Diebetes prediction using Machine learning

Life cycle of Machine Learning

- Understanding the problem Statement
- Data Collection
- Data Cleaning
- Exploring data Analysis
- Data pre-processing
- Model Training
- Choose best model

1.Problem Statement :

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Exploratory Data Analysis

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Read the dataset

In [4]:
```
df = pd.read_csv("/content/diabetes_dataset (1).csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1

Feature information

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Glucose                   768 non-null    int64
 1   BloodPressure             768 non-null    int64
 2   SkinThickness             768 non-null    int64
 3   Insulin                   768 non-null    int64
 4   BMI                       768 non-null    float64
 5   DiabetesPedigreeFunction  768 non-null    float64
 6   Age                       768 non-null    int64
 7   Outcome                   768 non-null    int64
dtypes: float64(2), int64(6)
memory usage: 48.1 KB
```

In [7]:
```
#Descriptive statistics of the data set accessed
df.describe()
```

Out[7]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Check for null values

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Check Duplicacy in the data

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

Replacing 0 value with mean and median

In [10]:
```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness'
print(df.isnull().sum())
```

```
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

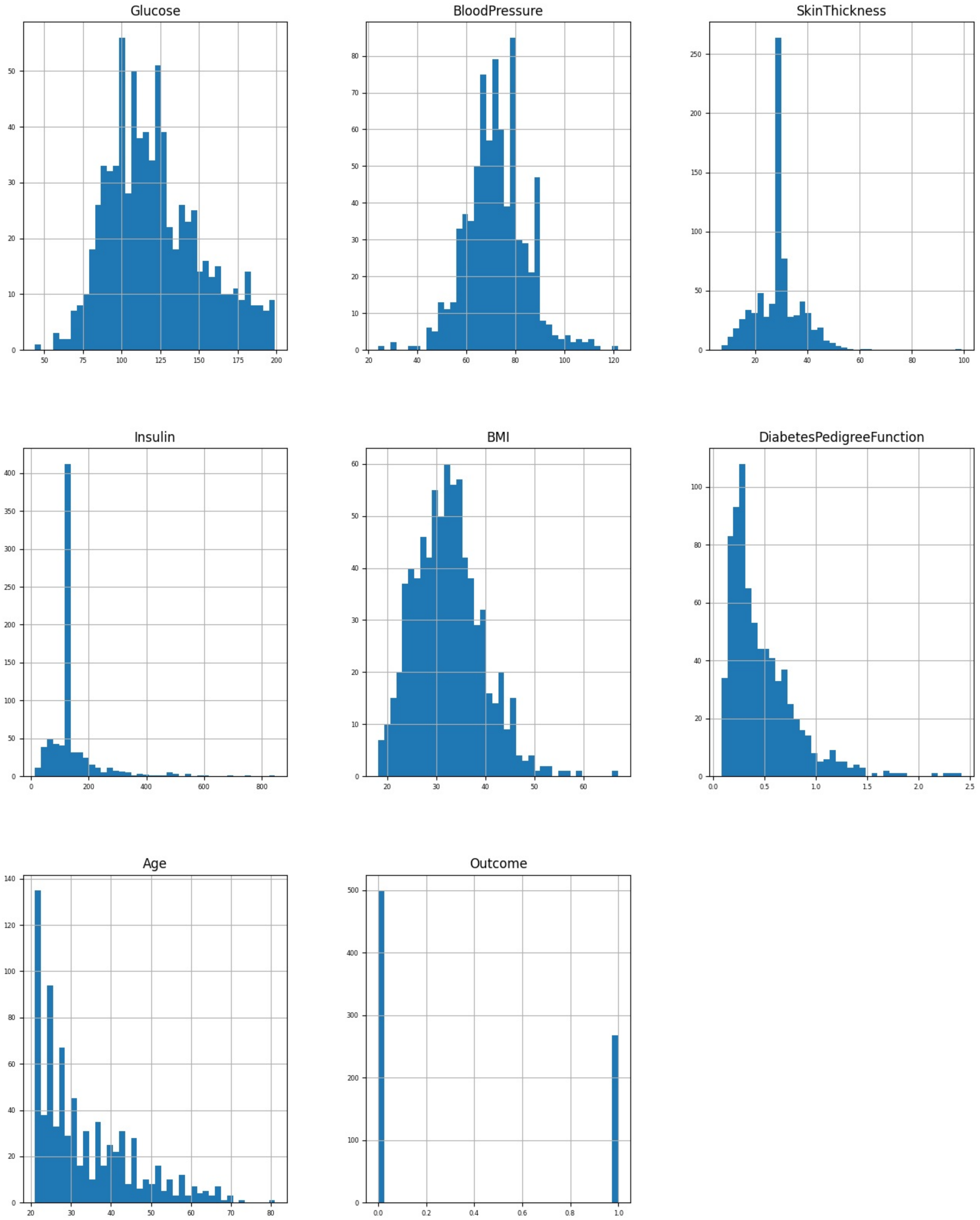
Replace null values with Specific value

In [11]:
```
df['Glucose'].fillna(df['Glucose'].mean(), inplace = True)
df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace = True)
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace = True)
df['Insulin'].fillna(df['Insulin'].median(), inplace = True)
df['BMI'].fillna(df['BMI'].median(), inplace = True)
```

Data Distribution

In [ ]:
```
df.hist(figsize=(16, 20), bins=40, xlabelsize=6, ylabelsize=6)
```

Out[ ]:
```
array([[<Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>,
        <Axes: title={'center': 'SkinThickness'}>],
       [<Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>,
        <Axes: title={'center': 'DiabetesPedigreeFunction'}>],
       [<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>, <Axes: >]], dtype=object)
```



In [ ]:

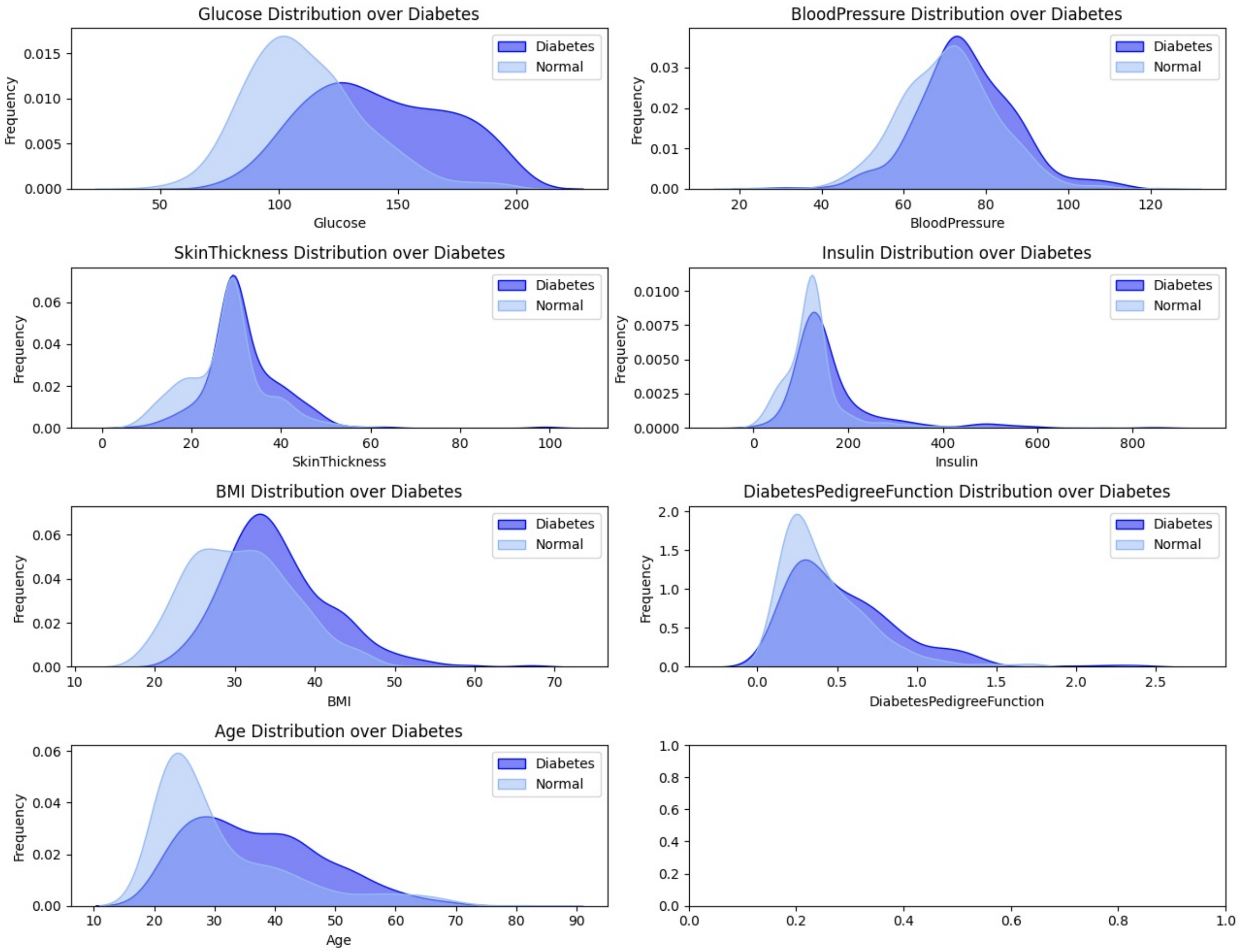
```
fig, axes = plt.subplots(nrows=len(df.columns) // 2, ncols=2, figsize=(13, 10))

