Diabetes_Prediciton_Model_Code_final

August 19, 2025

1 DIABETES PREDICTION :-

1.1 Importing required libraries

[1]:
```
import pandas as pd
#pandas for loading data
import matplotlib.pyplot as plt
import seaborn as sns
# matplot and sns for data visualization
```

1.2 Loading data set

[2]:
```
# we use read_csv fuction to load data
df = pd.read_csv(r"C:\Users\sonuk\OneDrive\Documents\diabetes.csv")
print(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
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
..                        …  …      …
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

[768 rows x 9 columns]
```

2 Exploring the dataset :-

[3]:
```
# exploring number of rows and columns of the dataset
df.shape
```

[3]:
```
(768, 9)
```

[4]:
```
# returning an object with all of the column headers
df.columns
```

[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

[5]:
```
# getting data types of each column
df.dtypes
```

[5]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

[6]:
```
# getting basic information about all the columns
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
```

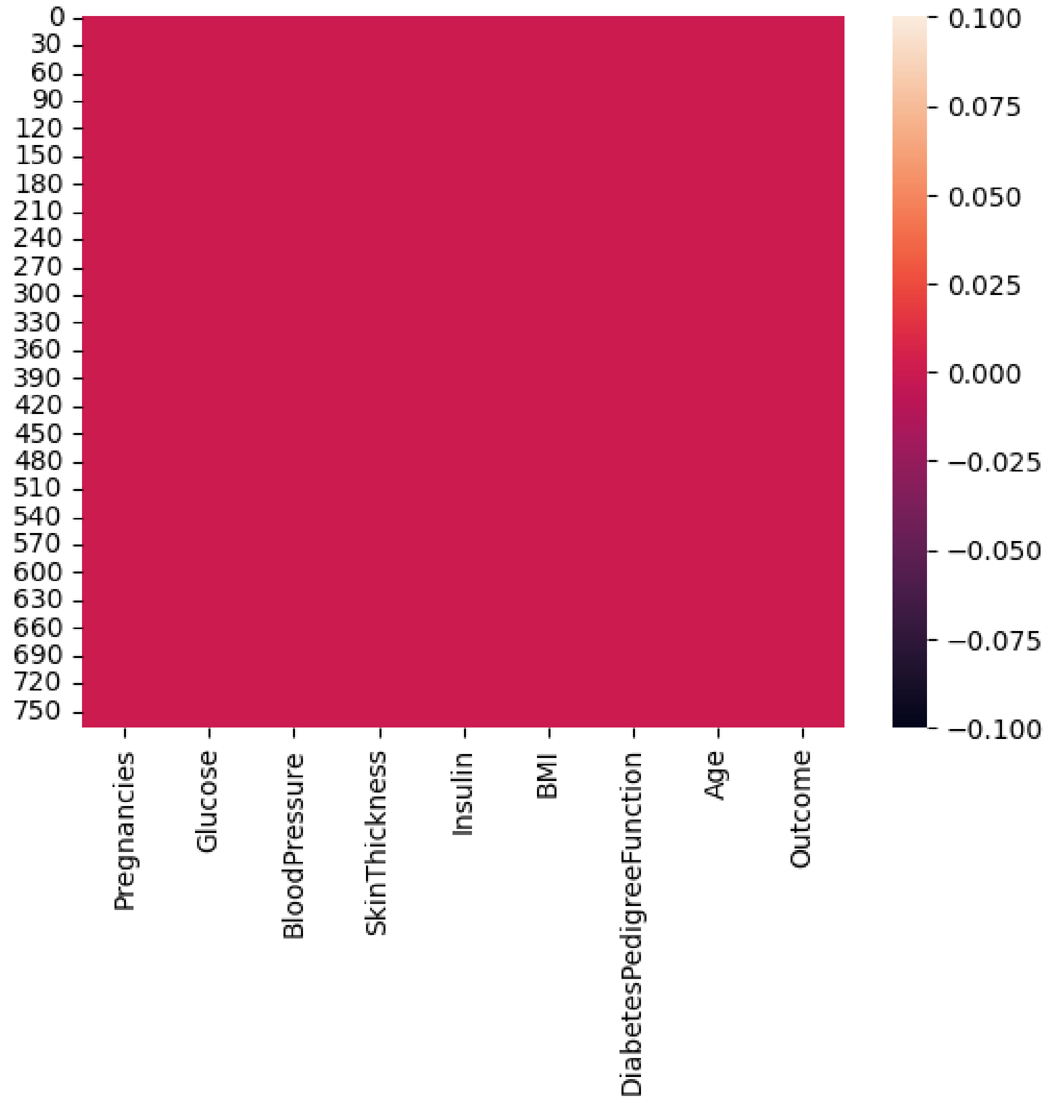
2.1 Check for null value / Missing data

[7]:
```
# checking if there are any null values in dataset
df.isnull().sum()
```

