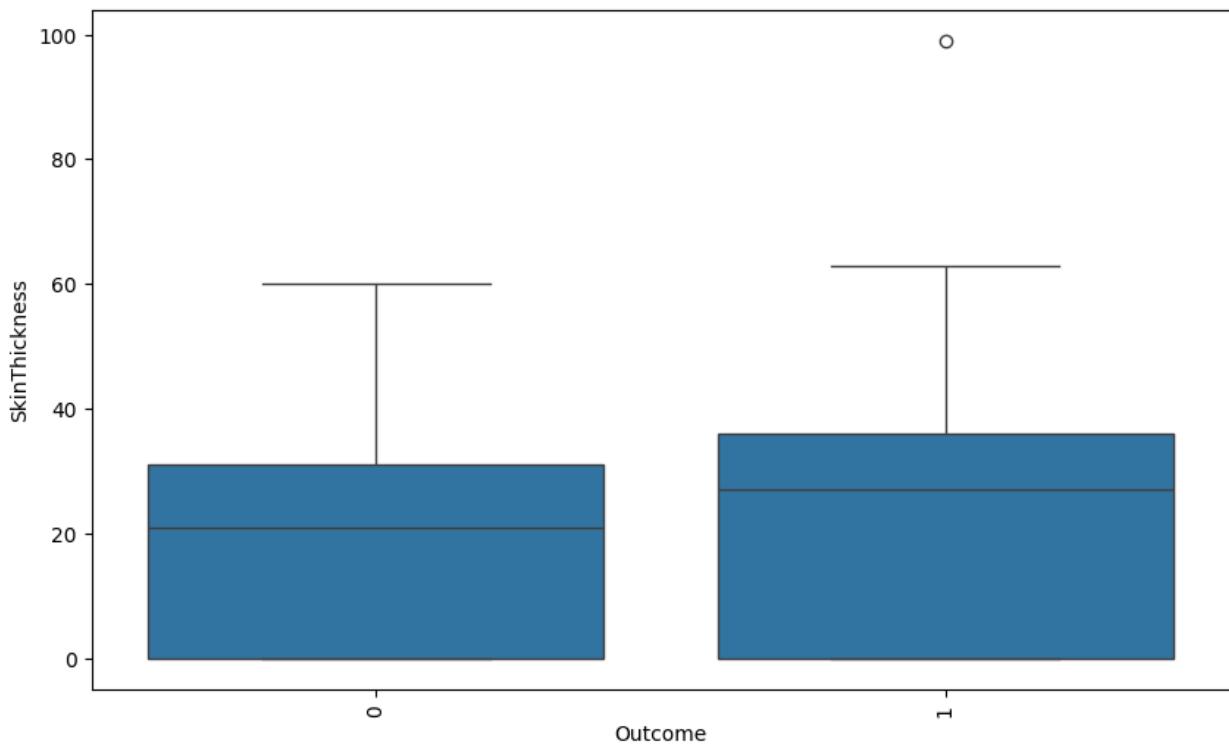
Outcome vs Glucose



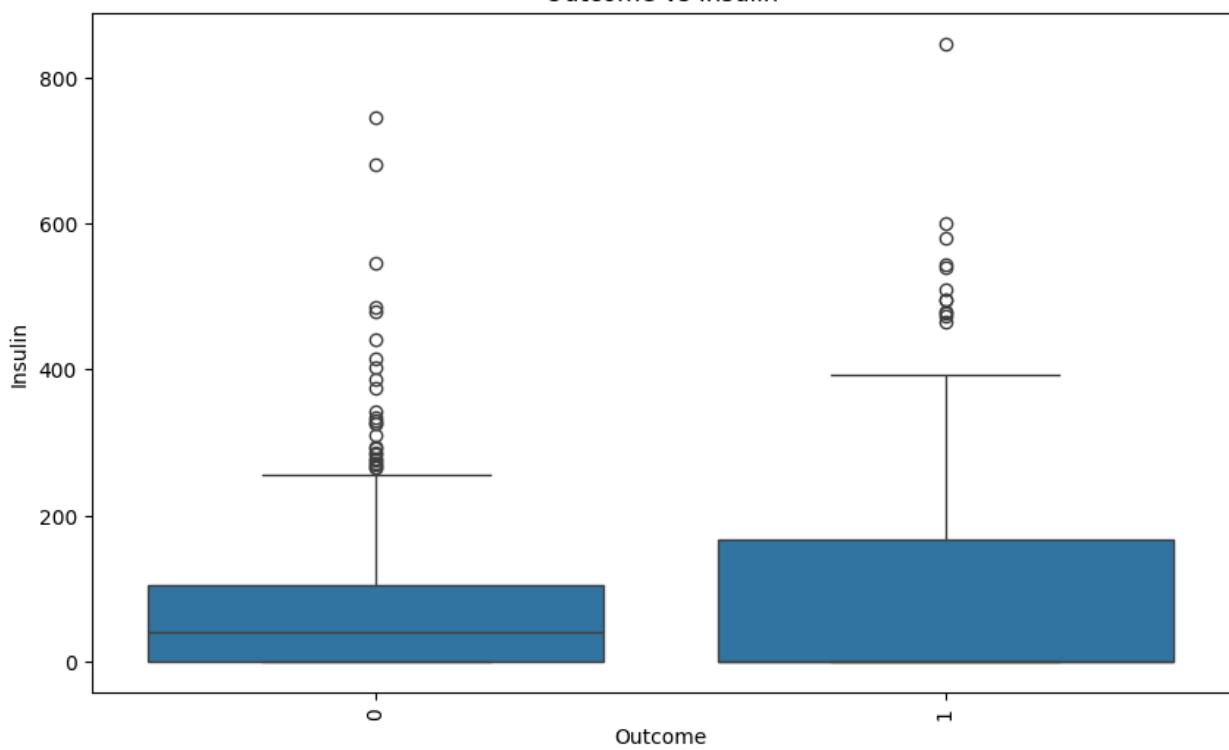
Outcome vs BloodPressure

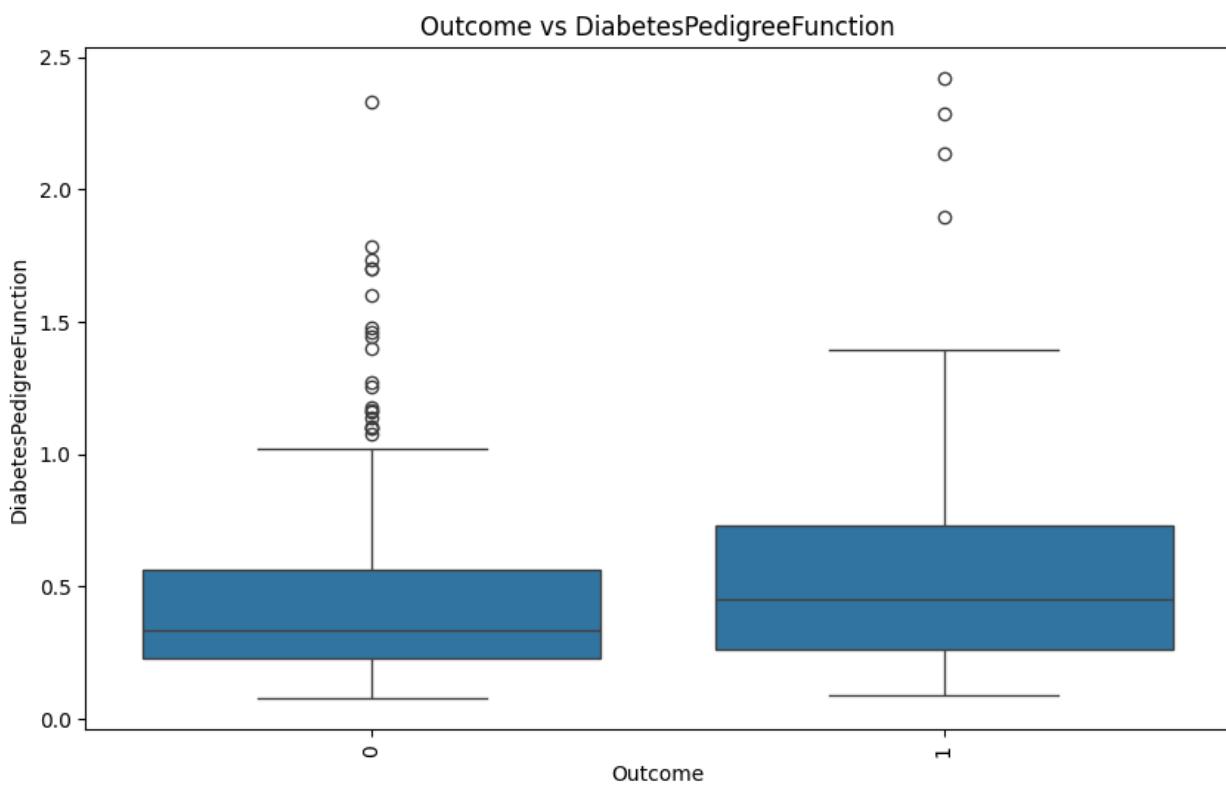
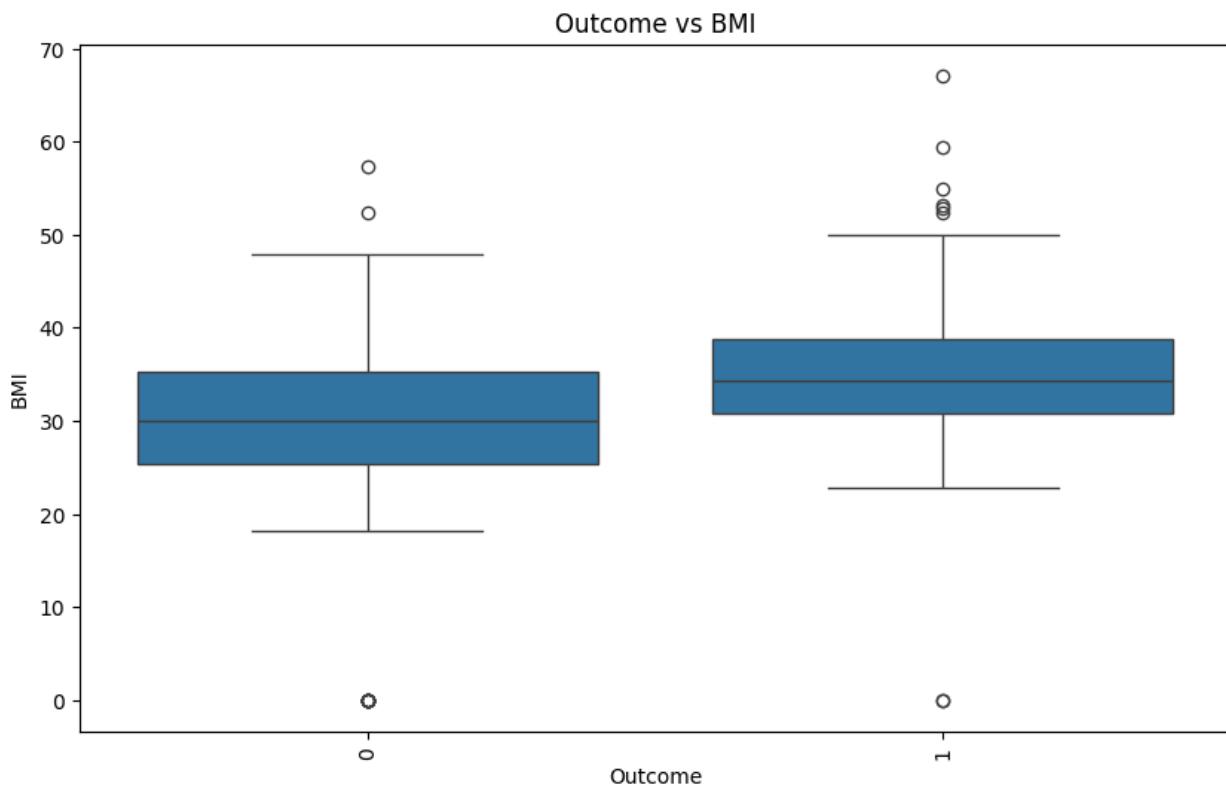