for idx, column in enumerate(df.drop(columns = 'Outcome')):
    row_idx = idx // 2
    col_idx = idx % 2

    sns.kdeplot(df[df["Outcome"] == 1][column], alpha=0.5, fill=True, color="#000CEB", label="Diabetes", ax=axe
    sns.kdeplot(df[df["Outcome"] == 0][column], alpha=0.5, fill=True, color="#97B9F4", label="Normal", ax=axes[

    axes[row_idx, col_idx].set_xlabel(column)
    axes[row_idx, col_idx].set_ylabel("Frequency")
    axes[row_idx, col_idx].set_title(f"{column} Distribution over Diabetes")
    axes[row_idx, col_idx].legend()
```

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



In [ ]:

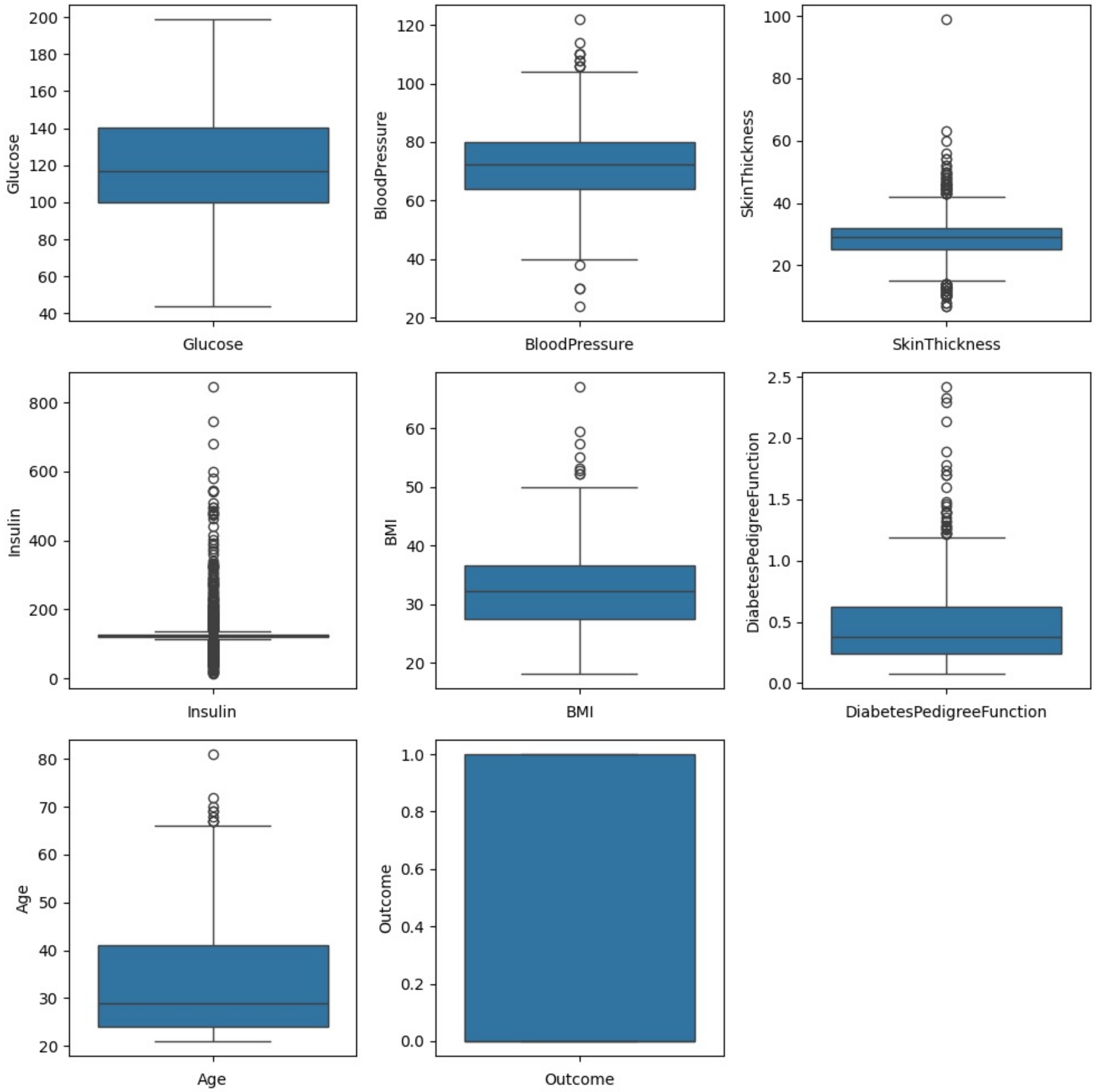
```python

```

Checking and Removing Outliers

In [ ]:

