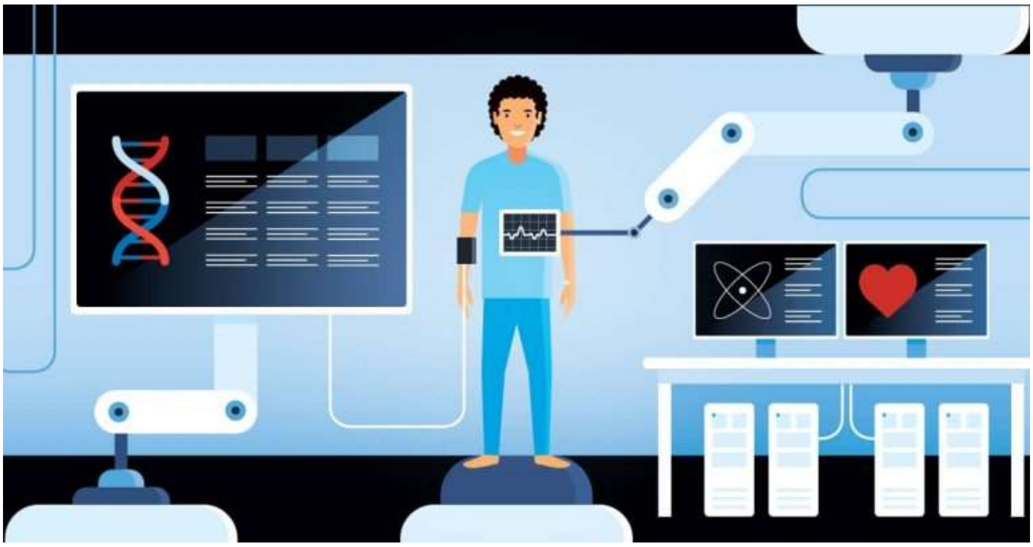
Diabetic Prediction



Dataset Link: https://www.kaggle.com/datasets/aemyjutt/diabetesdataanslysis/data

Import necessary libraries

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

```

Data Preprocessing

```
df=pd.read_csv("/content/diabetes.csv")      #gather data into dataframe
```

```
df.head()                        #display data
```

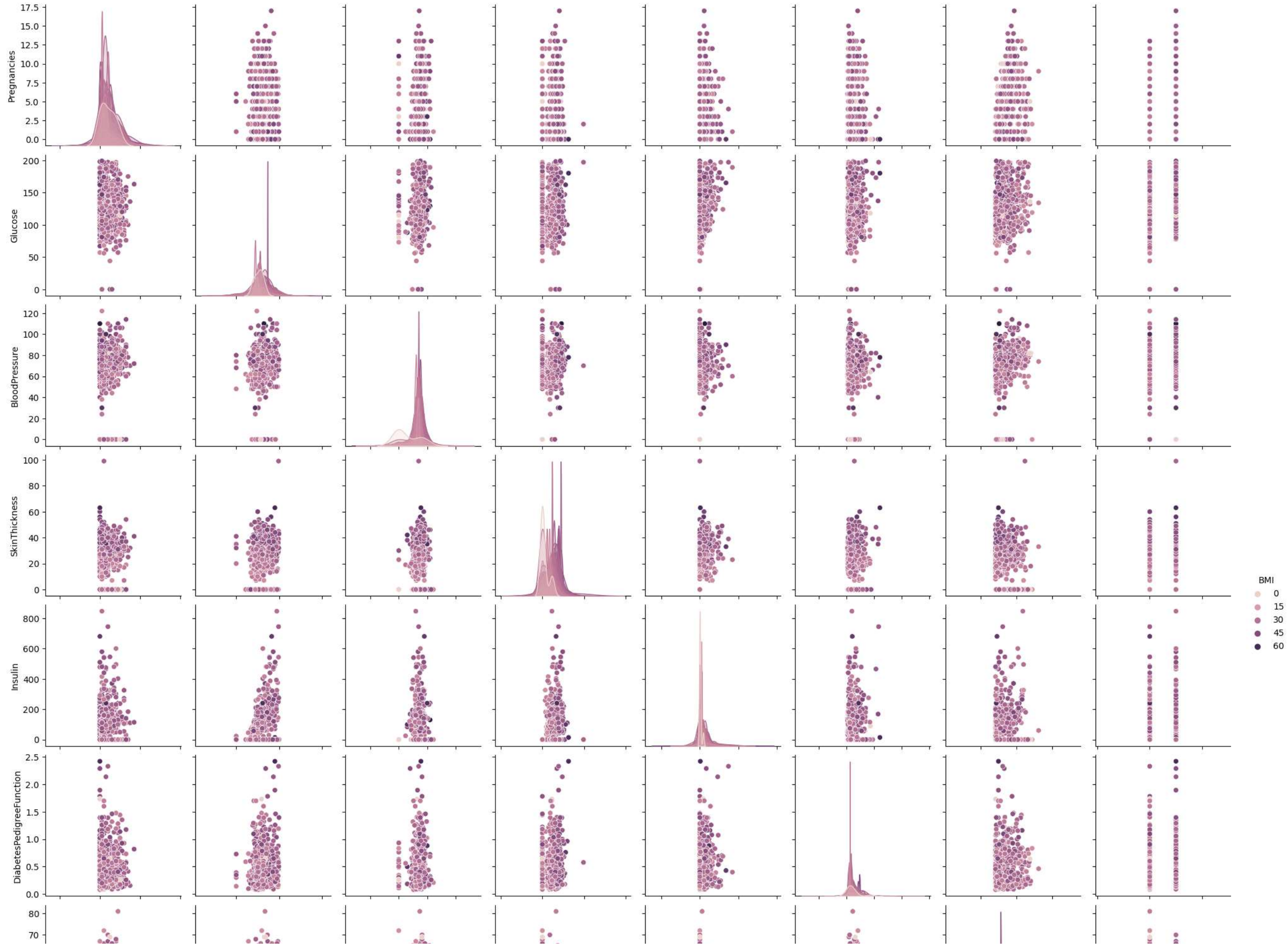
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.shape
```