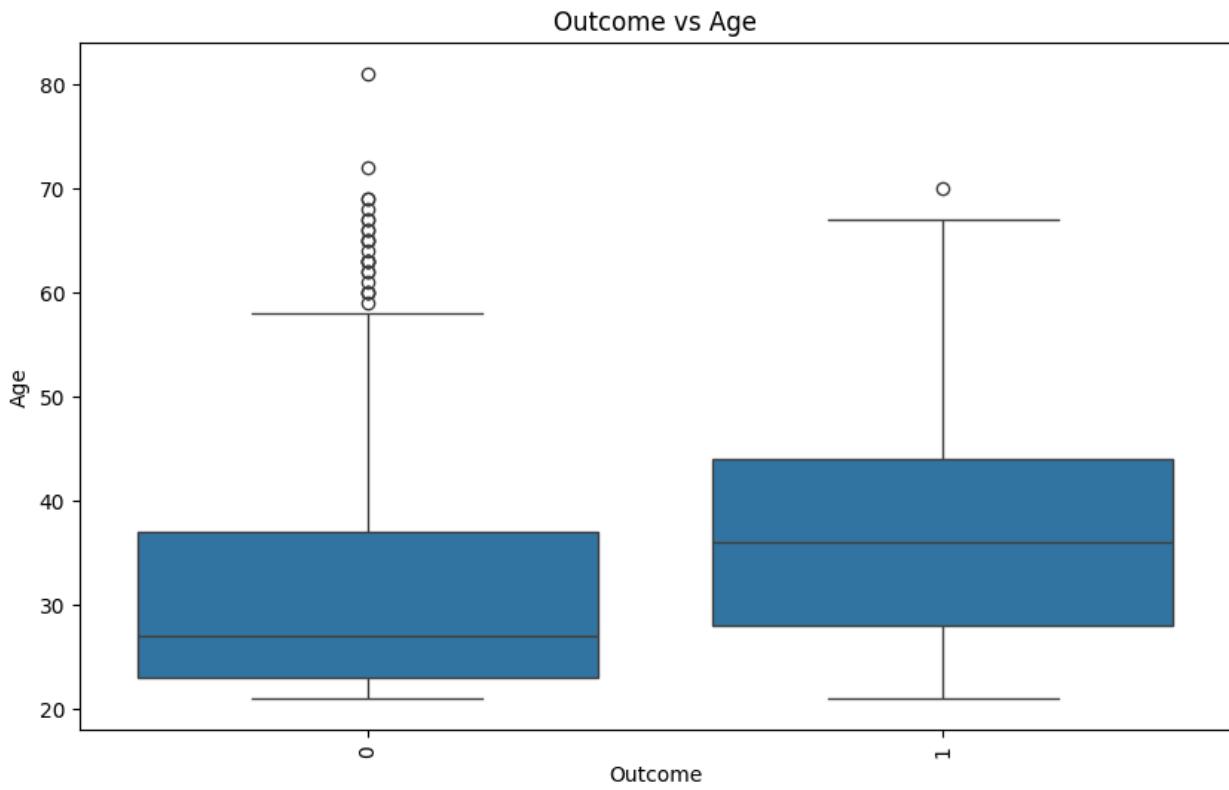
Outcome vs SkinThickness



Outcome vs Insulin

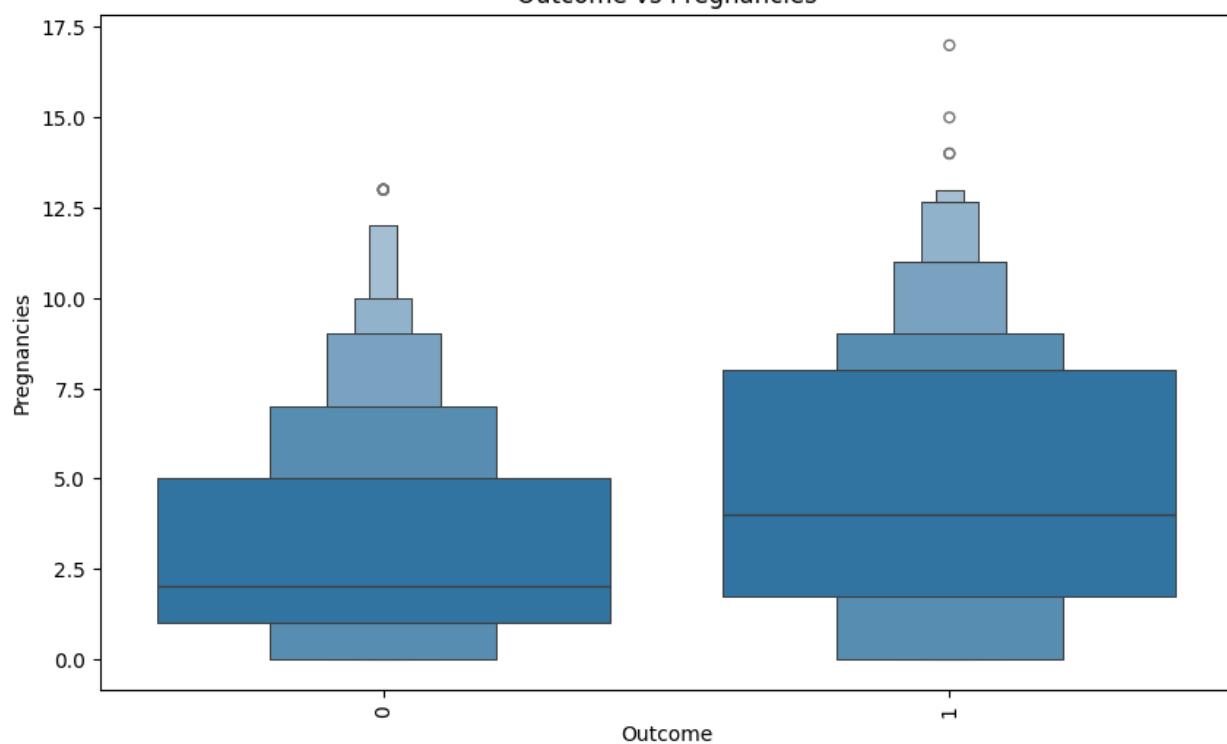




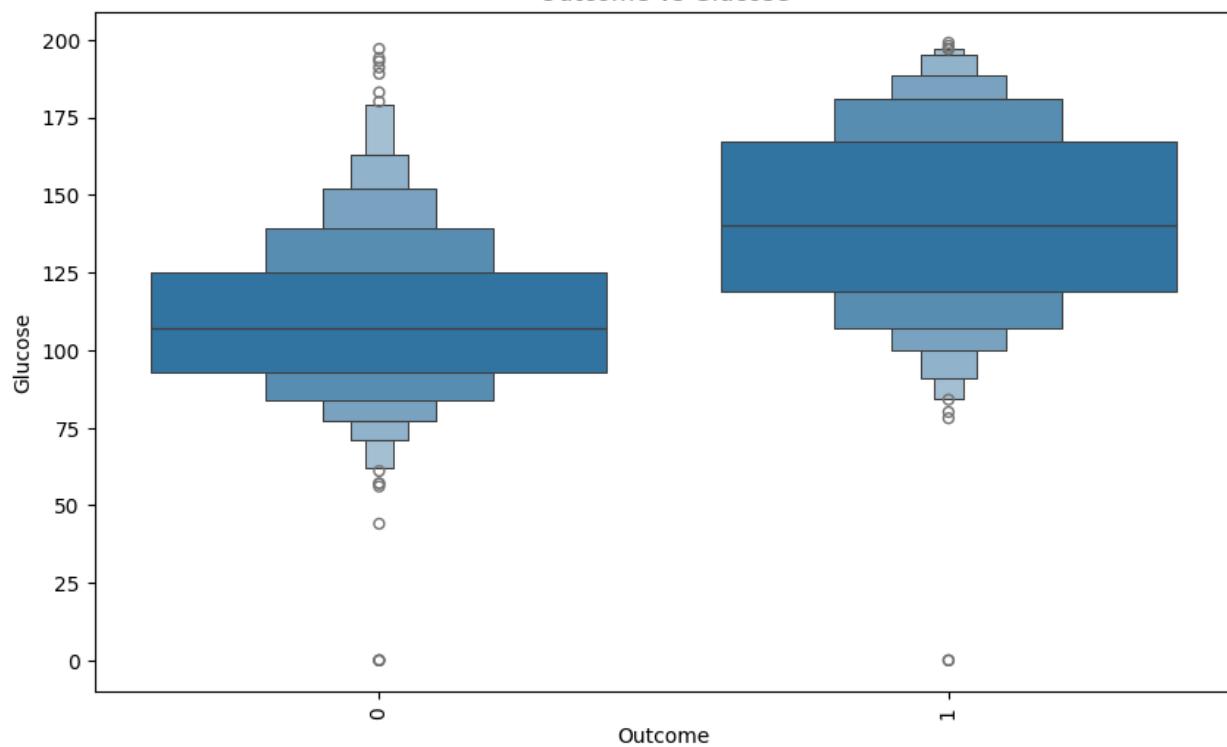


```
for dis in discrete:
    for cont in continuous:
        plt.figure(figsize=(10, 6))
        ax = sns.boxenplot(data=df, x=dis, y=cont)
        plt.title(f'{dis} vs {cont}')
        plt.xticks(rotation = 90)
        plt.show()
```