```python
plt.figure(figsize = (10,10), facecolor = 'white')
plotnumber = 1
for i in df.columns:
    ax = plt.subplot(3,3, plotnumber)
    sns.boxplot(df[i])
    plt.xlabel(i, fontsize = 10)
    plotnumber +=1
plt.tight_layout()
plt.show()
```



In [15]:
```python
percentage_outliers_dict = {}
for column_name in df.columns:
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column_name] < lower_bound) | (df[column_name] > upper_bound)]
    percentage_outliers = (len(outliers) / len(df)) * 100
    percentage_outliers_dict[column_name] = percentage_outliers
for column_name, percentage_outliers in percentage_outliers_dict.items():
    print(f"Percentage of outliers in column '{column_name}':{percentage_outliers:.2f}%")
```

```
Percentage of outliers in column 'Glucose':0.00%
Percentage of outliers in column 'BloodPressure':1.82%
Percentage of outliers in column 'SkinThickness':11.33%
Percentage of outliers in column 'Insulin':45.05%
Percentage of outliers in column 'BMI':1.04%
Percentage of outliers in column 'DiabetesPedigreeFunction':3.78%
Percentage of outliers in column 'Age':1.17%
Percentage of outliers in column 'Outcome':0.00%
```

In [16]:
```python
# This Code is doing IQR based filtering
def outlier_capping(dataframe: pd.DataFrame, outliers:list):
    capped_df = dataframe.copy()
    for i in outliers:
        q1 = capped_df[i].quantile(0.25)
        q3 = capped_df[i].quantile(0.75)
        iqr = q3 - q1
        upper_limit = q3 + 1.5 *iqr
```

