All Libraries:

```
# Importing all necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

diabetes dataset:

```
#Loading the diabetes dataset from a CSV file
diabets_dataset = pd.read_csv(r'C:\Users\DELL\Desktop\Diabets
Predection\diabetes.csv')

# Displaying the first 5 rows of the dataset to understand its
structure
diabets_dataset.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

diabets_dataset.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
```

```
# Getting the number of rows and columns in the dataset
diabets_dataset.shape
```

```
(768, 9)
```

```
# Getting the summary statistics of the dataset
diabets_dataset.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  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
```

```
# Getting the count of unique values in the 'Outcome' column
diabets_dataset['Outcome'].value_counts()
```

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

```
# Calculating the mean of each feature grouped by the 'Outcome' column
diabets_dataset.groupby('Outcome').mean()
```

```
         Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
Outcome

0           3.298000  109.980000      68.184000      19.664000
68.792000
1           4.865672  141.257463      70.824627      22.164179
100.335821

                BMI  DiabetesPedigreeFunction        Age
Outcome
0        30.304200                  0.429734  31.190000
1        35.142537                  0.550500  37.067164
```

```
# Separating the features (X) and the target variable (Y)
X = diabets_dataset.drop(columns = 'Outcome' , axis =1)  # Features
(independent variables)
Y = diabets_dataset['Outcome']  # Target variable (dependent variable)

print(X)
```

```
     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
```

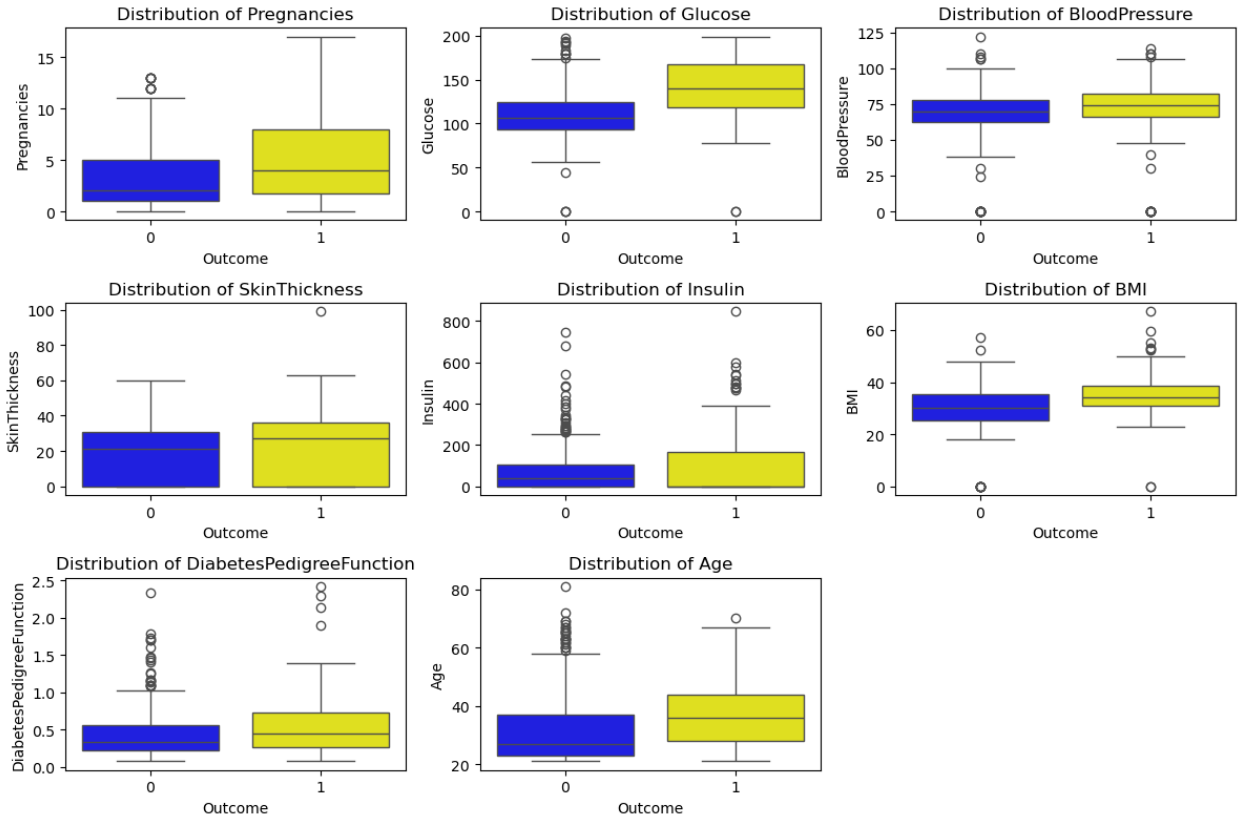
```
767            1       93             70             31        0  30.4