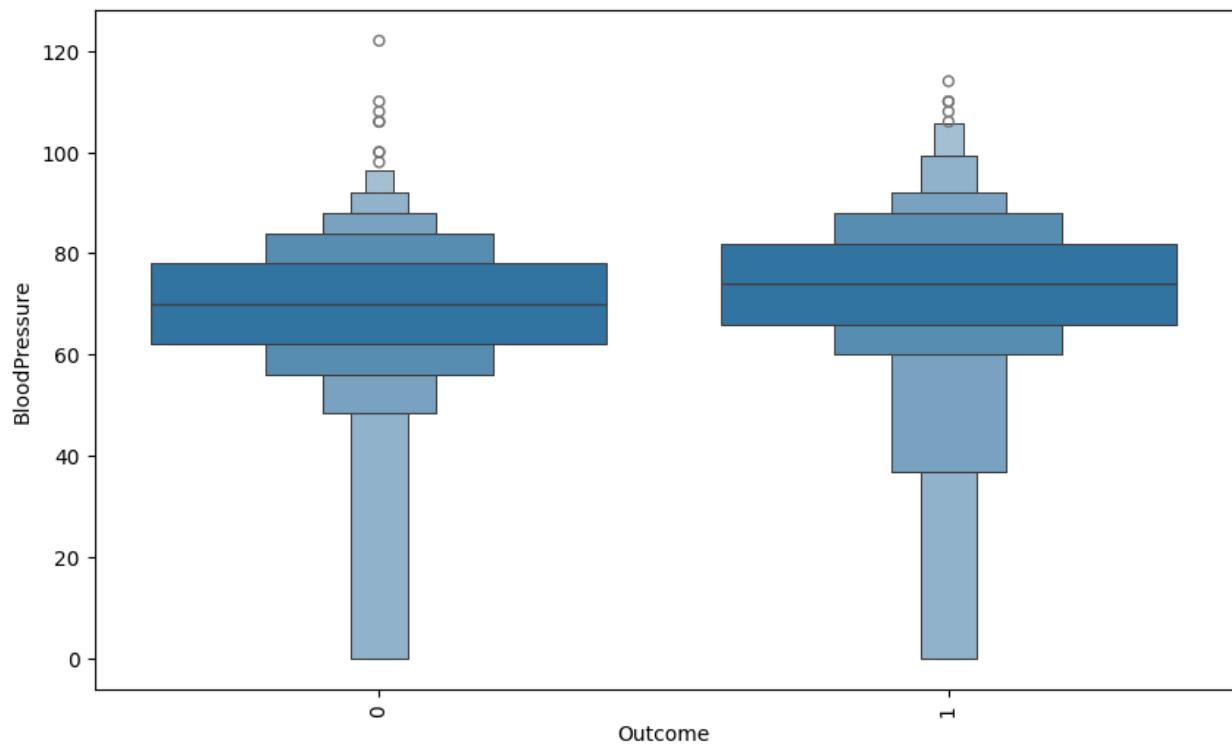
Outcome vs Pregnancies



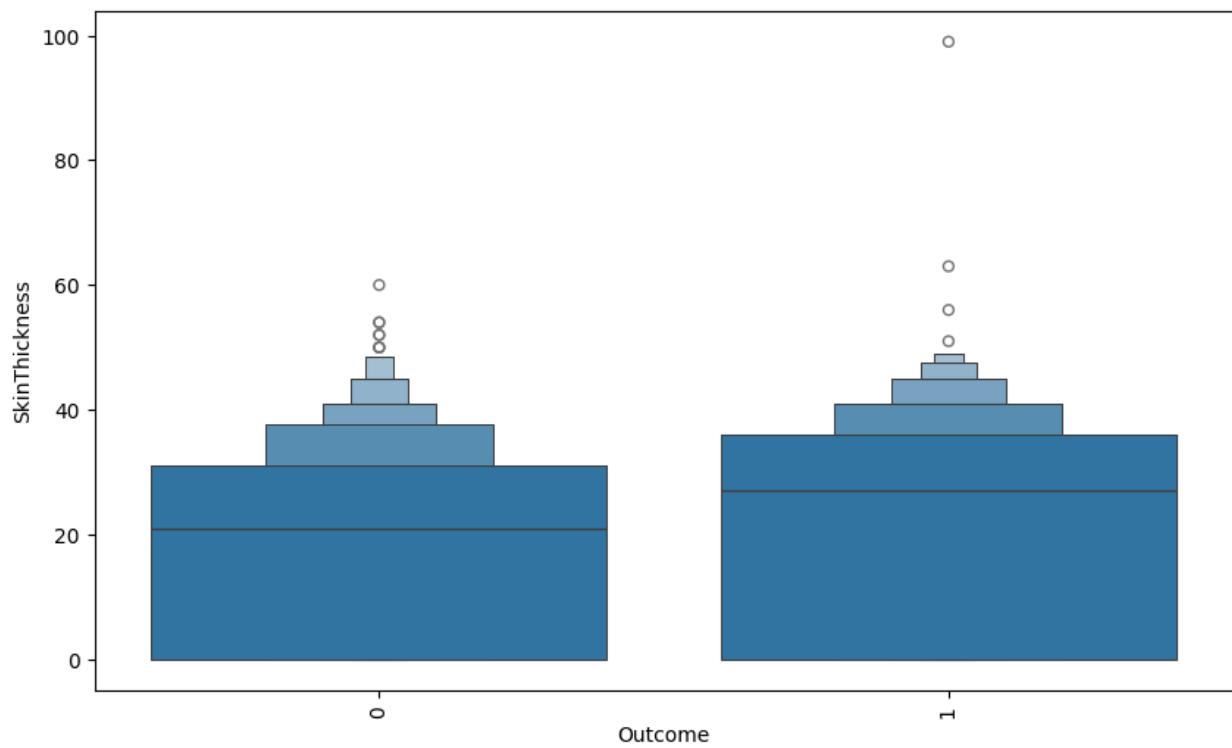
Outcome vs Glucose



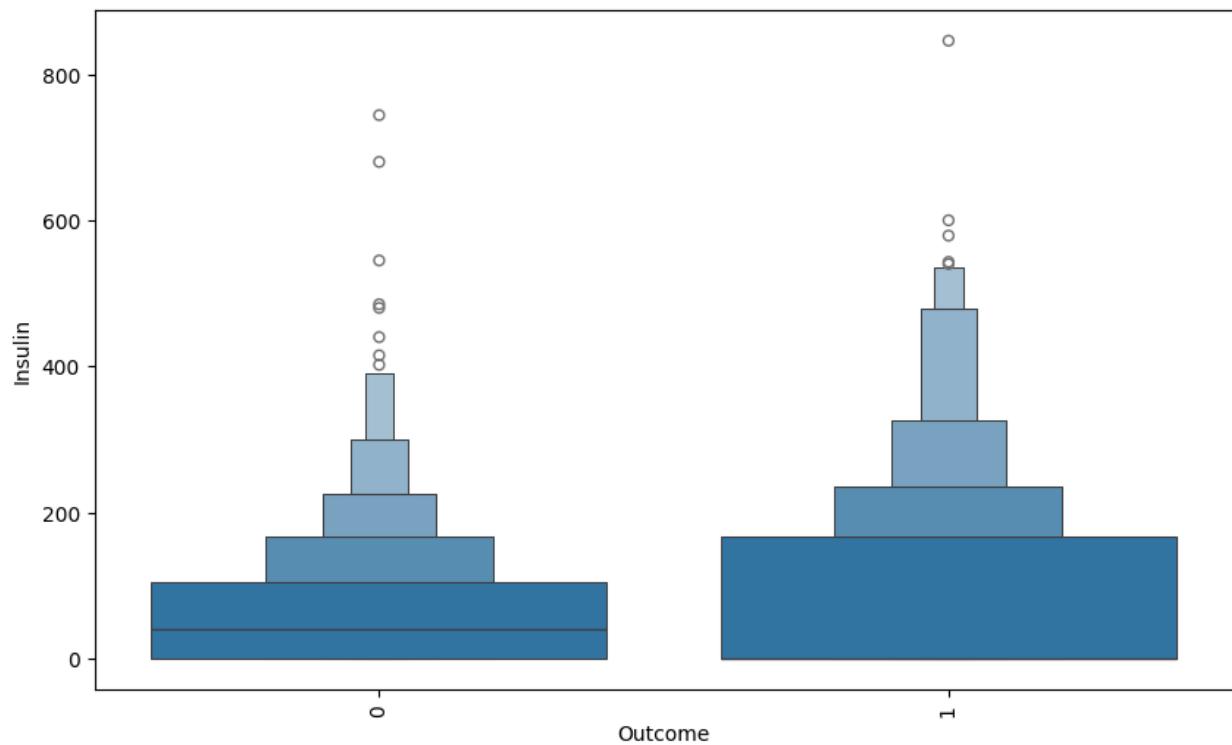
Outcome vs BloodPressure



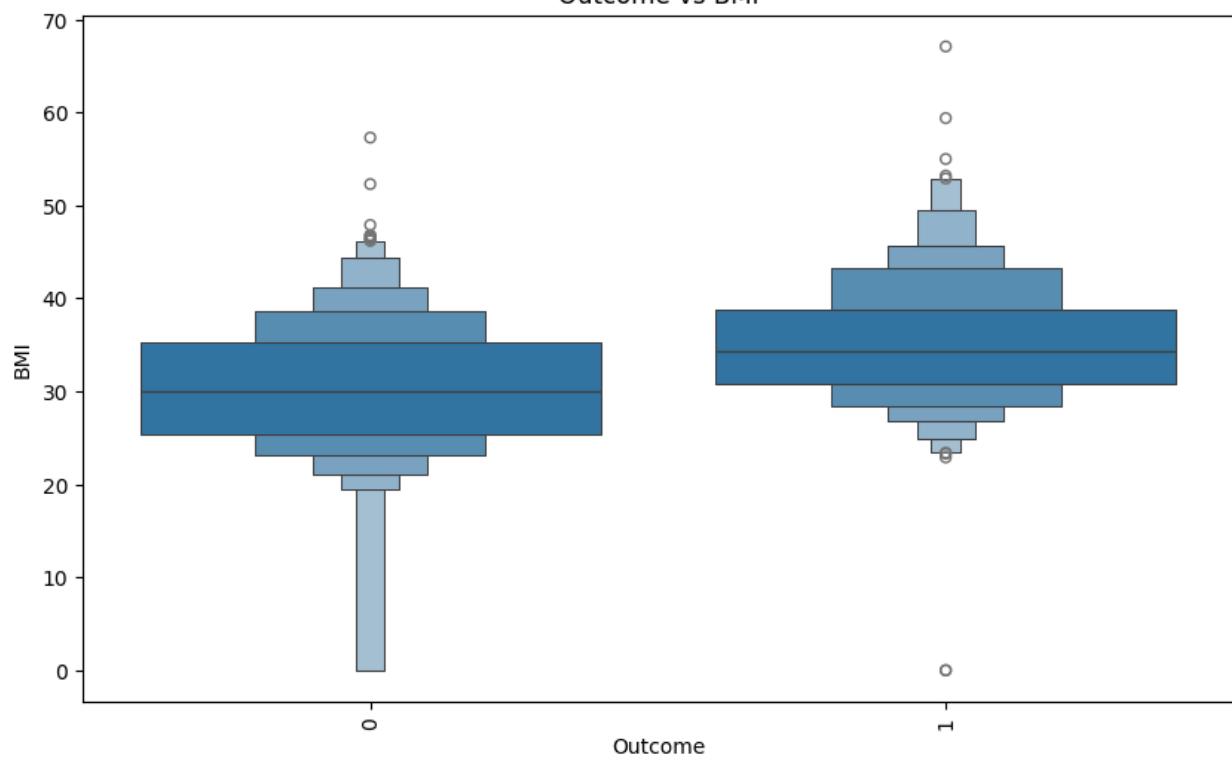
Outcome vs SkinThickness



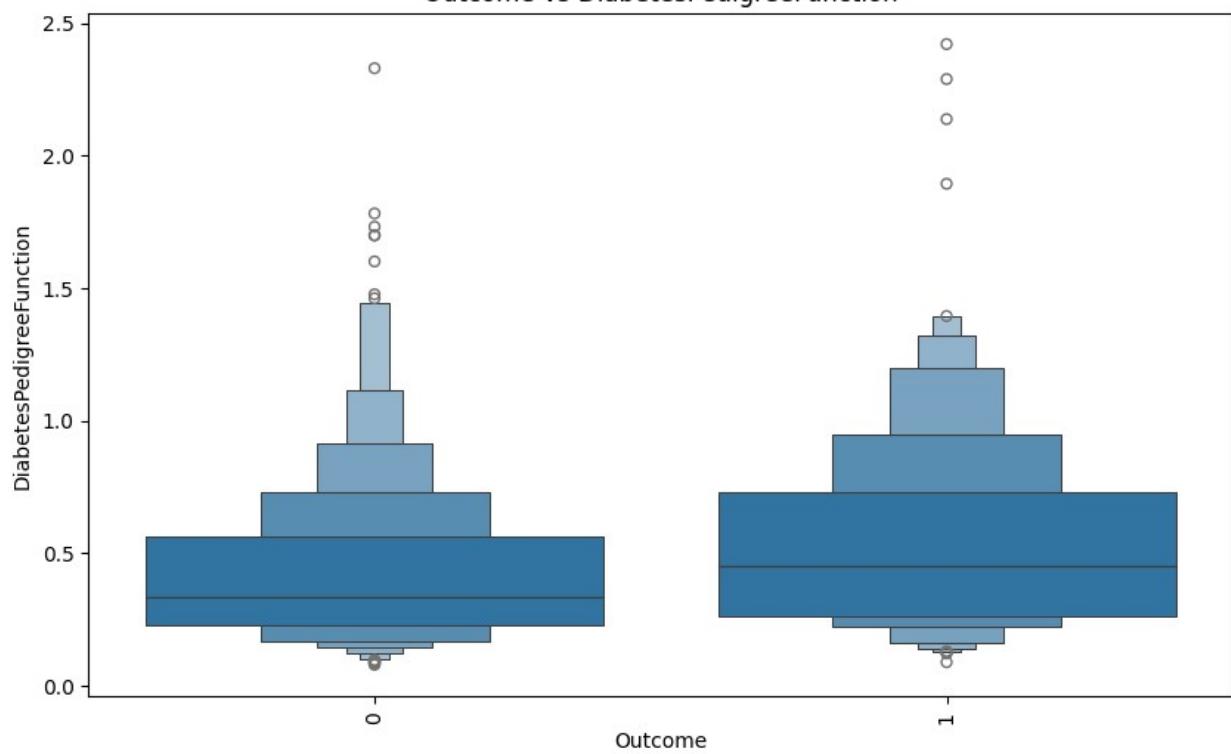
Outcome vs Insulin



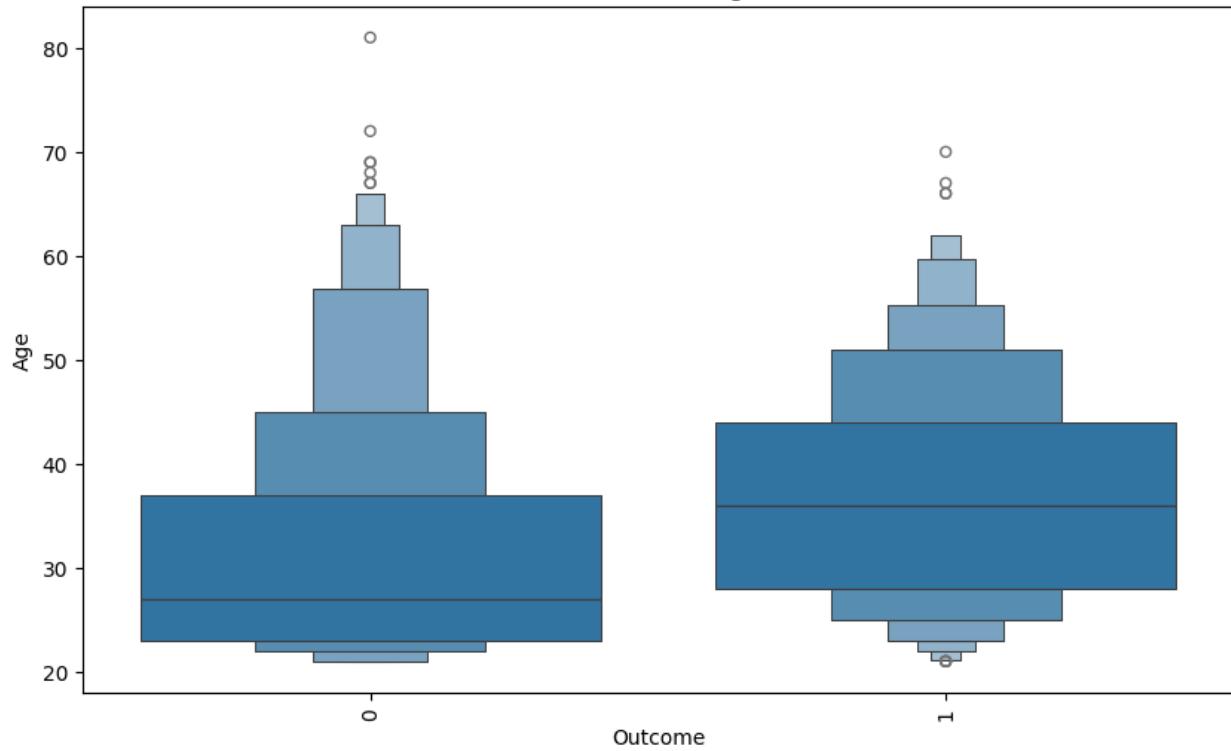
Outcome vs BMI