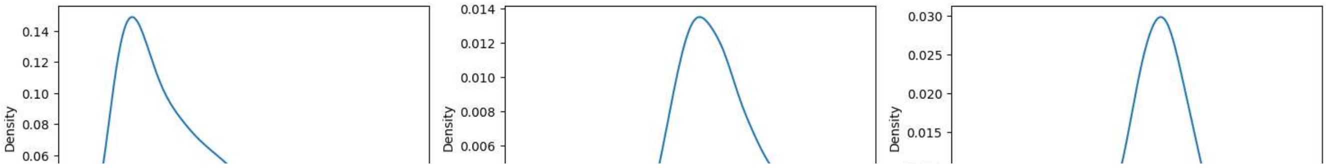
```
(768, 9)
```

```
sns.pairplot(df, hue="BMI")        #display data w.r.t Body mass index
```

```
<seaborn.axisgrid.PairGrid at 0x7c644808da80>
```




```
#disply distribution of data attributes
#we need to draw kernal density estimation to saw 0 values in all features except outcome feature
int_vars = df.select_dtypes(include = ['int','float'])
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
axs = axs.flatten()

for i, var in enumerate (int_vars):
    sns.kdeplot(x=var,data=df,ax=axs[i])

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



there is 0 values in every columns means surely its nan value

```
df.replace({'Pregnancies':0,
            'Glucose':0,
            'BloodPressure':0,
            'SkinThickness':0,
            'Insulin':0,
            'BMI':0,
            'DiabetesPedigreeFunction':0,
            'Age':0},np.nan,inplace=True)
```

```
df.isnull().sum()         #check null values
```

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

```
#fill null value with mode value
#mean and median shrink the shape of ND increase std that why i fill nan values with mode
df['Pregnancies'] = df['Pregnancies'].fillna(df['Pregnancies'].mode()[0])
df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mode()[0])
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mode()[0])
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].mode()[0])
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].mode()[0])
df['BMI'] = df['BMI'].fillna(df['BMI'].mode()[0])
#df.dropna(inplace=True)
```

```
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()
```

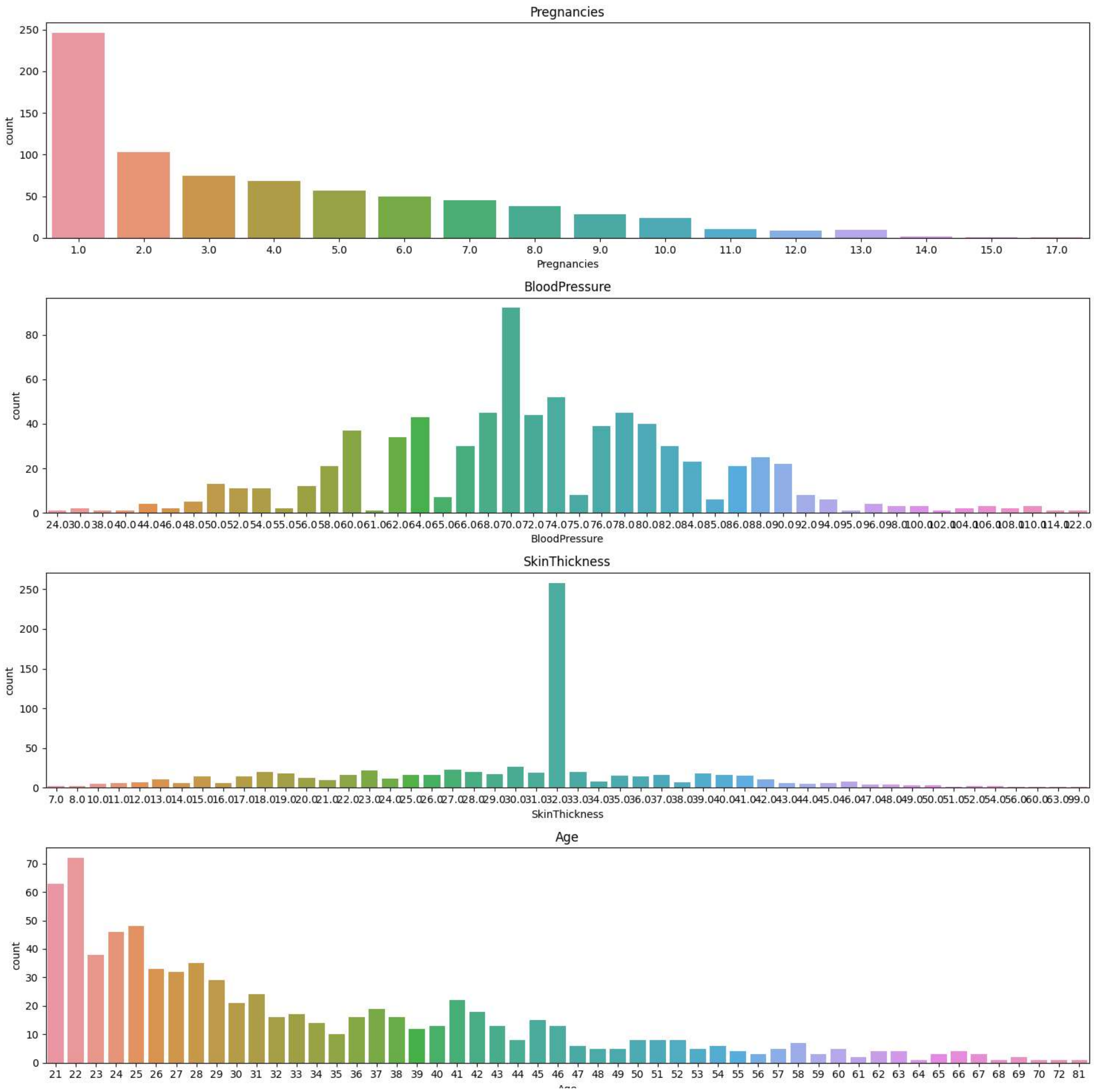
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    float64
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(7), int64(2)
memory usage: 54.1 KB
```