```
        lower_limit = q3 - 1.5 *iqr
        capped_df.loc[capped_df[i] >upper_limit, i] = upper_limit
        capped_df.loc[capped_df[i] <lower_limit, i] = lower_limit
    return capped_df
```

In [17]:
```
df_filtered=outlier_capping(df,df.columns)
```

In [18]:
```
df_filtered.describe()
```

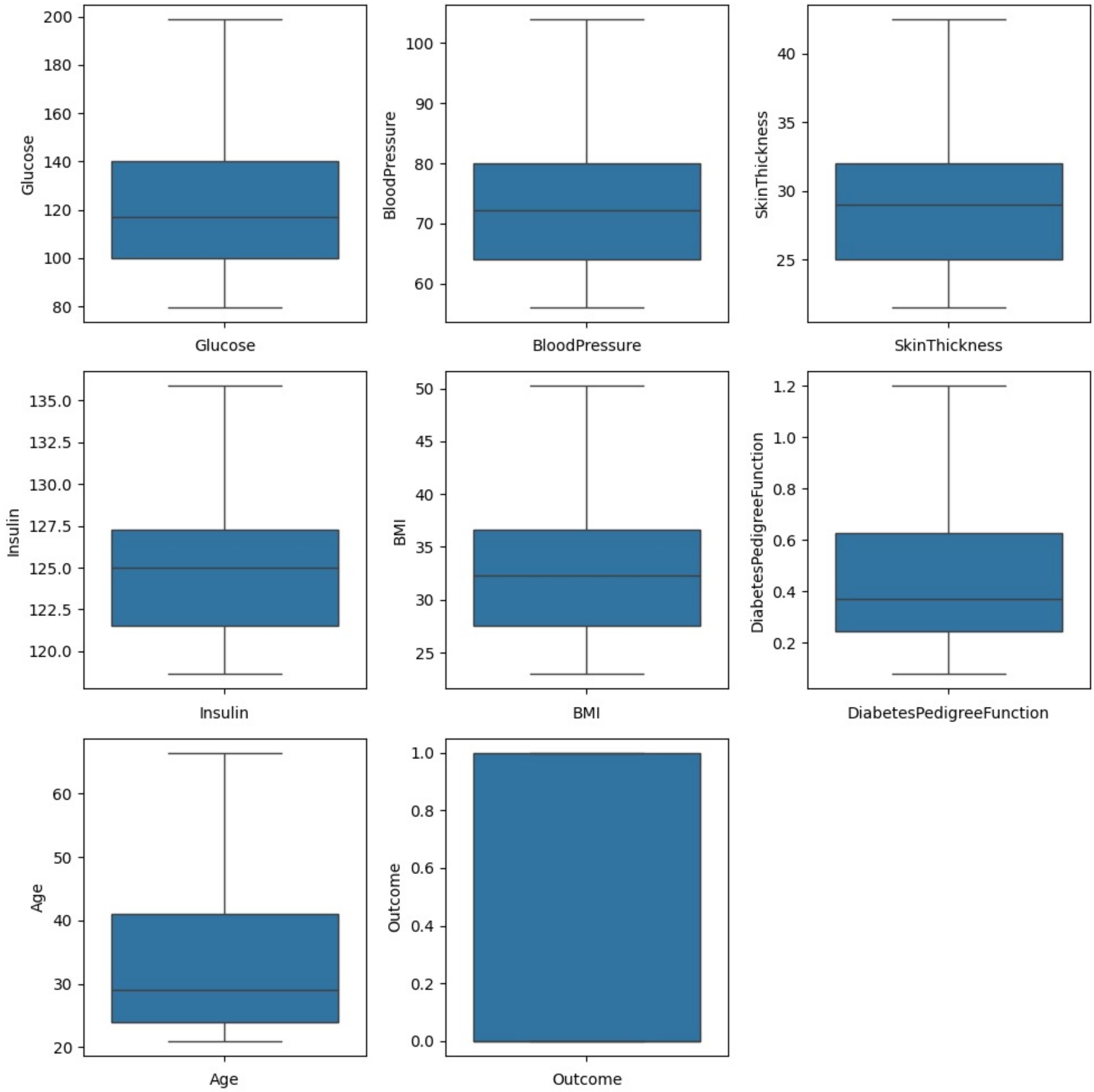
Out[18]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	122.094315	72.811434	29.647135	126.033529	32.517839	0.458914	33.199870	0.348958
std	29.754639	10.879635	6.235897	6.072769	6.459613	0.285596	11.628404	0.476951
min	79.500000	56.000000	21.500000	118.625000	22.950000	0.078000	21.000000	0.000000
25%	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	117.000000	72.202592	29.000000	125.000000	32.300000	0.372500	29.000000	0.000000
75%	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	199.000000	104.000000	42.500000	135.875000	50.250000	1.200000	66.500000	1.000000

Printing the data afterward we can notice two of our extreme observations which were acting as outliers get removed.

Now generate box plots for each column in a pandas DataFrame. It visualizes how the data is spread out, what its distribution looks like, and identifies any outliers

In [ ]:
```
plt.figure(figsize = (10,10), facecolor = 'white')
plotnumber = 1
for i in df_filtered.columns:
    ax = plt.subplot(3,3, plotnumber)
    sns.boxplot(df_filtered[i])
    plt.xlabel(i, fontsize = 10)
    plotnumber +=1
plt.tight_layout()
plt.show()
```



By looking at these box plots, you can gain insights into the distribution, spread, median, and outliers of each column's data. This helps in understanding the characteristics of the data in each column.

Scaling

The provided code utilizes MinMaxScaler from sklearn to perform Min-Max scaling on the columns of a DataFrame, transforming the data to a fixed range, typically between 0 and 1. This scaling ensures uniformity across features without distorting their original differences. The scaled data is then stored in a new DataFrame, maintaining the original column names. This normalization process is beneficial for various machine learning algorithms, particularly those reliant on consistent feature scales, such as distance-based methods like k-nearest neighbors or optimization algorithms like gradient descent.

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaled=MinMaxScaler()
df_scaled = scaled.fit_transform(df_filtered)
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
```