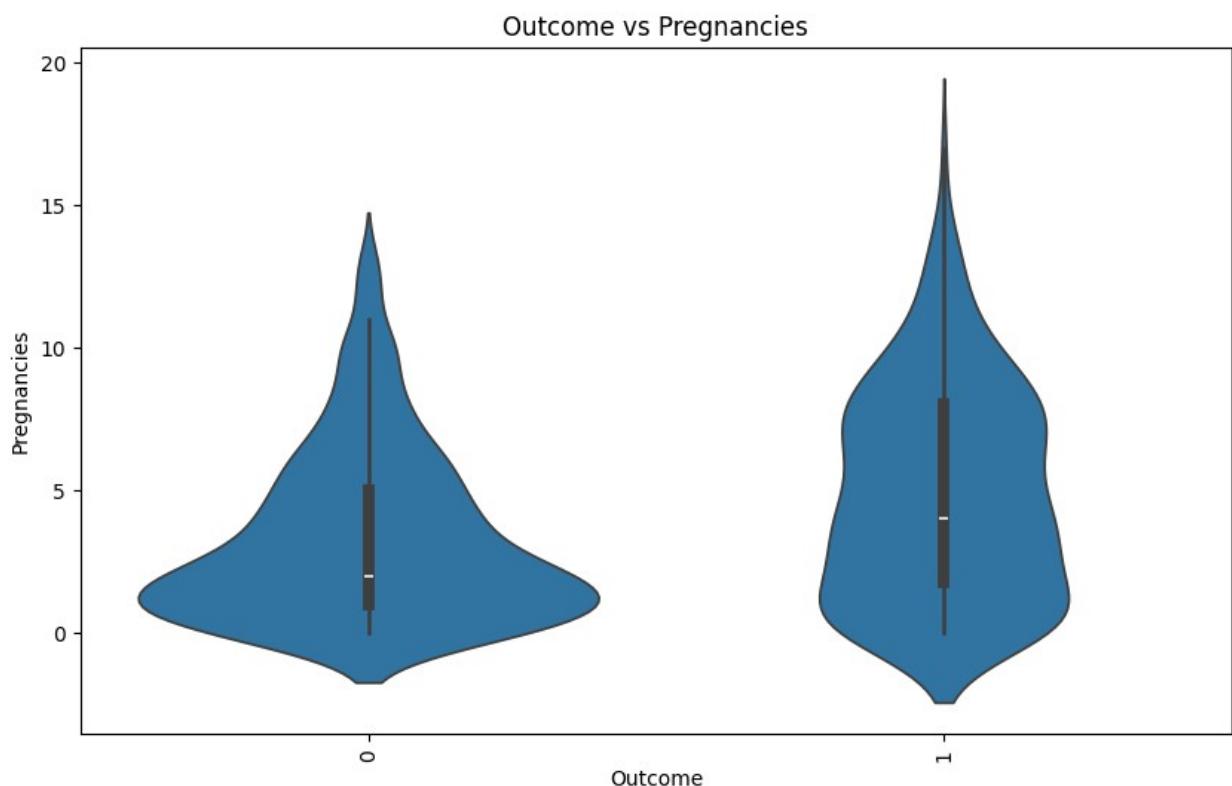
Outcome vs DiabetesPedigreeFunction



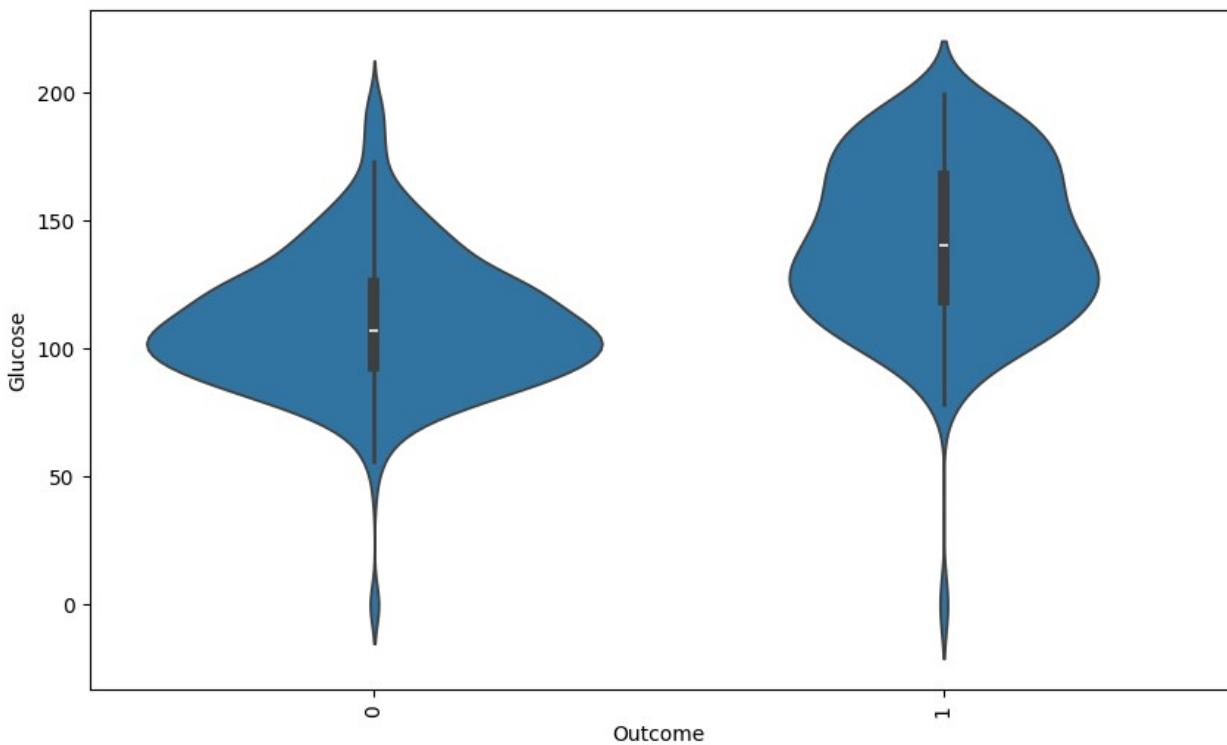
Outcome vs Age



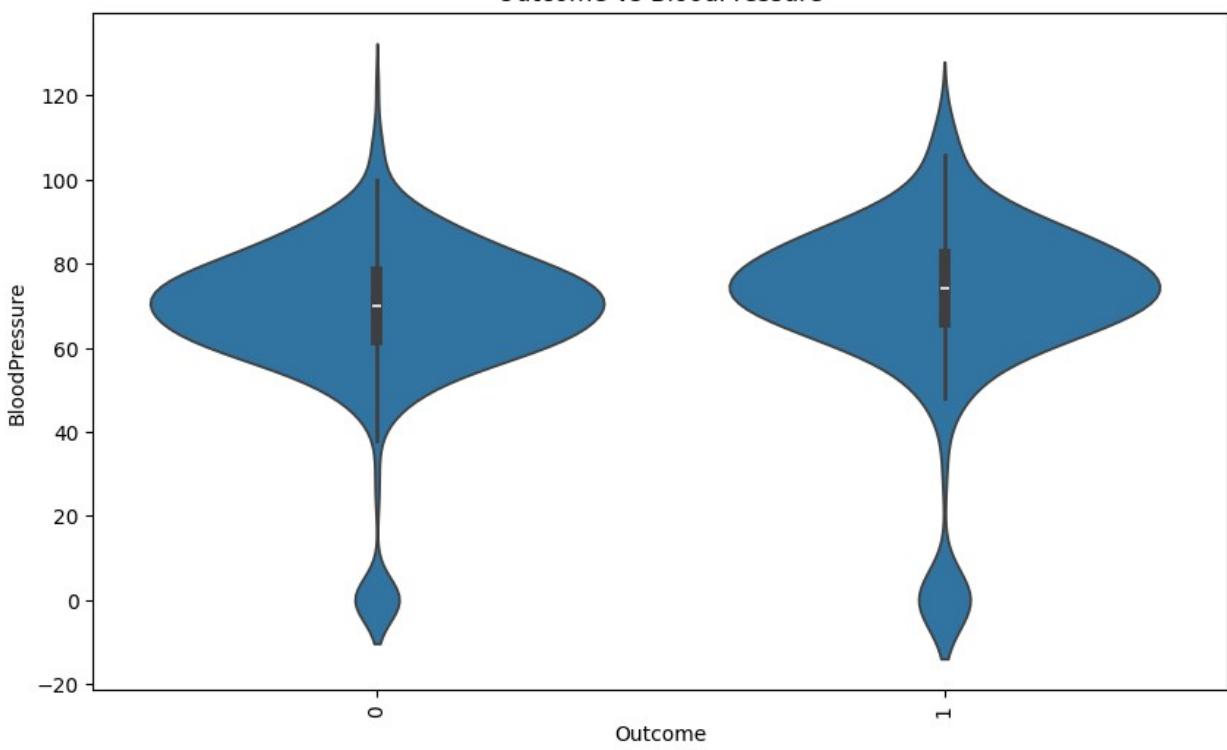
```
for dis in discrete:  
    for cont in continuous:  
        plt.figure(figsize=(10, 6))  
        ax = sns.violinplot(data=df, x=dis, y=cont)  
        plt.title(f'{dis} vs {cont}')  
        plt.xticks(rotation = 90)  
        plt.show()
```