     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

```
print(Y)
```

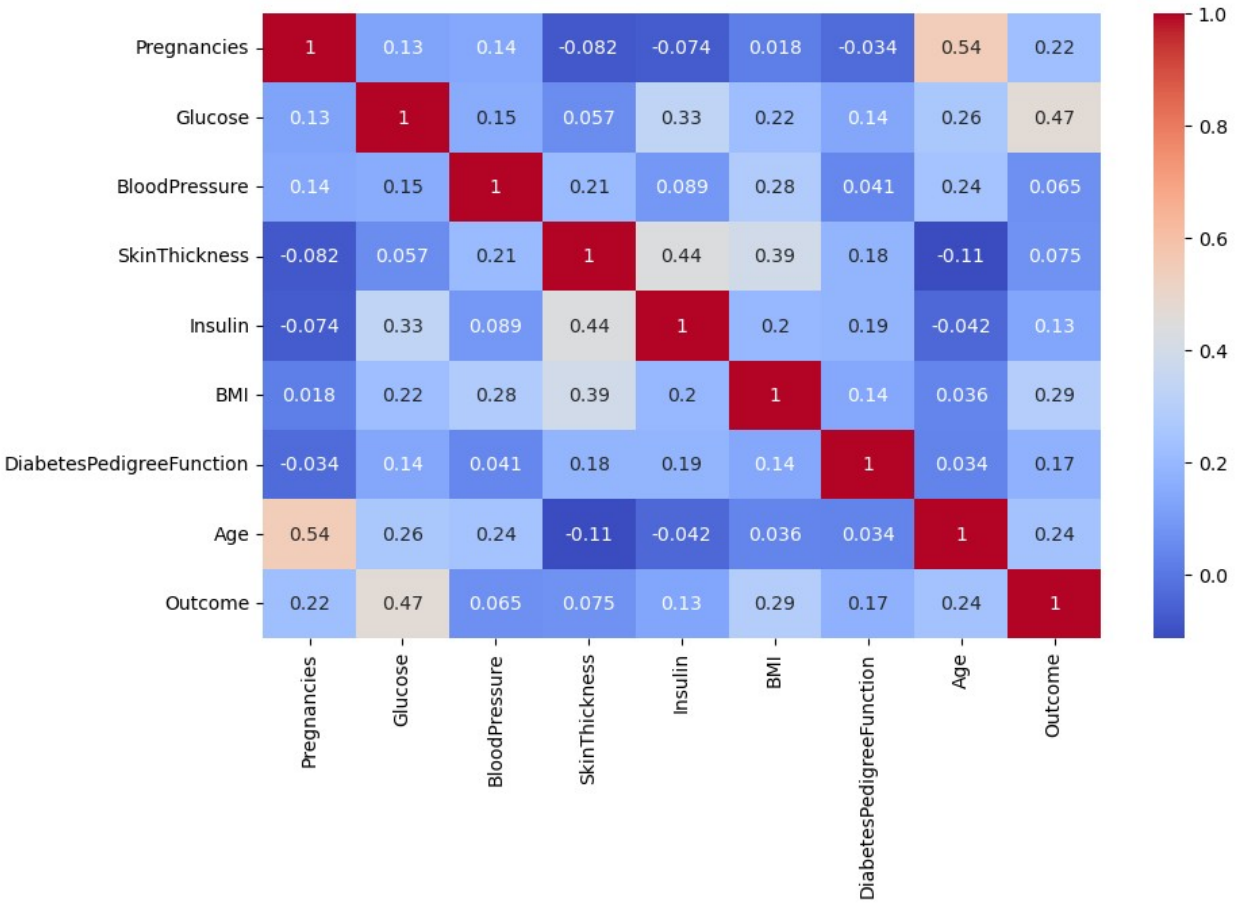
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
# Getting feature names
features = X.columns.tolist()