In [ ]:
```
df_scaled.head()
```

Out[ ]:

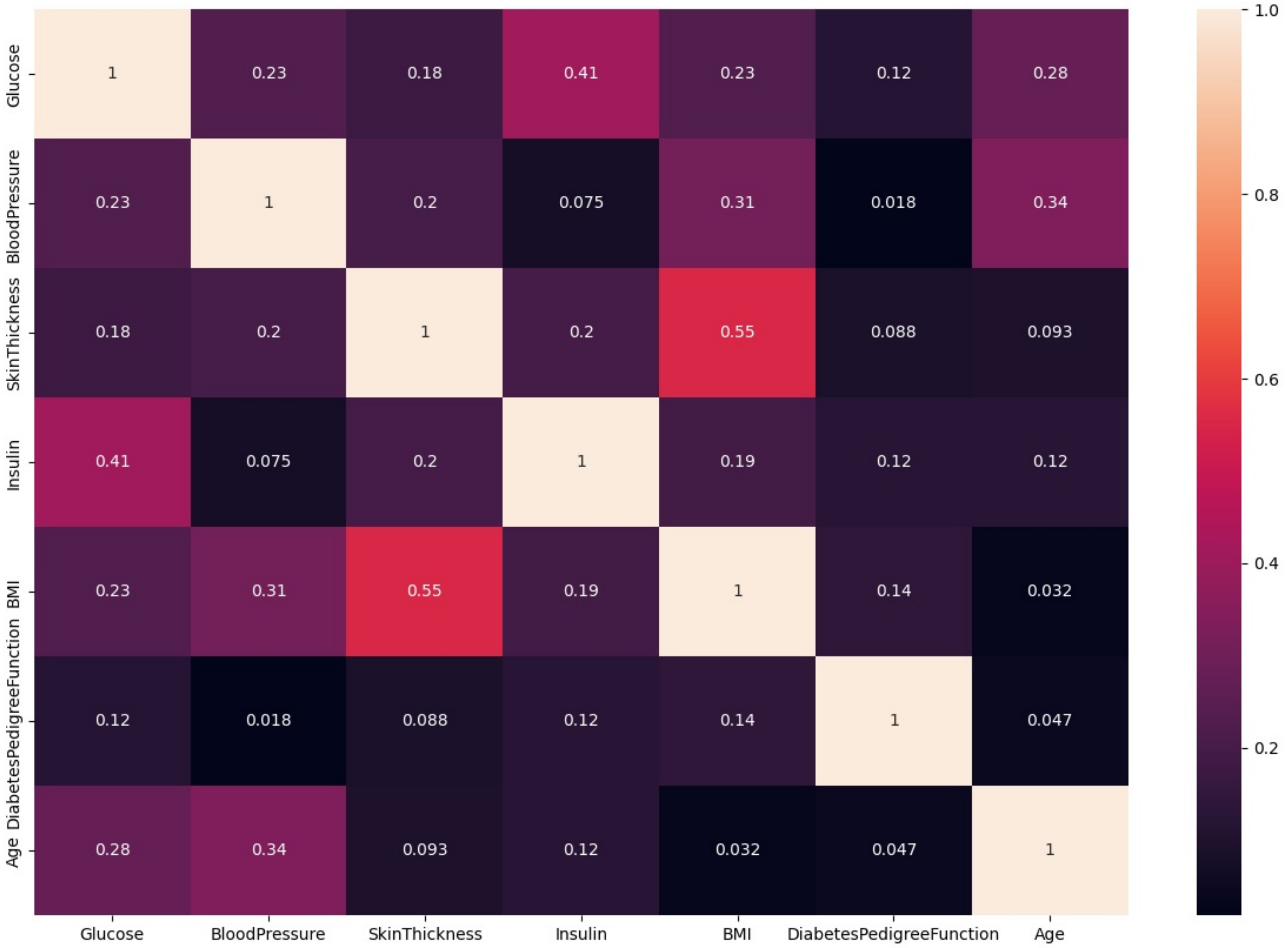
	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.573222	0.333333	0.642857	0.369565	0.390110	0.489305	0.637363	1.0
1	0.046025	0.208333	0.357143	0.369565	0.133700	0.243316	0.219780	0.0
2	0.866109	0.166667	0.357143	0.369565	0.012821	0.529412	0.241758	1.0
3	0.079498	0.208333	0.071429	0.000000	0.188645	0.079323	0.000000	0.0
4	0.481172	0.000000	0.642857	1.000000	0.738095	1.000000	0.263736	1.0