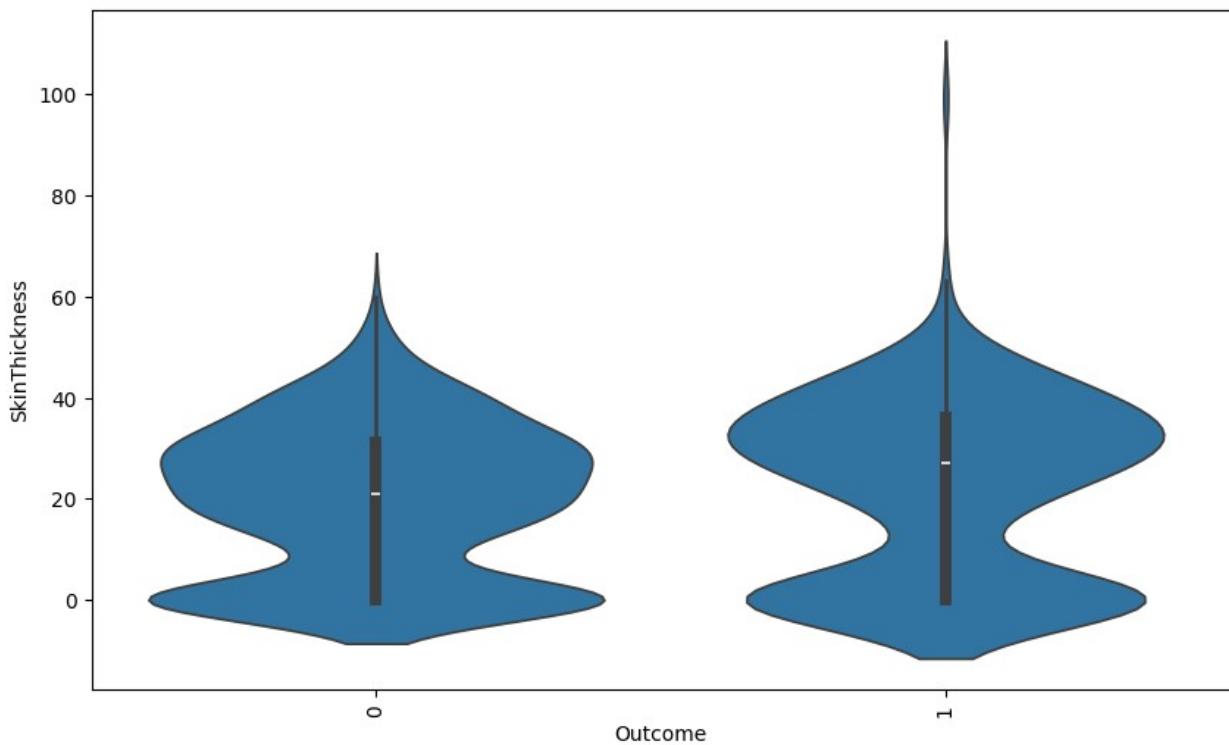
Outcome vs Glucose



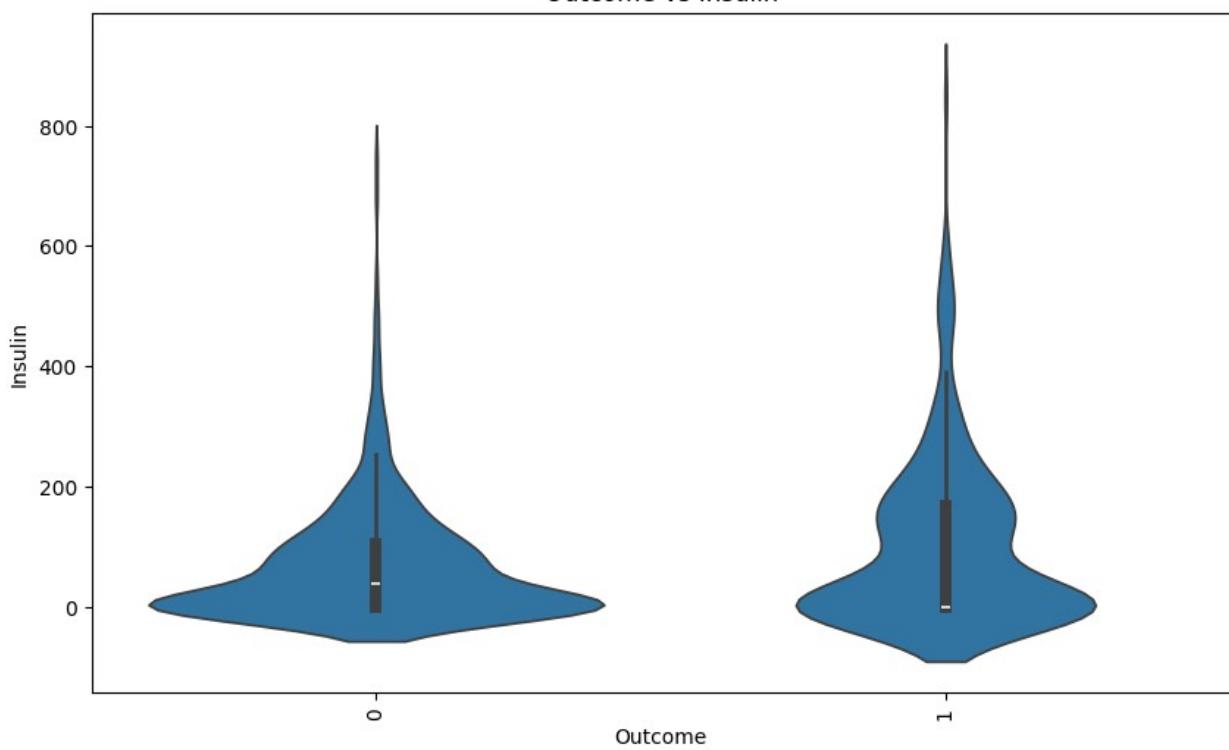
Outcome vs BloodPressure



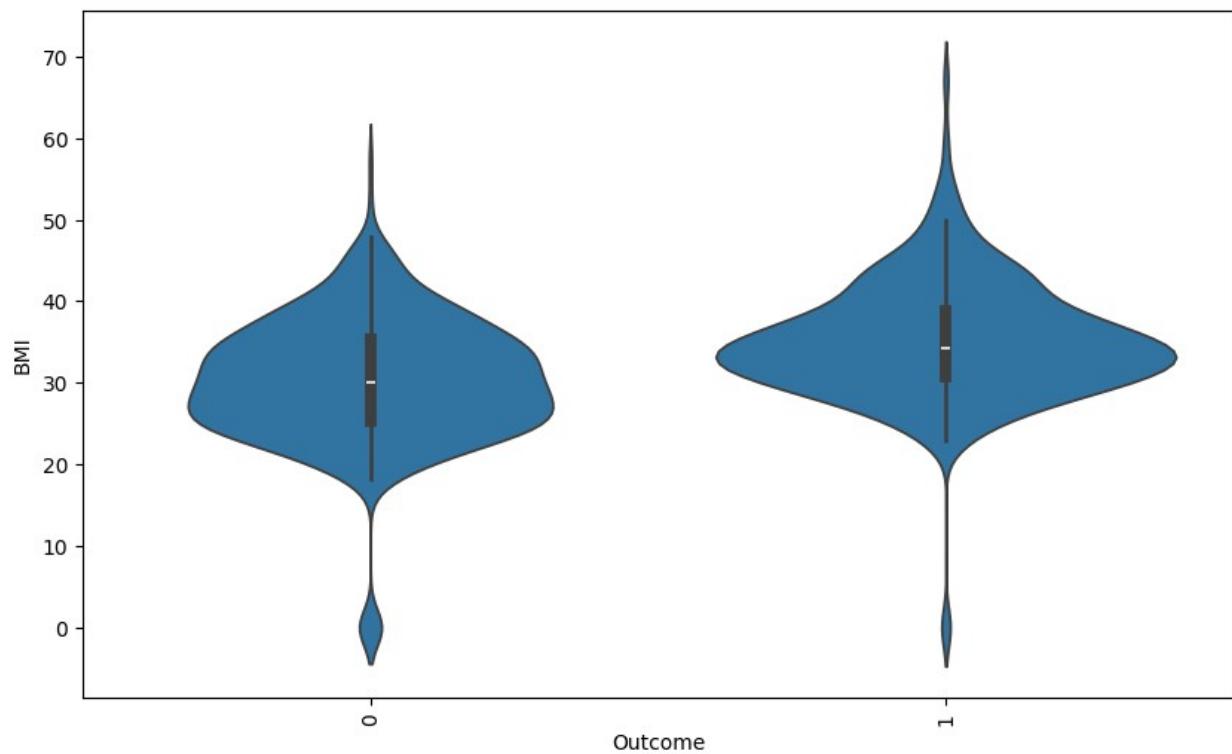
Outcome vs SkinThickness



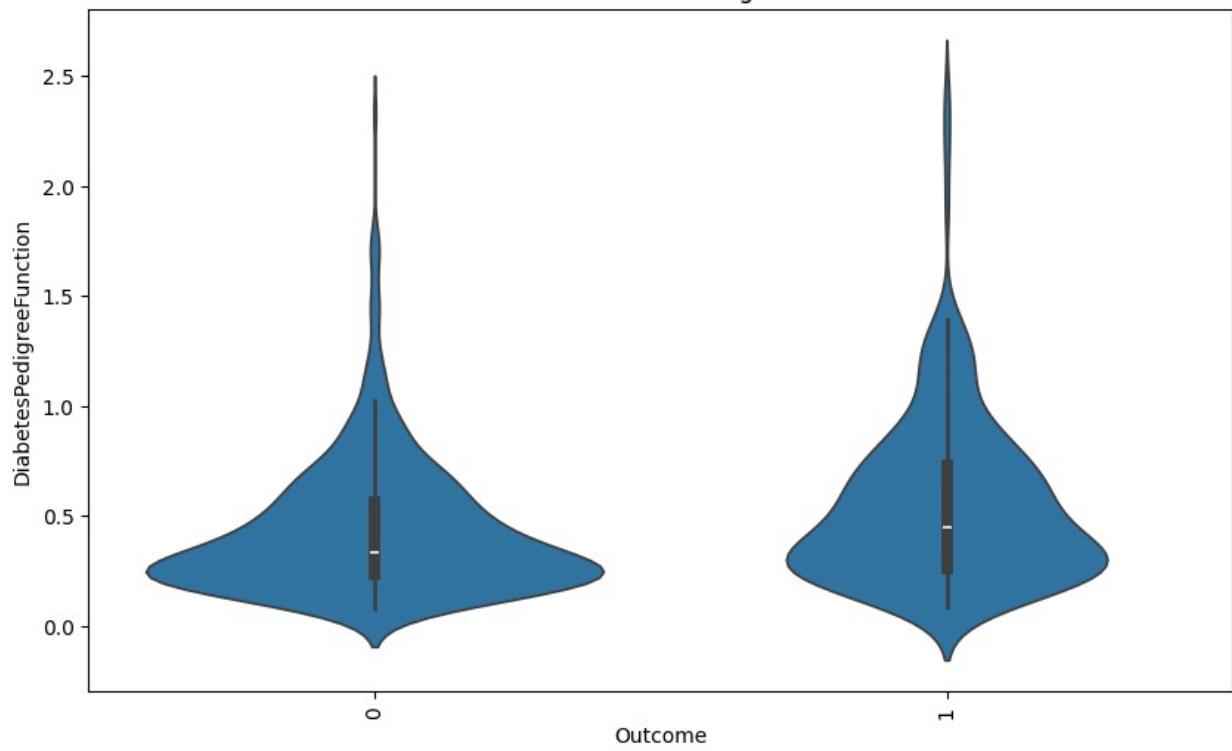
Outcome vs Insulin

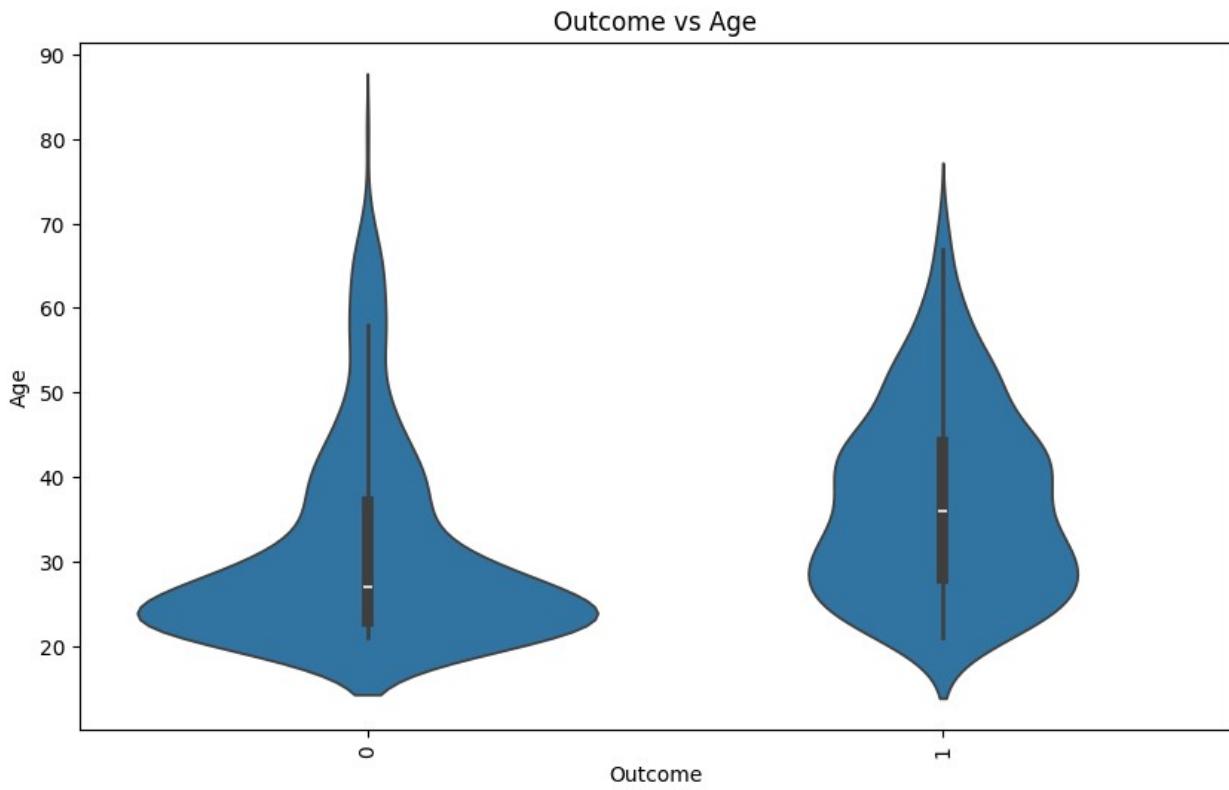


Outcome vs BMI



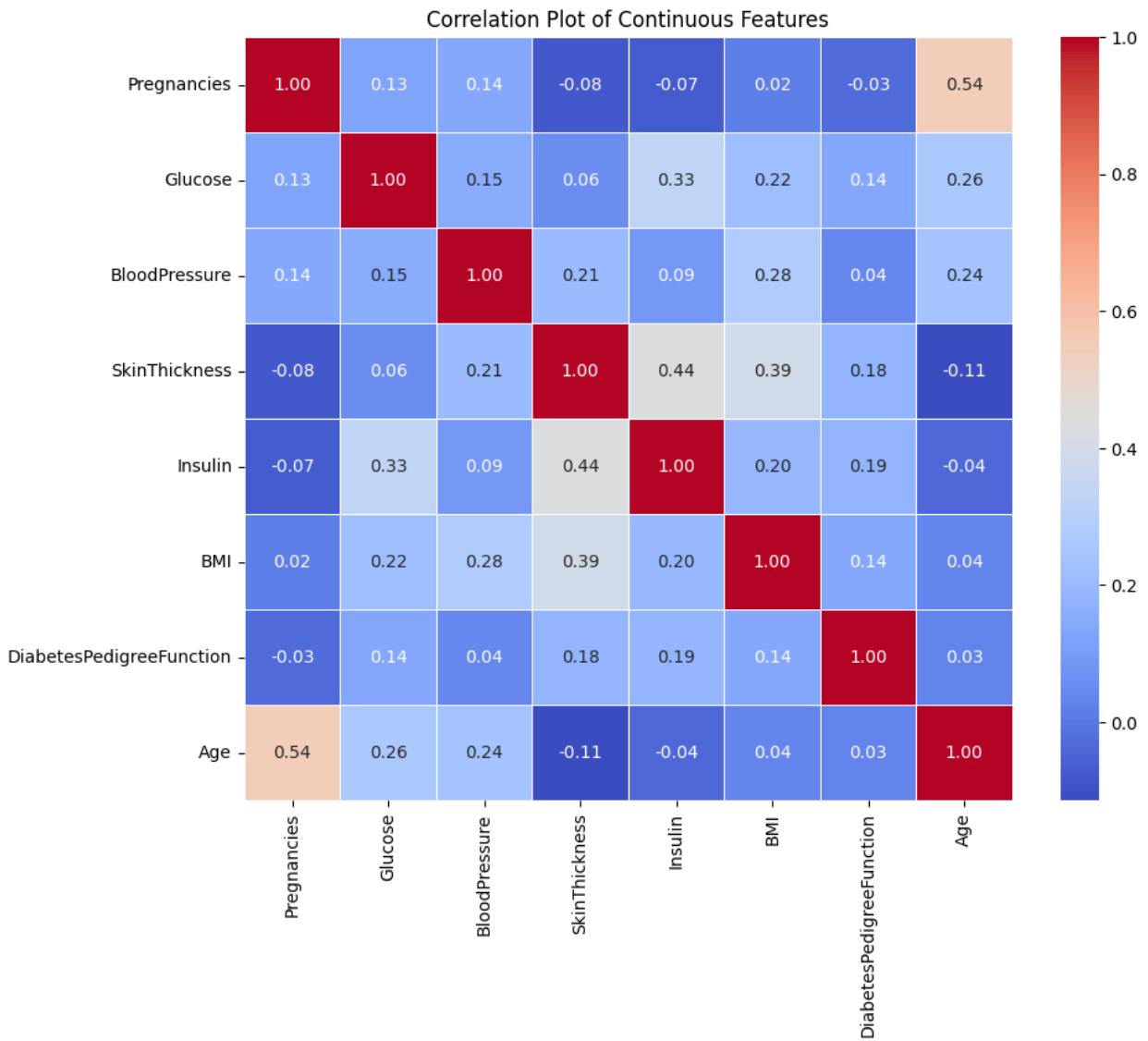
Outcome vs DiabetesPedigreeFunction





```
corr_matrix = df[continuous].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f",
            linewidths=0.5)
plt.title('Correlation Plot of Continuous Features')
plt.show()
```



`corr_matrix`

	Pregnancies	Glucose	BloodPressure
SkinThickness \ Pregnancies	1.000000	0.129459	0.141282
Glucose	0.129459	1.000000	0.152590
BloodPressure	0.141282	0.152590	1.000000
SkinThickness	-0.081672	0.057328	0.207371
Insulin	-0.073535	0.331357	0.088933
BMI	0.017683	0.221071	0.281805



DiabetesPedigreeFunction	-0.033523	0.137337	0.041265
0.183928			
Age	0.544341	0.263514	0.239528
0.113970			-
	Insulin	BMI	DiabetesPedigreeFunction
\			
Pregnancies	-0.073535	0.017683	-0.033523
Glucose	0.331357	0.221071	0.137337
BloodPressure	0.088933	0.281805	0.041265
SkinThickness	0.436783	0.392573	0.183928
Insulin	1.000000	0.197859	0.185071
BMI	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	0.185071	0.140647	1.000000
Age	-0.042163	0.036242	0.033561
	Age		
Pregnancies	0.544341		
Glucose	0.263514		
BloodPressure	0.239528		
SkinThickness	-0.113970		
Insulin	-0.042163		
BMI	0.036242		
DiabetesPedigreeFunction	0.033561		
Age	1.000000		

```

threshold = 0.7
features_to_drop = []
for i in range(len(corr_matrix.columns)):
    for j in range(i):
        if abs(corr_matrix.iloc[i, j]) > threshold:
            colname = corr_matrix.columns[i]
            if colname not in features_to_drop:
                features_to_drop.append(colname)

features_to_drop
[]
```



Feature Engineering

```
def bmi_category(bmi):
    if bmi <= 18.5:
        return 'Underweight'
    elif 18.5 < bmi <= 24.9:
        return 'Healthy'
    elif 25 <= bmi <= 29.9:
        return 'Pre-obese'
    elif 30 <= bmi <= 34.9:
        return 'Obesity class 1'