Data visualization

```
df.describe()      #non graphical EDA
```

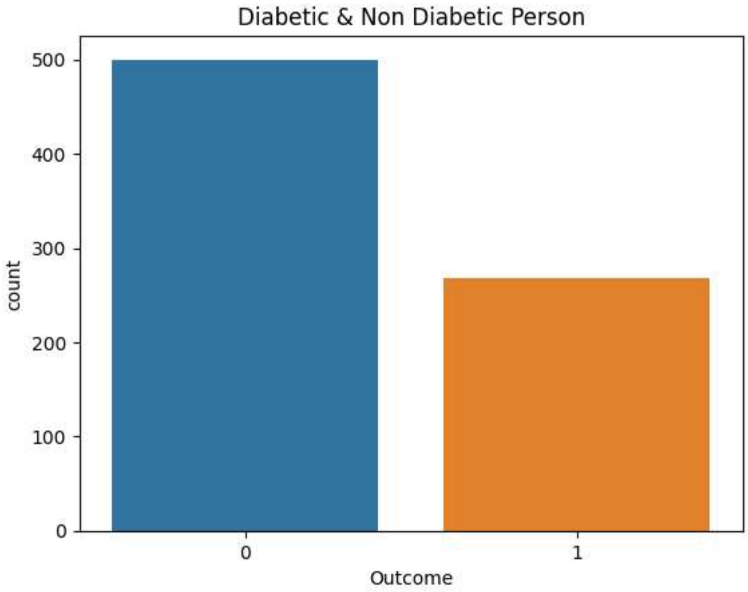
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcom
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.989583	121.539062	72.295573	29.994792	130.932292	32.450911	0.471876	33.240885	0.34895
std	3.219464	30.490660	12.106756	8.886506	88.700443	6.875366	0.331329	11.760232	0.47695
min	1.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.00000
25%	1.000000	99.000000	64.000000	25.000000	105.000000	27.500000	0.243750	24.000000	0.00000
50%	3.000000	117.000000	72.000000	32.000000	105.000000	32.000000	0.372500	29.000000	0.00000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00000

```
#graphical EDA
#get the count of these major features
#univariate analysis
int_vars = ['Pregnancies','BloodPressure','SkinThickness','Age']
fig,axs = plt.subplots(nrows=4, ncols=1, figsize=(15,15))
axs=axs.flatten()
for i, var in enumerate (int_vars):
    sns.countplot(x=var,data=df,ax=axs[i])
    axs[i].set_title(var)

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

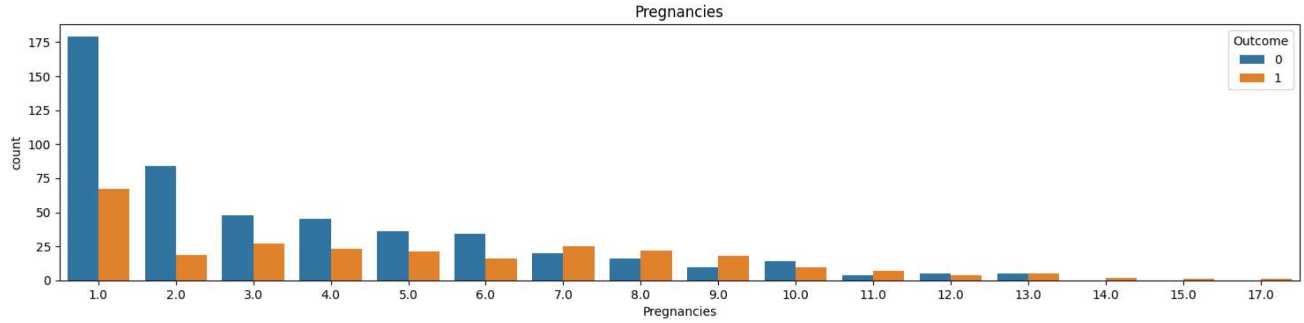


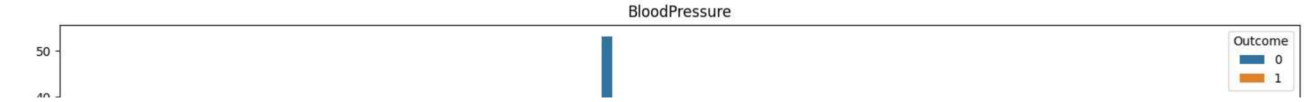
We see the different factor's occurance now let's see the occurance of diabetic & non-diabetic

```
#count the outcome
sns.countplot(x=df['Outcome'],data=df)
plt.title('Diabetic & Non Diabetic Person')
plt.show()
```




```
#get the count of these major features with respect to Outcome
#Multivariate analysis
int_vars = ['Pregnancies','BloodPressure','SkinThickness','Age']
fig,axs = plt.subplots(nrows=4, ncols=1, figsize=(15,15))
axs=axs.flatten()
for i, var in enumerate (int_vars):
    sns.countplot(x=var,hue='Outcome',data=df,ax=axs[i])
    axs[i].set_title(var)

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





with these comparisons, as the age increase the diabetic patient is increase and also if blood pressure inc then there may be chances

```
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    float64
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(7), int64(2)
memory usage: 54.1 KB
```

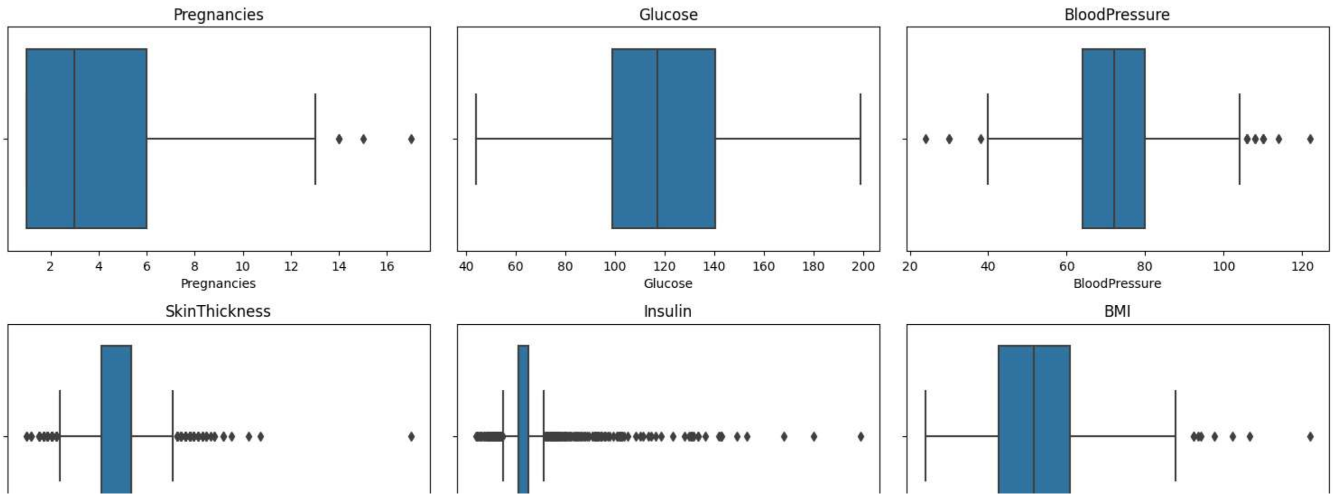
As we see the BMI & DiabetesPedigreeFunction is in float with this we are detecting the outliers

```
#some value are not in int so to get all type same we are converted the features into int
#df['Pregnancies'] = df['Pregnancies'].astype('int')
df['Glucose'] = df['Glucose'].astype('int')
df['SkinThickness'] = df['SkinThickness'].astype('int')
df['Insulin'] = df['Insulin'].astype('int')
df['BloodPressure'] = df['BloodPressure'].astype('int')
```