Multicollinearity

In [ ]:

```
corr = df_scaled.drop(columns= 'Outcome').corr()
fig , ax = plt.subplots(figsize=(15 , 10))
sns.heatmap(corr ,annot= True , ax=ax )
```

Out[ ]: <Axes: >

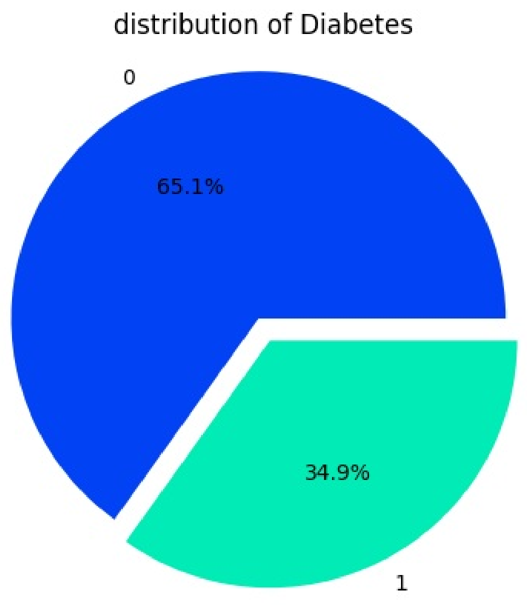


Target column balance

In [ ]:

```
Diabetes_column = df.Outcome.value_counts()

# pie chart for target column
plt.pie(Diabetes_column, labels = Diabetes_column.index, autopct="%1.1f%%", explode = [0,0.1], colors = ["#0142I
plt.title("distribution of Diabetes")
plt.axis("equal")
plt.show()
```



model Building

In [ ]:
```
X= df_scaled.drop(["Outcome"], axis=1)
y = df_scaled["Outcome"]
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [ ]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

Using Logistic Regression,Support Vector Classifier,GaussianNB,Decision Tree Classifier and KNeighbors Classifier to build the model.By this we can get best accuracy score & precicsion

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```
svc = SVC(kernel='linear', gamma=0.001)
knc = KNeighborsClassifier(n_neighbors=11)
gnb = GaussianNB()
dtc = DecisionTreeClassifier(max_depth=5)
lrc = LogisticRegression()
```

In [ ]:
```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': gnb,
    'DT': dtc,
    'LR': lrc,
}
```

In [ ]:
```
def train_classifier(clf,X_train,y_train,X_test,y_test):
    clf.fit(X_train,y_train)
    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)
    return accuracy,precision,recall
```

In [ ]:
```
train_classifier(svc,X_train,y_train,X_test,y_test)
```

Out[ ]:
```
(0.7662337662337663, 0.6097560975609756, 0.5555555555555556)
```

In [ ]:
```
accuracy_scores = []
precision_scores = []
recall_scores = []

for name,clf in clfs.items():

    current_accuracy,current_precision,current_recall = train_classifier(clf, X_train,y_train,X_test,y_test)
```

```
    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)
    print("Recall - ",current_recall)

    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
    recall_scores.append(current_recall)
```

```
For  SVC
Accuracy -  0.7662337662337663
Precision -  0.6097560975609756
Recall -  0.5555555555555556
For  KN
Accuracy -  0.7662337662337663
Precision -  0.6046511627906976
Recall -  0.5777777777777777
For  NB
Accuracy -  0.7337662337662337
Precision -  0.5434782608695652
Recall -  0.5555555555555556
For  DT
Accuracy -  0.7597402597402597
Precision -  0.6
Recall -  0.5333333333333333
For  LR
Accuracy -  0.7597402597402597
Precision -  0.6
Recall -  0.5333333333333333
```