    elif 35 <= bmi <= 39.9:
        return 'Obesity class 2'
    else:
        return 'Obesity class 3'

df['BMI_Category'] = df['BMI'].apply(bmi_category)

def insulin_category(insulin):
    if insulin < 25:
        return 'Fasting or 3+ hours after glucose ingestion'
    elif 30 <= insulin <= 230:
        return '30 minutes after glucose administration'
    elif 18 <= insulin <= 276:
        return '1 hour after glucose ingestion'
    elif 16 <= insulin <= 166:
        return '2 hours after glucose ingestion'
    else:
        return 'Other'

df['Insulin_Category'] = df['Insulin'].apply(insulin_category)
```

df

	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   BMI_Catagory \
0                  0.627   50        1  Obesity class 1
1                  0.351   31        0      Pre-obese
2                  0.672   32        1       Healthy
3                  0.167   21        0      Pre-obese
4                 2.288   33        1  Obesity class 3
..                   ...
763                 0.171   63        0  Obesity class 1
764                 0.340   27        0  Obesity class 2
765                 0.245   30        0      Pre-obese
766                 0.349   47        1  Obesity class 1
767                 0.315   23        0  Obesity class 1

                           Insulin_Catagory
0  Fasting or 3+ hours after glucose ingestion
1  Fasting or 3+ hours after glucose ingestion
2  Fasting or 3+ hours after glucose ingestion
3      30 minutes after glucose administration
4      30 minutes after glucose administration
..                   ...
763      30 minutes after glucose administration
764  Fasting or 3+ hours after glucose ingestion
765      30 minutes after glucose administration
766  Fasting or 3+ hours after glucose ingestion
767  Fasting or 3+ hours after glucose ingestion

```

[768 rows x 11 columns]

```

object_columns = df.select_dtypes(include=['object']).columns
print("Object type columns:")
print(object_columns)

numerical_columns = df.select_dtypes(include=['int64',
'float64']).columns
print("\nNumerical type columns:")
print(numerical_columns)

```