# Creating subplots (arrange in a 3x3 grid)
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(12, 8))
axs = axs.flatten()

# Define custom colors: 0 (Non-Diabetic) = Blue, 1 (Diabetic) = Yellow
custom_palette = {0: "blue", 1: "yellow"}

# Plot boxplots for each feature
for index, feature in enumerate(features):
    sns.boxplot(data=diabets_dataset, x='Outcome', y=feature,
ax=axs[index], hue="Outcome", palette=custom_palette, legend=False)
    axs[index].set_title(f'Distribution of {feature}')

# Hide any extra subplot if the number of features is less than 9
if len(features) < 9:
```

```
    for i in range(len(features), 9):
        axs[i].set_visible(False)

# Display the plots
plt.tight_layout()
plt.show()
```




```
corr_matrix = diabets_dataset.corr()

fig, ax = plt.subplots(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap = 'coolwarm', ax=ax)
plt.show()
```




```
print(diabets_dataset.isnull().sum())
```

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

```
print(diabets_dataset.duplicated().sum())
```

```
0
```

```
# Initializing the StandardScaler to scale the features
scaler = StandardScaler()

# Scaling the features in X using the StandardScaler
X_train_scaled = scaler.fit_transform(X)
```

```
print(X_train_scaled)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

```
# Assigning the scaled features to X and the target variable to Y
X = X_train_scaled  # Scaled features
Y = diabets_dataset['Outcome']  # Target variable (Outcome)

print(X)
print(Y)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
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
```

```
# Splitting the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
test_size=0.2, stratify=Y, random_state=2)

# Initializing the Support Vector Classifier (SVC) with a linear
kernel
classifier = svm.SVC(kernel = 'linear')

# Training the classifier using the training data
classifier.fit(X_train, Y_train)
```

```
SVC(kernel='linear')
```

```
# Making predictions on the training data and calculating accuracy
X_train_prediction = classifier.predict(X_train)  # Predictions on the
training set
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
# Accuracy on the training set

# Printing the accuracy score of the training data
print('accuracy score of the training data : ',
training_data_accuracy)
```

```
accuracy score of the training data :  0.7866449511400652
```

```
# Making predictions on the testing data and calculating accuracy
X_test_prediction = classifier.predict(X_test)  # Predictions on the
test set
testing_data_accuracy = accuracy_score(X_test_prediction, Y_test)  #
Accuracy on the test set

# Printing the accuracy score of the testing data
print('Accuracy score of the testing data: ', testing_data_accuracy)
```

```
Accuracy score of the testing data:  0.7727272727272727
```

```
# Defining feature names and input data
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
input_data = (4, 110, 92, 0, 0, 37.6, 0.191, 30)  # Sample input data

# Converting the input data into a numpy array and then into a pandas
DataFrame
input_data_as_numpy_array = np.asarray(input_data)
input_data_df = pd.DataFrame([input_data_as_numpy_array],
columns=feature_names)

# Scaling the input data using the previously fitted scaler
std_data = scaler.transform(input_data_df)

# Making a prediction using the trained classifier
prediction = classifier.predict(std_data)
```

```python
# Printing the result of the prediction
if prediction[0] == 0:
    print('The person is not diabetic')
else:
    print('The person is diabetic')
```

```
The person is not diabetic
```