Outlier detection

Age

```

#Outlier Detection
#Boxplot is help to clearly see the outliers
int_vars = df.select_dtypes(include = ['int','float'])
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
axs = axs.flatten()
for i, var in enumerate (int_vars):
    sns.boxplot(x=var,data=df,ax=axs[i])
    axs[i].set_title(var)

plt.tight_layout()
plt.show()

```

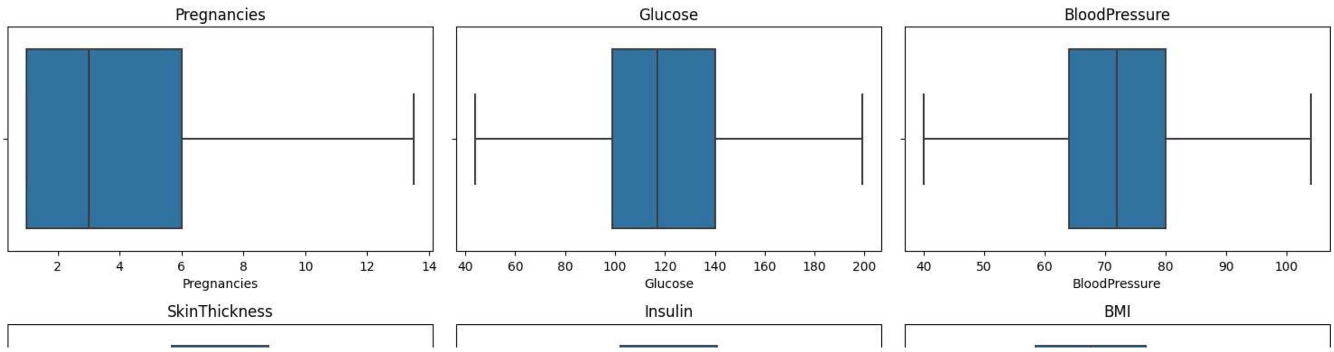


there are many no. of outliers in BMI, Pregnancies, BloodPressure, SkinThickness, Insulin, DiabetesPedigreeFunction, Age

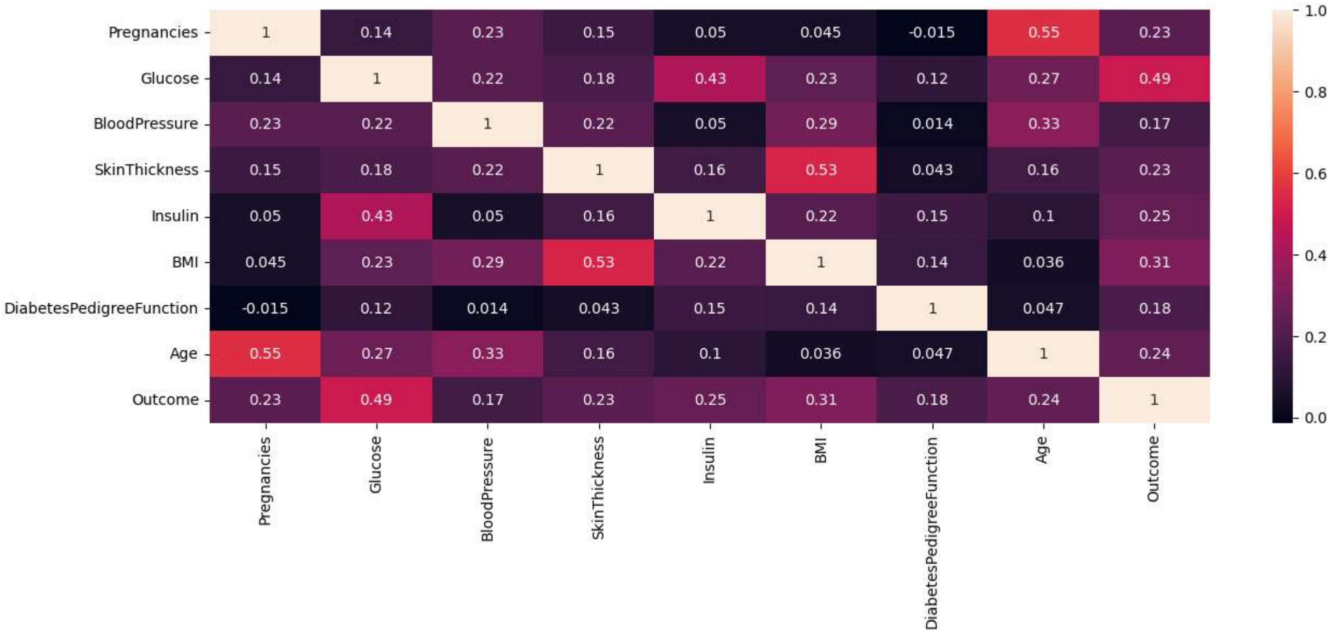
```
fea_outlier=['BMI', 'Pregnancies', 'BloodPressure', 'SkinThickness', 'Insulin', 'DiabetesPedigreeFunction', 'Age']
```

```
#remove outliers
#calculate the Q1, Q2, Q3 ... if data is not in inter quartile range then it eliminate
#make user defined function
def remove_outliers_iqr(data):
    # Calculate the first quartile (Q1) and third quartile (Q3)
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    # Calculate the interquartile range (IQR)
    IQR = Q3 - Q1
    # Define the lower and upper bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove outliers
    data = np.where(data>upper_bound, upper_bound, np.where(data<lower_bound,lower_bound,data))

    return data[(data >= lower_bound) & (data <= upper_bound)]

#apply for those coulmns which contain outliers
for col in fea_outlier:
  df[col]= remove_outliers_iqr(df[col])
```

```
#now again see the any outlier is remain or not
int_vars = df.select_dtypes(include = ['int','float'])
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
axs = axs.flatten()
for i, var in enumerate (int_vars):
    sns.boxplot(x=var,data=df,ax=axs[i])
    axs[i].set_title(var)

plt.tight_layout()
plt.show()