[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[8]:
```
sns.heatmap(df.isnull())
```

[8]:
```
<Axes: >
```



3 Observations

1. There are 9 features in the dataset.
2. Each feature is either of integer or float datatype.
3. There are zero NaN values in the dataset.
4. Map is blank so there is no null value here
5. In the outcome column, 1 represents that person has diabetes and 0 represents person does'nt have diabetes

4 Data Visualization

[9]:
```
# Importing essential libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

[10]:
```
df = df.rename(columns={'DiabetesPedigreeFunction':'DPF'})
df.head()
```

[10]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI    DPF  \
0            6      148             72             35        0  33.6  0.627
1            1       85             66             29        0  26.6  0.351
2            8      183             64              0        0  23.3  0.672
3            1       89             66             23       94  28.1  0.167
4            0      137             40             35      168  43.1  2.288

   Age  Outcome
0   50        1
1   31        0
2   32        1
3   21        0
4   33        1
```

[11]:
```
list1 = df.columns[:-1]
list1
```

[11]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DPF', 'Age'],
      dtype='object')
```

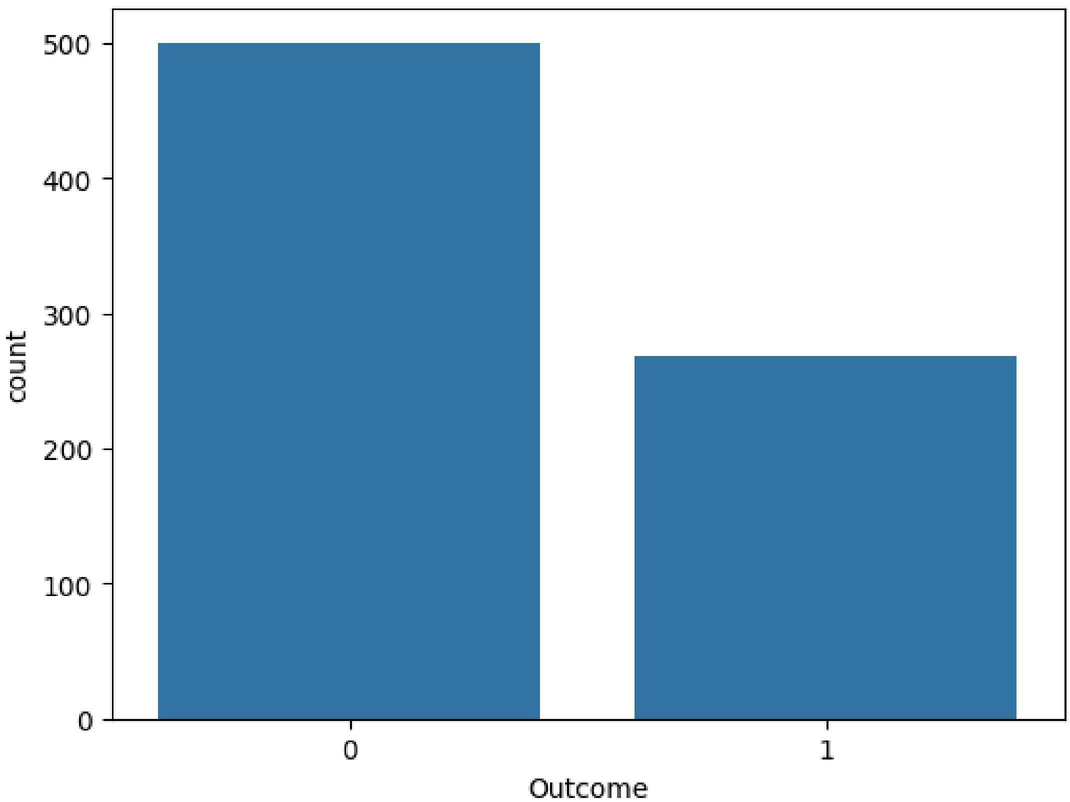
[12]:
```
# return basic statistics
df.describe()
```

[12]:
```
       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         DPF         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
```

[13]:
```
# Outcome countplot
sns.countplot(x='Outcome', data = df)
print(df.Outcome.value_counts())
```

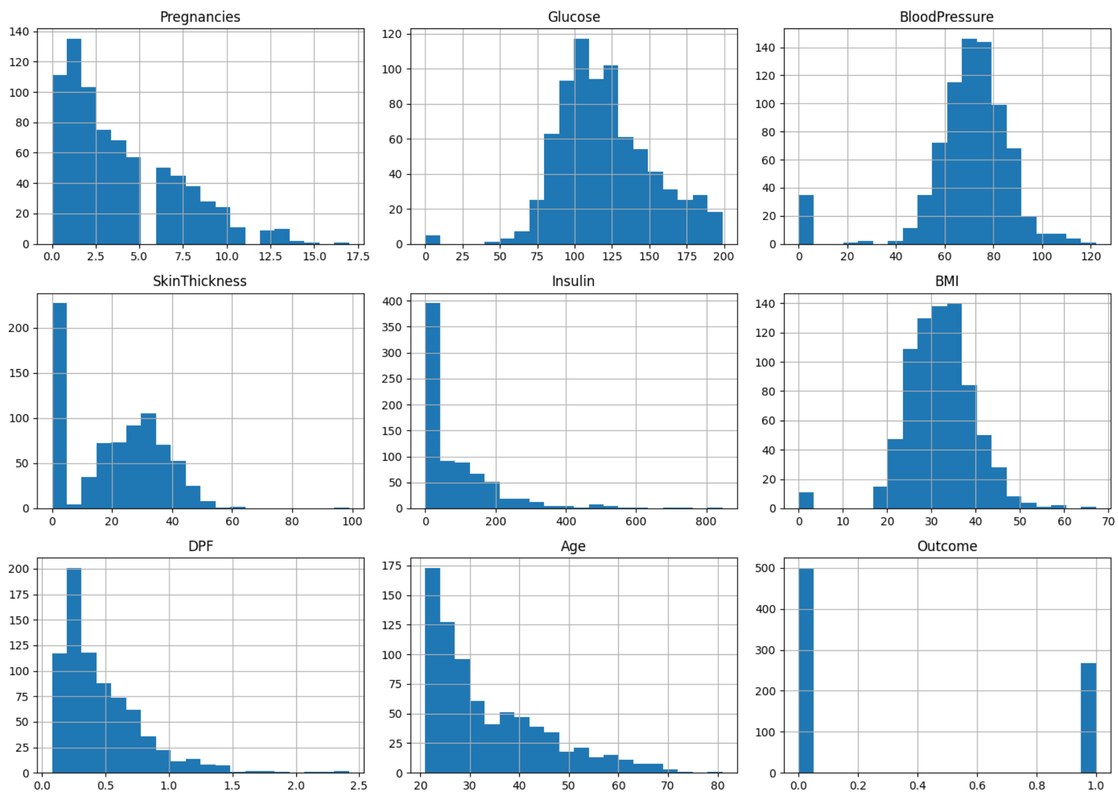
```
Outcome
0    500
1    268
Name: count, dtype: int64
```



[14]:
```
# df.hist(...) → Plots histogram for each numeric column in the DataFrame.