Object type columns:
Index(['BMI_Catagory', 'Insulin_Catagory'], dtype='object')

Numerical type columns:
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',



```

        'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
       dtype='object')

df['Outcome']

0      1
1      0
2      1
3      0
4      1
 ..
763     0
764     0
765     0
766     1
767     0
Name: Outcome, Length: 768, dtype: int64

df_new = pd.get_dummies(df, columns=['BMI_Category',
 'Insulin_Category'])

numerical_columns_1 = ['Pregnancies', 'Glucose', 'BloodPressure',
 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']

pivot_table = pd.pivot_table(df_new, values='Glucose',
index='Outcome', aggfunc='mean')
pivot_table

          Glucose
Outcome
0      109.980000
1      141.257463

pivot_table = pd.pivot_table(df_new, values=['Glucose',
 'BloodPressure', 'Insulin','BMI', 'Age'], index=['Outcome'],
aggfunc=['mean', 'max'])
pivot_table

          mean
max      \
          Age      BMI BloodPressure      Glucose      Insulin
Age    BMI
Outcome

0      31.190000  30.304200    68.184000  109.980000  68.792000
81   57.3
1      37.067164  35.142537    70.824627  141.257463  100.335821
70   67.1

          BloodPressure      Glucose      Insulin

```



Outcome			
0	122	197	744
1	114	199	846

Modeling

```

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_new[numerical_columns_1] =
scaler.fit_transform(df_new[numerical_columns_1])

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify =
y, random_state= 42, test_size = 0.20)

from imblearn.over_sampling import SMOTE

sm = SMOTE()

X_train, y_train = sm.fit_resample(X_train, y_train)

from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)

LogisticRegression()

y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))

Accuracy: 0.76
Precision: 0.62

```

```

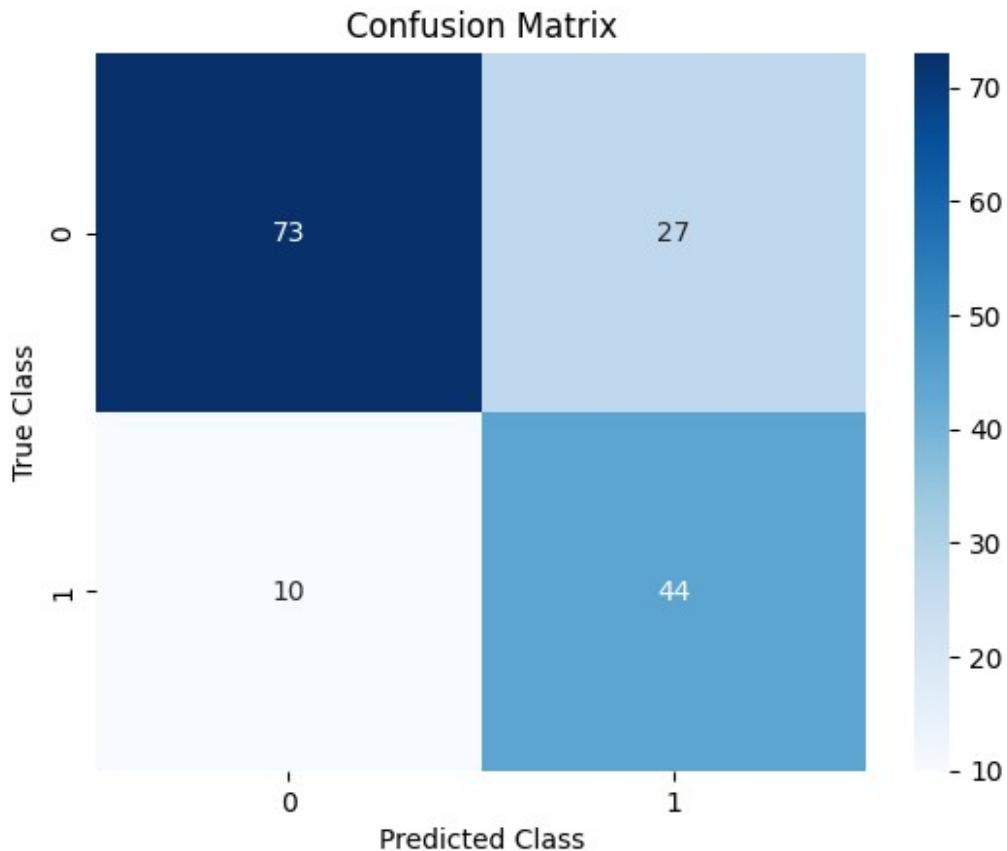
Recall: 0.81
F1-score: 0.70

cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

Confusion matrix:
[[73 27]
 [10 44]]

sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

```



```

report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)