```




```
#correlation matrix
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Clealry see that all the outliers are remove

Data Modeling

```
correlation_matrix = df.corr()
print(correlation_matrix['Outcome'].abs().sort_values(ascending=False))

```

```
Outcome                     1.000000
Glucose                     0.491524
BMI                         0.313030
Insulin                     0.254564
Age                         0.242702
Pregnancies                 0.230182
SkinThickness               0.225530
DiabetesPedigreeFunction    0.184969
BloodPressure               0.167055
Name: Outcome, dtype: float64
```

Encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in df.columns:
  if df[col].dtypes == 'object':
    df[col] = le.fit_transform(df[col])
```

Data splitting

```
x=df.drop('Outcome',axis=1)     #Independent variable
y=df['Outcome']                 #Dependent variable
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.33,random_state=20)
```

```
# scaling data
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()
x_train = scale.fit_transform(x_train)
x_test = scale.transform(x_test)
print('Shape of x_train:', x_train.shape)
print('Shape of x_test:',x_test.shape)
```

```
Shape of x_train: (514, 8)
Shape of x_test: (254, 8)
```

Model Building

We use different algo to get more best results

Logistic Regression : It is widely used for this type of classification & give best results.

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
yL_train_pred=lr.predict(x_train)
yL_test_pred=lr.predict(x_test)
```

```
#check accuracy
from sklearn.metrics import accuracy_score
print("Accuracy : ",accuracy_score(y_test,yL_test_pred))
```

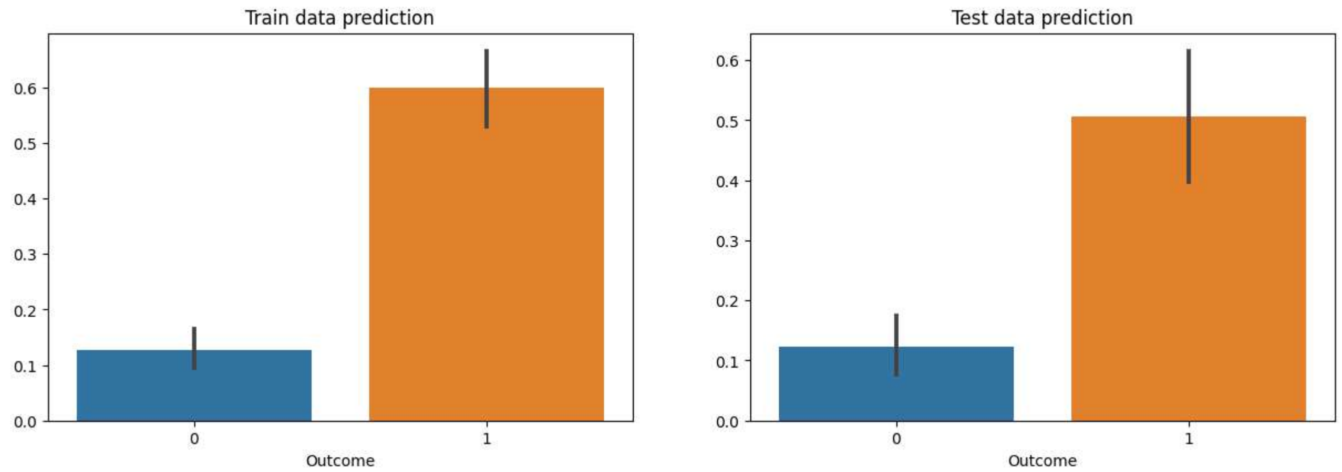
```
Accuracy :  0.7559055118110236
```

```
# importing evaluation metrices
from sklearn.metrics import r2_score, mean_squared_error, classification_report, confusion_matrix

print('Train data results')
print('RMSE:', np.sqrt(mean_squared_error(yL_train_pred, y_train)))
print('R-squared:',r2_score(yL_train_pred,y_train))
print('Test data results')
print('RMSE:', np.sqrt(mean_squared_error(yL_test_pred, y_test)))
print('R-squared:',r2_score(yL_test_pred, y_test))
print("\n Classification Report:\n",classification_report(y_test,yL_test_pred))
print("Confusion Matrix : \n", confusion_matrix(y_test,yL_test_pred))
```

```
Train data results
RMSE: 0.4750588740905069
R-squared: -0.07949957452971956
Test data results
RMSE: 0.4940591950252281
R-squared: -0.30873431396991613

 Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.88      0.83       171
           1       0.67      0.51      0.58        83

    accuracy                           0.76       254
   macro avg       0.73      0.69      0.70       254
weighted avg       0.75      0.76      0.75       254

Confusion Matrix :
 [[150  21]
 [ 41  42]]
```

```
# plotting results for visual
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
plt.title('Train data prediction')
sns.barplot(x=y_train,y=yL_train_pred)
plt.subplot(2,2,2)
plt.title('Test data prediction')
```

```
sns.barplot(x=y_test,y=yL_test_pred)
plt.show()
```



Random Forest : It combine multible decision tree and also it is an ensemble technique

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(x_train, y_train)
yR_train_pred=RF.predict(x_train)
yR_test_pred=RF.predict(x_test)
```