# bins=20 → Divides data into 20 intervals (controls smoothness of histogram).

# figsize=(14,10) → Makes the figure larger.

# plt.tight_layout() → Adjusts spacing so plots don't overlap.
```

```
# plt.show() → Displays the plots.
```

[15]:
```
# Plotting histogram of each feature
df.hist(bins=20, figsize=(14, 10))
plt.tight_layout()
plt.show()
```



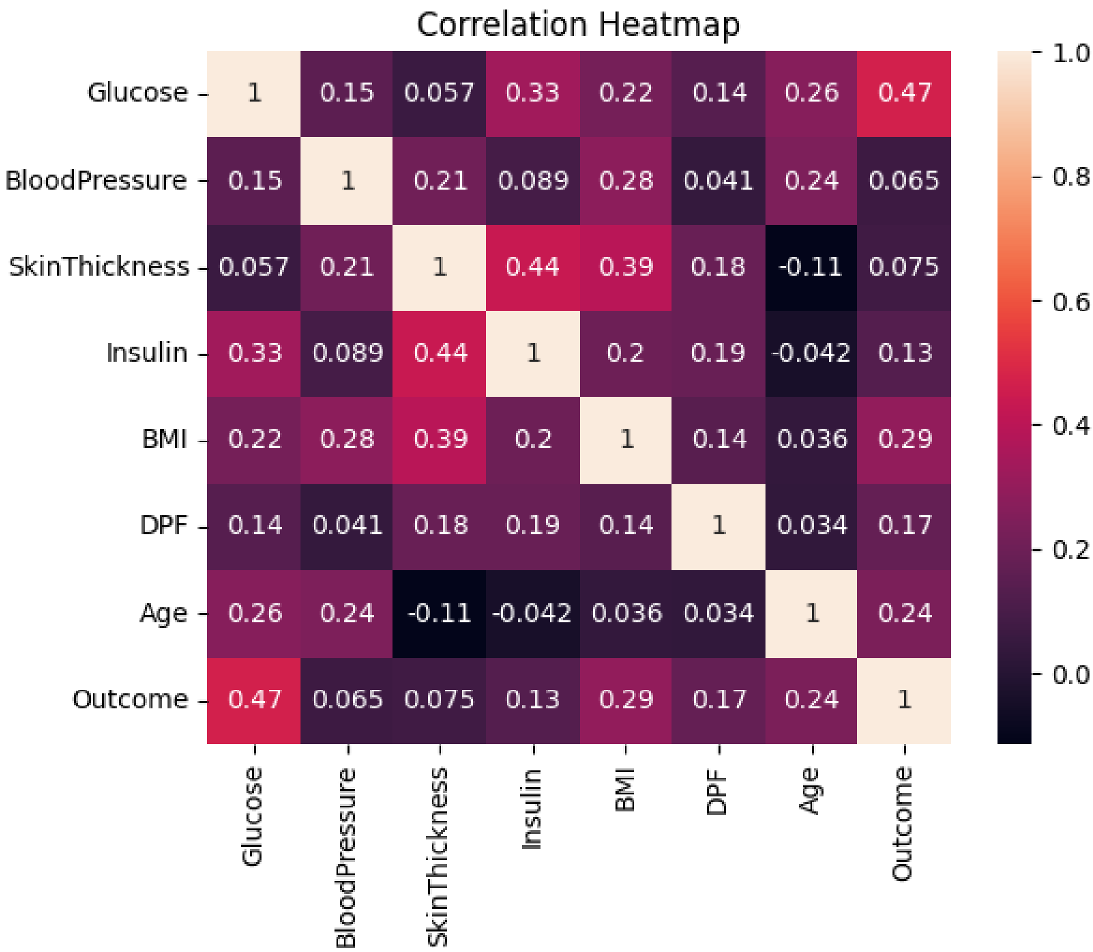
4.1 Co-relation matrix

[16]:
```
# What is Correlation?

# Correlation measures how strongly two variables are related.

# Value ranges between -1 and +1:

# +1 → Perfect positive relation (when one increases, the other also increases).

# -1 → Perfect negative relation (when one increases, the other decreases).

# 0 → No linear relationship.
```

[17]:
```
correlation =␣
  ↪df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DPF','Age','Outcome']].
  ↪corr()
correlation
```

[17]:
```
               Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
Glucose       1.000000       0.152590       0.057328  0.331357  0.221071
BloodPressure 0.152590       1.000000       0.207371  0.088933  0.281805
SkinThickness 0.057328       0.207371       1.000000  0.436783  0.392573
Insulin       0.331357       0.088933       0.436783  1.000000  0.197859
BMI           0.221071       0.281805       0.392573  0.197859  1.000000
DPF           0.137337       0.041265       0.183928  0.185071  0.140647
Age           0.263514       0.239528      -0.113970 -0.042163  0.036242
Outcome       0.466581       0.065068       0.074752  0.130548  0.292695