Classification report:
      precision    recall  f1-score   support


```

	0	0.88	0.73	0.80	100
	1	0.62	0.81	0.70	54
accuracy				0.76	154
macro avg		0.75	0.77	0.75	154
weighted avg		0.79	0.76	0.76	154

```

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

DecisionTreeClassifier()

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))

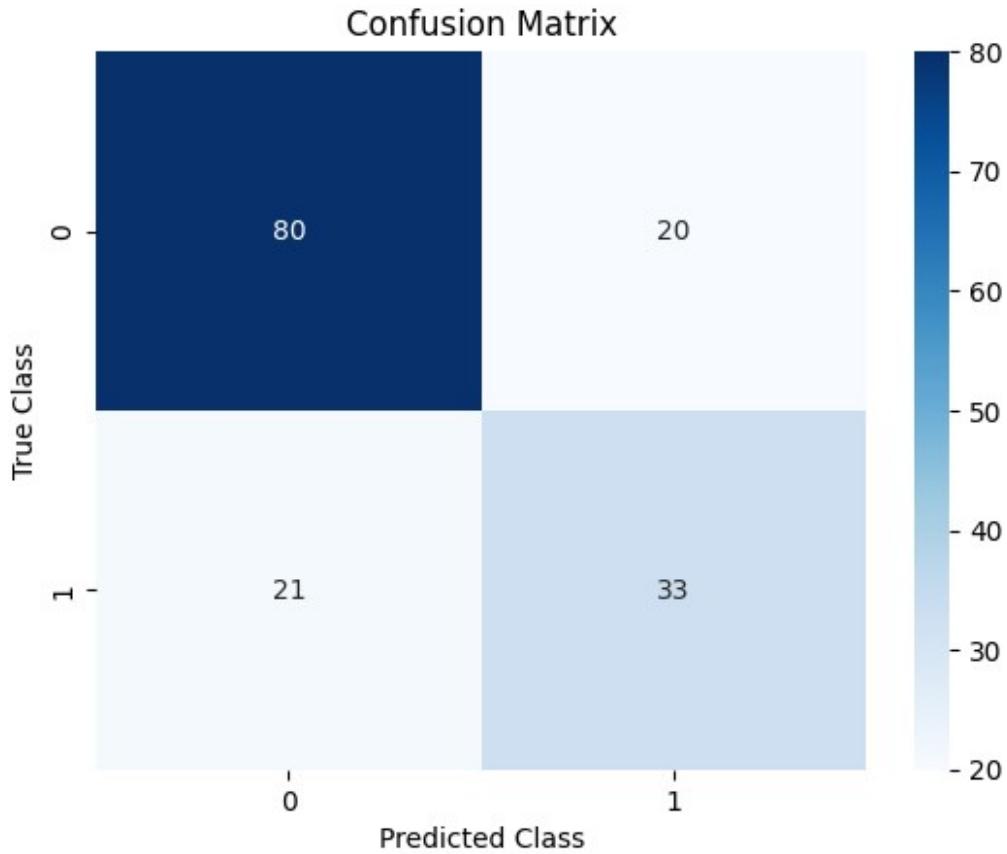
Accuracy: 0.73
Precision: 0.62
Recall: 0.61
F1-score: 0.62

cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

Confusion matrix:
[[80 20]
 [21 33]]

sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

```



```

report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)

Classification report:
      precision    recall  f1-score   support
          0       0.79      0.80      0.80      100
          1       0.62      0.61      0.62       54
   accuracy                           0.73      154
  macro avg       0.71      0.71      0.71      154
weighted avg       0.73      0.73      0.73      154

```

```

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

RandomForestClassifier()
y_pred = rf.predict(X_test)

```

```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))

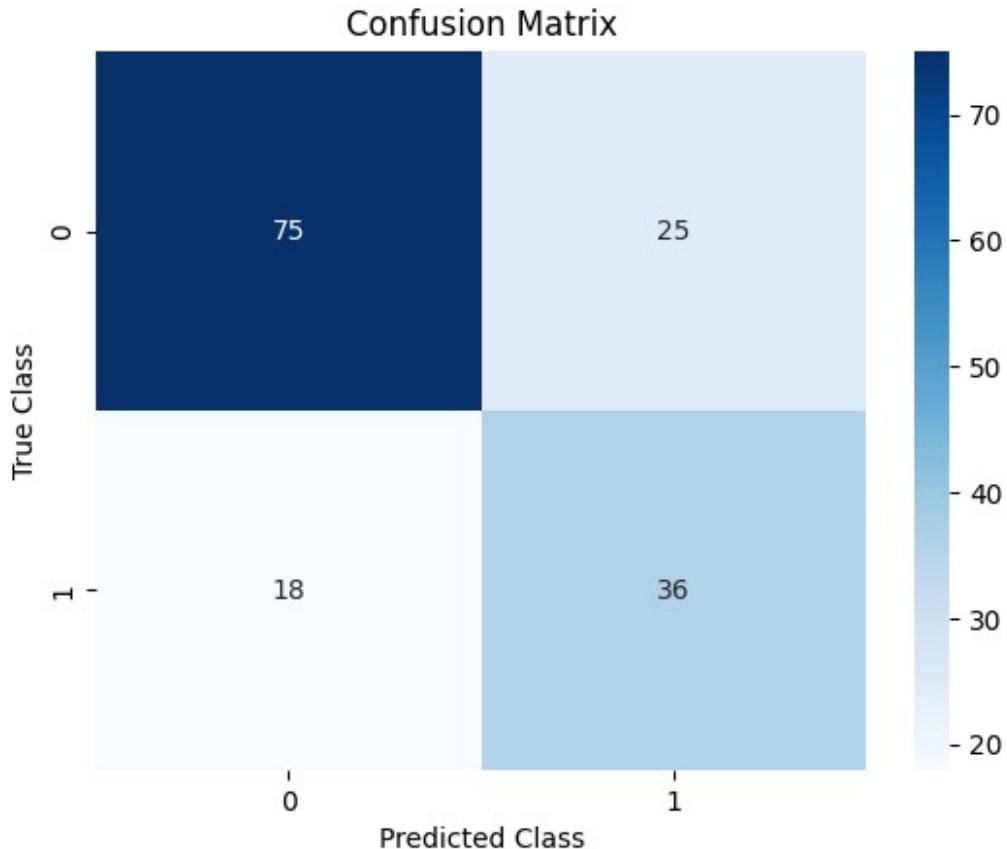
Accuracy: 0.72
Precision: 0.59
Recall: 0.67
F1-score: 0.63

cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

Confusion matrix:
[[75 25]
 [18 36]]

sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

```



```

report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)

Classification report:
      precision    recall  f1-score   support
          0       0.81     0.75     0.78     100
          1       0.59     0.67     0.63      54
   accuracy                           0.72     154
  macro avg       0.70     0.71     0.70     154
weighted avg       0.73     0.72     0.72     154

```

```

from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train, y_train)
SVC(kernel='linear')
y_pred = sv.predict(X_test)

```

```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))

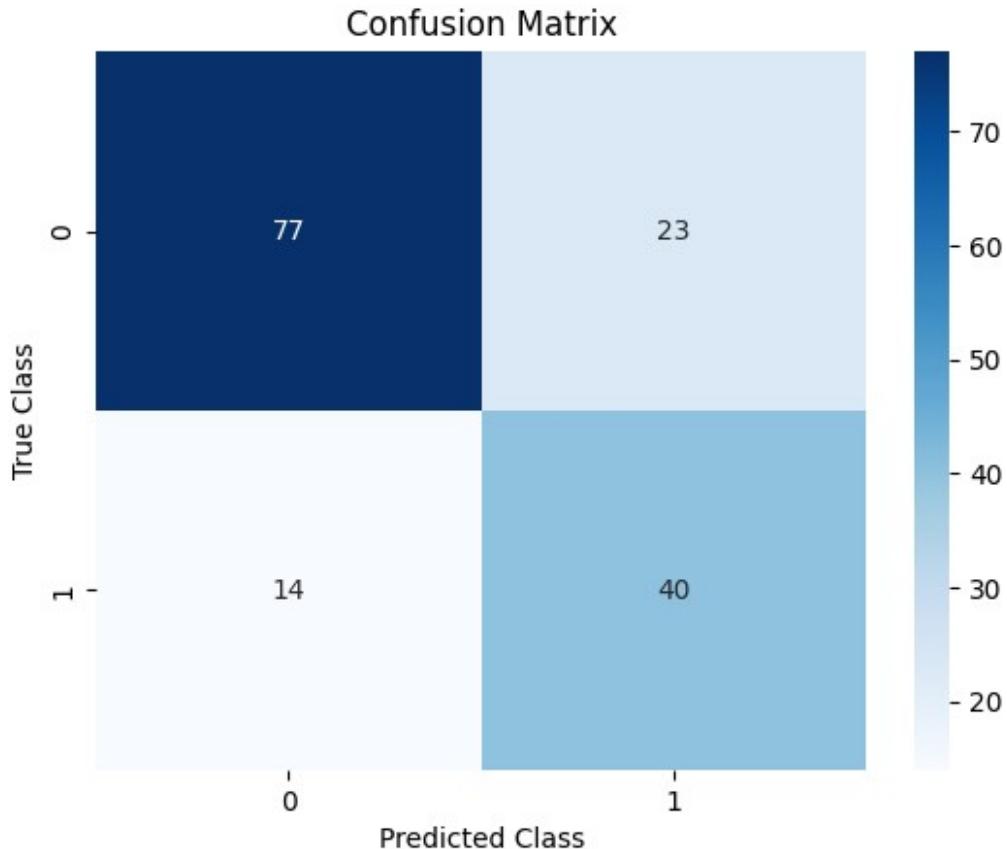
Accuracy: 0.76
Precision: 0.63
Recall: 0.74
F1-score: 0.68

cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

Confusion matrix:
[[77 23]
 [14 40]]

sns.heatmap(cm, annot=True, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

```



```

report = classification_report(y_test, y_pred)
print("Classification report:")
print(report)

Classification report:
             precision    recall  f1-score   support
              0       0.85      0.77      0.81      100
              1       0.63      0.74      0.68       54
        accuracy                           0.76      154
      macro avg       0.74      0.76      0.75      154
    weighted avg       0.77      0.76      0.76      154

from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)

GradientBoostingClassifier()
y_pred = gbc.predict(X_test)

```



```

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))

Accuracy: 0.76
Precision: 0.63
Recall: 0.76
F1-score: 0.69

import pickle

```

Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.

```

model_filename = 'svm_model_diabetes.pkl'
with open(model_filename, 'wb') as file:
    pickle.dump(sv, file)

print(f"Model saved to {model_filename}")

Model saved to svm_model_diabetes.pkl

```

Thanks !!!