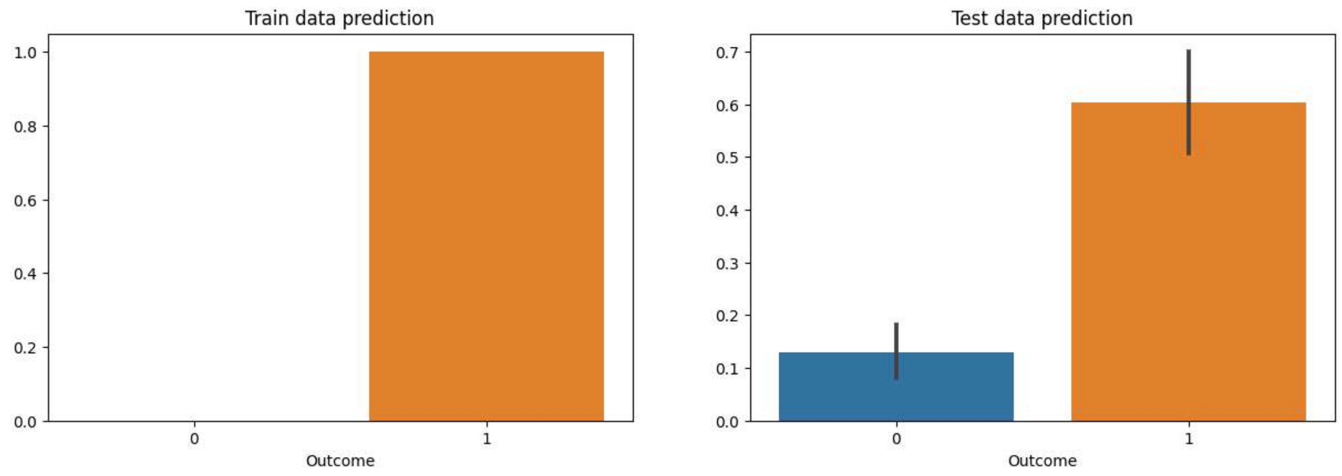
```
#check accuracy
print("Accuracy : ",accuracy_score(y_test,yR_test_pred))
```

```
Accuracy :  0.7834645669291339
```

```
#Evalution metrics
print('Train data results')
print('RMSE:', np.sqrt(mean_squared_error(yR_train_pred, y_train)))
print('R-squared:',r2_score(yR_train_pred,y_train))
print('Test data results')
print('RMSE:', np.sqrt(mean_squared_error(yR_test_pred, y_test)))
print('R-squared:',r2_score(yR_test_pred, y_test))
print("\n Classification Report:\n",classification_report(y_test,yR_test_pred))
print("Confusion Matrix : \n", confusion_matrix(y_test,yR_test_pred))
```

```
Train data results
RMSE: 0.0
R-squared: 1.0
Test data results
RMSE: 0.4653336792785003
R-squared: -0.06608669108669085

 Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.87      0.84       171
           1       0.69      0.60      0.65        83

    accuracy                           0.78       254
   macro avg       0.76      0.74      0.74       254
weighted avg       0.78      0.78      0.78       254

Confusion Matrix : 
 [[149  22]
 [ 33  50]]
```

```
# plotting results for visualization
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
plt.title('Train data prediction')
sns.barplot(x=y_train,y=yR_train_pred)
plt.subplot(2,2,2)
plt.title('Test data prediction')
sns.barplot(x=y_test,y=yR_test_pred)
plt.show()
```



Support Vector Machine : This classify the elements on the bases of hyperplane & with good margin

```
from sklearn.svm import SVC
sv = SVC()
sv.fit(x_train, y_train)
yS_train_pred=sv.predict(x_train)
yS_test_pred=sv.predict(x_test)
```

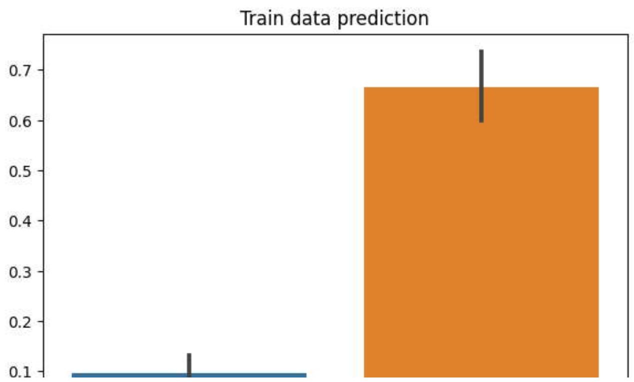
```
#check accuracy
print("Accuracy : ",accuracy_score(y_test,yS_test_pred))
```

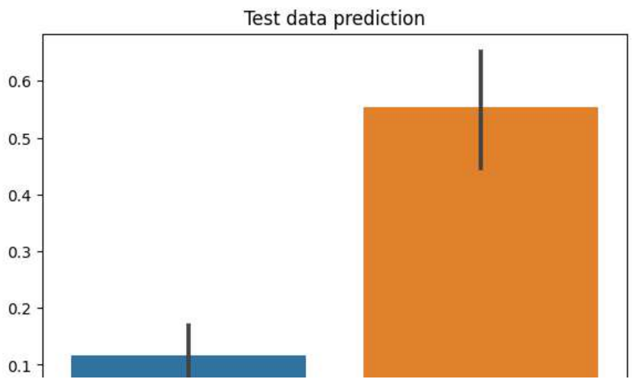
```
Accuracy :  0.7755905511811023
```

```
#Evalution metrics
print('Train data results')
print('RMSE:', np.sqrt(mean_squared_error(yS_train_pred, y_train)))
print('R-squared:',r2_score(yS_train_pred,y_train))
print('Test data results')
print('RMSE:', np.sqrt(mean_squared_error(yS_test_pred, y_test)))
print('R-squared:',r2_score(yS_test_pred, y_test))
print("\n Classification Report:\n",classification_report(y_test,yS_test_pred))
print("Confusion Matrix : \n", confusion_matrix(y_test,yS_test_pred))

```

```
Train data results
RMSE: 0.42764398444489615
R-squared: 0.13170994698535354
Test data results
RMSE: 0.47371874442426026
R-squared: -0.1668278529980658

 Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.88      0.84       171
           1       0.70      0.55      0.62        83

    accuracy                           0.78       254
   macro avg       0.75      0.72      0.73       254
weighted avg       0.77      0.78      0.77       254

Confusion Matrix :
 [[151  20]
 [ 37  46]]
```

```
# plotting results for visualization
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
plt.title('Train data prediction')
sns.barplot(x=y_train,y=yS_train_pred)
plt.subplot(2,2,2)
plt.title('Test data prediction')
sns.barplot(x=y_test,y=yS_test_pred)
plt.show()