                   DPF       Age   Outcome
Glucose       0.137337  0.263514  0.466581
BloodPressure 0.041265  0.239528  0.065068
SkinThickness 0.183928 -0.113970  0.074752
Insulin       0.185071 -0.042163  0.130548
BMI           0.140647  0.036242  0.292695
DPF           1.000000  0.033561  0.173844
Age           0.033561  1.000000  0.238356
Outcome       0.173844  0.238356  1.000000
```

[18]:
```
# plotting heatmap
#plt.figure(figsize=(10, 7))
sns.heatmap(correlation, annot=True)
plt.title('Correlation Heatmap')
plt.show()
```



5 Observations from above plots:

1.Counplot tells us that dataset is imbalanced, as number of patients who don't have diabetes is more than those who do.

2.From the correlation heatmap, we can see that there is a high correlation between Outcome and [Glucose, BMI, Age, Insulin]. We can select these
features/columns to accept the input from user and predict the

6 Data preprocssing

[19]:
```
# Feature scaling using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
#It scales all features to a [0, 1] range.
#Especially useful when the features have varying units or scales (like age vs␣
↪income).
```

```python
# Separate features(X) and target variable(y)
X = df.drop('Outcome', axis=1) # All features except target
y = df['Outcome']      # Target variable

# Initialize scaler
scaler = MinMaxScaler()

# Fit and transform the features
X_scaled = scaler.fit_transform(X)

# Convert back to DataFrame for readability
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

 # First 5 rows of scaled data
print(X_scaled.head())

# Combine scaled features with the target variable for further analysis
data_scaled = pd.concat([X_scaled, y], axis=1)
print(data_scaled.head())
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000  0.500745
1     0.058824  0.427136       0.540984       0.292929  0.000000  0.396423
2     0.470588  0.919598       0.524590       0.000000  0.000000  0.347243
3     0.058824  0.447236       0.540984       0.232323  0.111111  0.418778
4     0.000000  0.688442       0.327869       0.353535  0.198582  0.642325

        DPF       Age
0  0.234415  0.483333
1  0.116567  0.166667
2  0.253629  0.183333
3  0.038002  0.000000
4  0.943638  0.200000
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.352941  0.743719       0.590164       0.353535  0.000000  0.500745
1     0.058824  0.427136       0.540984       0.292929  0.000000  0.396423
2     0.470588  0.919598       0.524590       0.000000  0.000000  0.347243
3     0.058824  0.447236       0.540984       0.232323  0.111111  0.418778
4     0.000000  0.688442       0.327869       0.353535  0.198582  0.642325

        DPF       Age  Outcome
0  0.234415  0.483333        1
1  0.116567  0.166667        0
2  0.253629  0.183333        1
3  0.038002  0.000000        0
4  0.943638  0.200000        1
```

6.1 Model Training

[20]:
```
# X = dataset.iloc[:, :-1].values #Independent Variable
# Y = dataset.iloc[:, -1].values #Dependent Variable

from sklearn.model_selection import train_test_split

X = df.drop(columns='Outcome')
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=0)
# we specify that 0.2 means we split data into two part -> 20% for test set and␣
↪80% for train test

print('X_train size: {}, X_test size: {}'.format(X_train.shape, X_test.shape))
```

```
X_train size: (614, 8), X_test size: (154, 8)
```

[21]:
```
## training data
print(X_train,y_train)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI    DPF  \
603            7      150             78             29      126  35.2  0.692
118            4       97             60             23        0  28.2  0.443
247            0      165             90             33      680  52.3  0.427
157            1      109             56             21      135  25.2  0.833
468            8      120              0              0        0  30.0  0.183
..           …        …            …            …      …    …     …
763           10      101             76             48      180  32.9  0.171
192            7      159             66              0        0  30.4  0.383
629            4       94             65             22        0  24.7  0.148
559           11       85             74              0        0  30.1  0.300
684            5      136             82              0        0   0.0  0.640

     Age
603   54
118   22
247   23
157   23
468   38
..   …
763   63
192   36
629   21
559   35
684   69
```

```
[614 rows x 8 columns] 603    1
118    0
247    0
157    0
468    1
      ..
763    0
192    1
629    0
559    0
684    0
Name: Outcome, Length: 614, dtype: int64
```

[22]:
```
## Testing data
print(X_test,y_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI    DPF  \
661            1      199             76             43        0  42.9  1.394
122            2      107             74             30      100  33.6  0.404
113            4       76             62              0        0  34.0  0.391
14             5      166             72             19      175  25.8  0.587
529            0      111             65              0        0  24.6  0.660
..           …        …              …              …       …     …      …
476            2      105             80             45      191  33.7  0.711
482            4       85             58             22       49  27.8  0.306
230            4      142             86              0        0  44.0  0.645
527            3      116             74             15      105  26.3  0.107
380            1      107             72             30       82  30.8  0.821