For SVC the accuracy score-76.62%,precision-60.9% and recall-55.55% For KN the accuracy score-76.62%,precision-60.4% and recall-57.77% For NB the accuracy score-73.37%,precision-54.35% and recall-53.33% For DT the accuracy score-75.97%,precision-60.0% and recall-53.33% For LR the accuracy score-75.97%,precision-60.9% and recall-53.33% According to accuracy the best fit algorithm is SVC and KN and according to precision the best fit algorithm is KN.Overall KN gives better result as compare to other algorithms.

In [ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score

def evaluate_performance(clfs, X_train, y_train, X_test, y_test):
    results = {}

    for clf_name, clf in clfs.items():
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        confusion = confusion_matrix(y_test, y_pred)
        auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1]) if hasattr(clf, 'predict_proba') else None

        results[clf_name] = {
            'accuracy': accuracy,
            'confusion_matrix': confusion,
            'auc': auc
        }

    return results
```

The code evaluates the performance of multiple classifiers using the evaluate_performance function, which presumably calculates various performance metrics such as accuracy, confusion matrix, and area under the ROC curve (AUC). After evaluating the performance, the code prints out the results for each classifier, including its name, accuracy, confusion matrix, and AUC.

In [ ]:
```
# Evaluate performance
performance_results = evaluate_performance(clfs, X_train, y_train, X_test, y_test)

# Print results
for clf_name, metrics in performance_results.items():
    print(f"Classifier: {clf_name}")
    print(f"Accuracy: {metrics['accuracy']}")
    print(f"Confusion Matrix:\n{metrics['confusion_matrix']}")
    print(f"AUC: {metrics['auc']}")
    print("-------------------------")
```

```
Classifier: SVC
Accuracy: 0.7662337662337663
Confusion Matrix:
[[93 16]
 [20 25]]
AUC: None
-------------------------
Classifier: KN
Accuracy: 0.7662337662337663
Confusion Matrix:
[[92 17]
 [19 26]]
AUC: 0.7984709480122324
-------------------------
Classifier: NB
Accuracy: 0.7337662337662337
Confusion Matrix:
[[88 21]
 [20 25]]
AUC: 0.7908256880733944
-------------------------
Classifier: DT
Accuracy: 0.7532467532467533
Confusion Matrix:
[[92 17]
 [21 24]]
AUC: 0.7906218144750254
-------------------------
Classifier: LR
Accuracy: 0.7597402597402597
Confusion Matrix:
[[93 16]
 [21 24]]
AUC: 0.8167176350662589
-------------------------
```

A comparative analysis of the performance of different classifiers on the given dataset. It provides insights into which classifier performs better in terms of accuracy, AUC, and how it's making predictions based on the confusion matrix. This information is crucial for selecting the most suitable classifier for the task at hand and understanding its strengths and weaknesses.

CONCLUSION :

In this study, we applied several machine learning algorithms to predict diabetes outcomes based on a dataset of relevant features. The classifiers evaluated include Support Vector Classifier (SVC), K-Nearest Neighbors Classifier (KNC), Gaussian Naive Bayes (GNB), Decision Tree Classifier (DTC), and Logistic Regression (LRC).

Performance evaluation revealed varying degrees of effectiveness among the classifiers. Support Vector Classifier exhibited the highest accuracy of [76.62%], closely followed by K Neighbors Classifier with an accuracy of [76.62%]. However, Logistic Regression Classifier achieved the highest AUC score of [81.67%], indicating its strong discriminative ability.

Analysis of confusion matrices provided insights into the classifiers' predictive capabilities, highlighting areas of correct and incorrect predictions. Further tuning and optimization of hyperparameters could enhance the classifiers' performance and generalize better to unseen data.

Overall, this study demonstrates the feasibility of using machine learning algorithms for diabetes prediction, with Support Vector Classifier and Logistic Regression showing promising results. Future work may involve exploring ensemble methods or incorporating additional features to improve predictive accuracy further.

This conclusion provides a summary of the study's findings, including the performance of different classifiers, insights from evaluation metrics, and suggestions for future research directions.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js