```





binary classification using NN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

# Initialize the model
model = Sequential()

# Add input layer and first hidden layer
model.add(Dense(units=128, activation='relu', input_dim=x_train.shape[1]))
model.add(Dropout(0.2))  # Dropout layer with a dropout rate of 0.2
# Add more hidden layers if needed
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.2))  # Dropout layer with a dropout rate of 0.2
# Add the output layer with sigmoid activation for binary classification
model.add(Dense(units=1, activation='sigmoid'))
model.add(Dropout(0.2))  # Dropout layer with a dropout rate of 0.2
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
#train
# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=32, validation_data=(x_test, y_test))
yNN_train_pred=model.predict(x_train)
yNN_test_pred=model.predict(x_test)
```

```
17/17 [==============================] - 0s 4ms/step - loss: 1.4493 - accuracy: 0.7451 - val_loss: 0.4664 - val_accuracy: 0.7638
Epoch 96/100
17/17 [==============================] - 0s 5ms/step - loss: 1.4067 - accuracy: 0.7588 - val_loss: 0.4622 - val_accuracy: 0.7638
Epoch 97/100
17/17 [==============================] - 0s 5ms/step - loss: 1.1311 - accuracy: 0.7957 - val_loss: 0.4676 - val_accuracy: 0.7559
Epoch 98/100
17/17 [==============================] - 0s 4ms/step - loss: 1.6362 - accuracy: 0.7490 - val_loss: 0.4644 - val_accuracy: 0.7559
Epoch 99/100
17/17 [==============================] - 0s 5ms/step - loss: 1.3002 - accuracy: 0.7821 - val_loss: 0.4644 - val_accuracy: 0.7520
Epoch 100/100
17/17 [==============================] - 0s 5ms/step - loss: 1.2291 - accuracy: 0.7938 - val_loss: 0.4584 - val_accuracy: 0.7598
17/17 [==============================] - 0s 2ms/step
8/8 [==============================] - 0s 2ms/step
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
#accuracy_score(yNN_test_pred,y_test)
```

```
8/8 [==============================] - 0s 2ms/step - loss: 0.4584 - accuracy: 0.7598
Test Loss: 0.458400160074234
Test Accuracy: 0.7598425149917603
```

```
#Evalution metrics
print('Train data results')
print('RMSE:', np.sqrt(mean_squared_error(yNN_train_pred, y_train)))
print('R-squared:',r2_score(yNN_train_pred,y_train))
print('Test data results')
print('RMSE:', np.sqrt(mean_squared_error(yNN_test_pred, y_test)))
print('R-squared:',r2_score(yNN_test_pred, y_test))

#We can't generate the classification report & confusion matrix becuase in that Classification metrics can't handle a mix of binary a
```

```
Train data results
RMSE: 0.3610654030293338
R-squared: -0.9307514933469379
Test data results
RMSE: 0.3879549954096411
R-squared: -1.4621144782383078
```

Stacking: Stacking combines predictions from multiple models by training a meta-model on top of them. It uses the predictions of base models as features for the meta-model.

```
from sklearn.ensemble import StackingClassifier
stacking_classifier = StackingClassifier(estimators=[(RF,sv)], final_estimator=lr)
stacking_classifier.fit(x_train, y_train)
ySC_train_pred=stacking_classifier.predict(x_train)
ySC_test_pred=stacking_classifier.predict(x_test)
print("Accuracy : ",accuracy_score(y_test,ySC_test_pred))
```

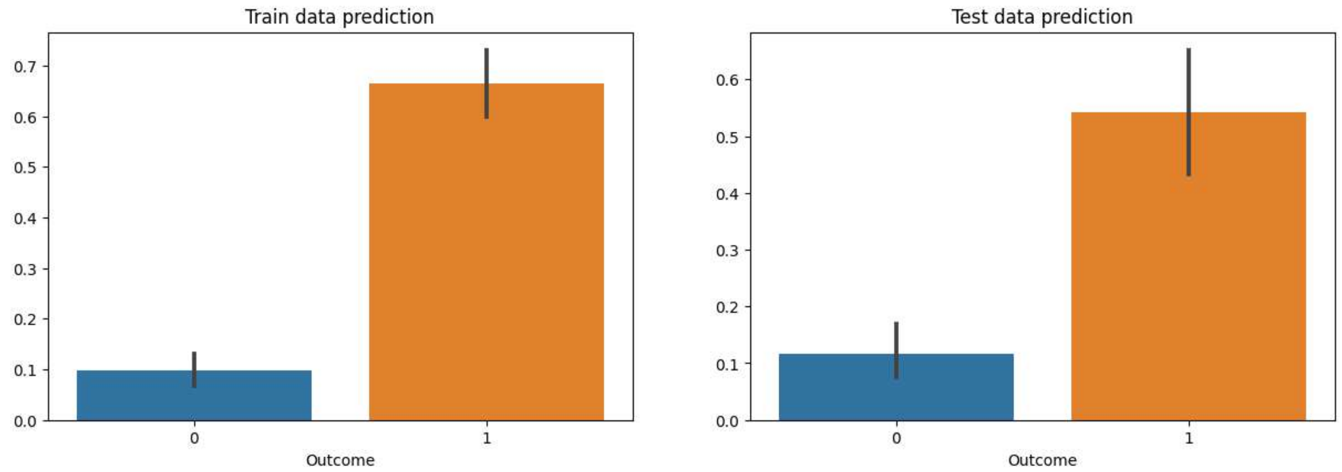
```
Accuracy :  0.7716535433070866
```

```
#Evalution metrics
print('Train data results')
print('RMSE:', np.sqrt(mean_squared_error(ySC_train_pred, y_train)))
print('R-squared:',r2_score(ySC_train_pred,y_train))
print('Test data results')
print('RMSE:', np.sqrt(mean_squared_error(ySC_test_pred, y_test)))
print('R-squared:',r2_score(ySC_test_pred, y_test))
print("\n Classification Report:\n",classification_report(y_test,ySC_test_pred))
print("Confusion Matrix : \n", confusion_matrix(y_test,ySC_test_pred))

```

```
Train data results
RMSE: 0.42764398444489615
R-squared: 0.13170994698535354
Test data results
RMSE: 0.4778561045889373
R-squared: -0.19918599918599922

 Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.88      0.84       171
           1       0.69      0.54      0.61        83

    accuracy                           0.77       254
   macro avg       0.75      0.71      0.72       254
weighted avg       0.76      0.77      0.76       254
```

```
Confusion Matrix :
 [[151  20]
 [ 38  45]]
```

```
# plotting results for visualization
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
plt.title('Train data prediction')
sns.barplot(x=y_train,y=ySC_train_pred)
plt.subplot(2,2,2)
plt.title('Test data prediction')
sns.barplot(x=y_test,y=ySC_test_pred)
plt.show()

```

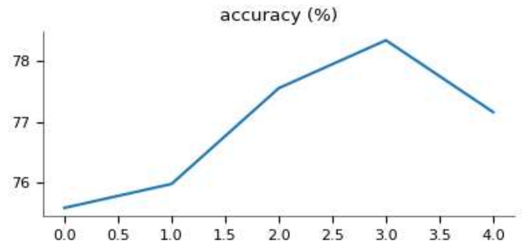


Final Report

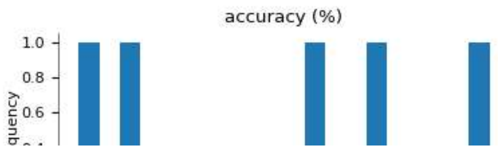
```
from sklearn.metrics import r2_score, mean_squared_error
data = [['Logistic_Regression',accuracy_score(y_test,yL_test_pred)*100],
        ['Artificial_Neural_Network',accuracy*100],
        ['SVM',accuracy_score(y_test,yS_test_pred)*100],
        ['Random_Forest',accuracy_score(y_test,yR_test_pred)*100],
        ['Stacking',accuracy_score(y_test,ySC_test_pred)*100]]
result = pd.DataFrame(data, columns=['algo','accuracy (%)'])
result
```

	algo	accuracy (%)
0	Logistic_Regression	75.590551
1	Artificial_Neural_Network	75.984251
2	SVM	77.559055
3	Random_Forest	78.346457
4	Stacking	77.165354

Values



Distributions



All the model appear to be almost same accuracy

But Stacking, SVM & random forest are pretty good because as compare to others it have less no. of False negative

Categorical distributions

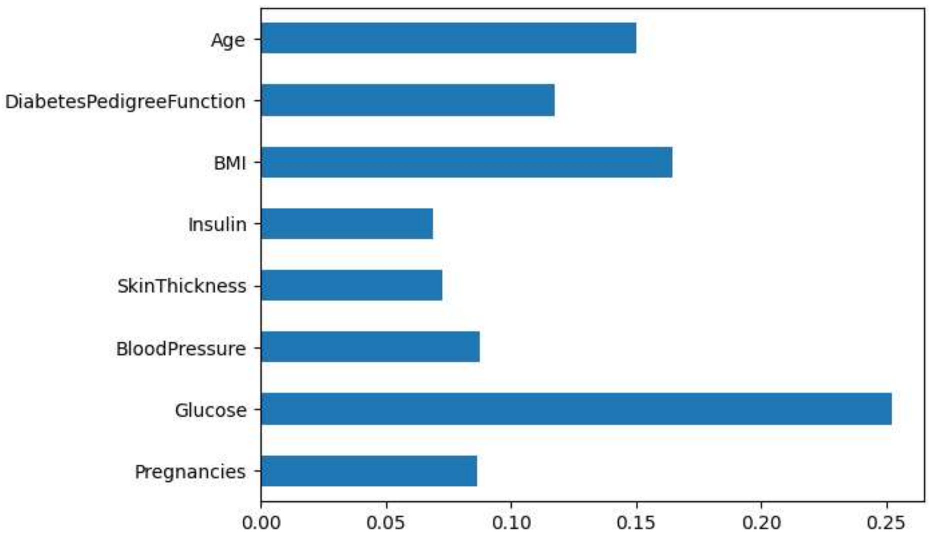
Feature Importance

As we getting the approx same accuracy level so Knowing about the feature importance is quite necessary as it shows that how much weightage each feature provides in the model building phase.

Getting feature importances

```
#through bar graph visualize the important feature in whole model
(pd.Series(RF.feature_importances_, index=x.columns).plot(kind='barh'))
```

```
<Axes: >
```



Glucose is the most imp feature as per data to detect the diabetic patient

Saving Model :- Stacking

```
import pickle     #pickle module in Python is used for serializing and deserializing Python objects. And  is useful when you want to
save_model = pickle.dumps(stacking_classifier)
```

```
#Then we will be loading that saved model
SC_from_pickle = pickle.loads(save_model)

# lastly, after loading that model we will use this to make predictions
SC_from_pickle.predict(x_test)
```

```
array([0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0])
```

TESTING THE OUTCOME

```
df.head(7)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6.0	148	72.0	35.0	105.000	33.6	0.627	50.0	1
1	1.0	85	66.0	29.0	105.000	26.6	0.351	31.0	0
2	8.0	183	64.0	32.0	105.000	23.3	0.672	32.0	1
3	1.0	89	66.0	23.0	94.000	28.1	0.167	21.0	0
4	1.0	137	40.0	35.0	160.625	43.1	1.200	33.0	1
5	5.0	116	74.0	32.0	105.000	25.6	0.201	30.0	0
6	3.0	78	50.0	32.0	88.000	31.0	0.248	26.0	1

```
#Detect either you are diabetic patient or not
stacking_classifier.predict([[3.0,78, 50.0, 32.0, 88.000, 31.0, 0.248,  26.0  ]])
```

```
array([0])
```

Result of stacking might be somewhat wrong

Because in dataset its positive result but model show negative

```
from sklearn.preprocessing import StandardScaler
input=(5,180,100,60,570,25.5,0.128,40)
input2array=np.asarray(input)
input_reshape=input2array.reshape(1,-1)
sd_data=scale.transform(input_reshape)
prediction=RF.predict(sd_data)
prediction
if prediction == 0:
  print("Patient is not diabetic ")
else:
  print("Patient is diabetic ")

```

```
Patient is diabetic
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler was
  warnings.warn(
```

But Random Forest detects the accurate result So that we can reach to both Random Forest as best technique for this diabetic prediction analysis