     Age
661   22
122   23
113   25
14    51
529   31
..   …
476   29
482   28
230   22
527   24
380   24

[154 rows x 8 columns] 661    1
122    0
113    0
14     1
529    0
      ..
```

```
476    1
482    0
230    1
527    0
380    0
Name: Outcome, Length: 154, dtype: int64
```

[23]:
```
## SelectKBest :- this is a method from sklearn.feature_selection that selects␣
 ↪the top k features from dataset based on scoring function
## f_classif :- This is the scoring function used to evaluate importance of␣
 ↪each feature (It performs Anova F-test, which is used for classification
##               tasks to check how much each features contributes to␣
 ↪separating function)
## k=5 :- this tells it to select the top 5 features based on their F-test␣
 ↪scores.
## (Why use it -> to reduce the number of input features in our dataset which␣
 ↪can improve model performance, reduce overfitting, speed up training)

## StandardScaler() :- it is a Preprocessing tool from sklearn that␣
 ↪standardizes features by removing the mean and scaling to unit variance
```

[24]:
```
from sklearn.feature_selection import SelectKBest, chi2, mutual_info_classif,␣
 ↪f_classif
selector = SelectKBest(f_classif,k=5)
```

[25]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[26]:
```
## GridSearchCV :- for hyperparameter tuning
## shuffleSplit :- A cross-validation strategy that randomly splits the data.

## Three classifiers :- Decision Tree, Random Forest , Support Vector Machine
```

[27]:
```
# Using GridSearchCV to find the best algorithm for this problem
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

[28]:
```
# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'criterion': ['gini', 'entropy']
}

## n_estimators: Number of trees in the forest.
## max_depth: Maximum depth of each tree (None means nodes expand until all␣
↪leaves are pure).
## criterion: Function to measure the quality of a split (gini or entropy).

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,␣
↪n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Score: {grid_search.best_score_:.4f}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 50}
Best Score: 0.7688
```

[29]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, KFold
import pandas as pd

# Define cross-validation strategy
cv = KFold(n_splits=5, shuffle=True, random_state=0)

# Decision Tree
dt = DecisionTreeClassifier(criterion='gini', max_depth=5)
dt_scores = cross_val_score(dt, X_train, y_train, cv=cv)
print("Decision Tree")
print(f"Mean Accuracy: {dt_scores.mean():.4f}")
print(f"Standard Deviation: {dt_scores.std():.4f}\n")

# Random Forest
rf = RandomForestClassifier(n_estimators=50, criterion='gini')
rf_scores = cross_val_score(rf, X_train, y_train, cv=cv)
print("Random Forest")
print(f"Mean Accuracy: {rf_scores.mean():.4f}")
print(f"Standard Deviation: {rf_scores.std():.4f}\n")
```

```
# Support Vector Machine
svm = SVC(C=1, kernel='rbf', gamma='auto')
svm_scores = cross_val_score(svm, X_train, y_train, cv=cv)
print("Support Vector Machine")
print(f"Mean Accuracy: {svm_scores.mean():.4f}")
print(f"Standard Deviation: {svm_scores.std():.4f}\n")
```

```
Decision Tree
Mean Accuracy: 0.6971
Standard Deviation: 0.0210

Random Forest
Mean Accuracy: 0.7346
Standard Deviation: 0.0251

Support Vector Machine
Mean Accuracy: 0.7526
Standard Deviation: 0.0402
```

[30]:
```
# Using cross_val_score for gaining average accuracy
## Decision Tree
from sklearn.model_selection import cross_val_score
scores = cross_val_score(DecisionTreeClassifier(criterion='gini',␣
↪random_state=0), X_train, y_train, cv=5)
print('Average Accuracy (Decision Tree) : {}%'.format(round(sum(scores)*100/
↪len(scores)), 3))

## Random forest
from sklearn.model_selection import cross_val_score
scores = cross_val_score(RandomForestClassifier(n_estimators=80,␣
↪random_state=0), X_train, y_train, cv=5)
print('Average Accuracy (RandomForest) : {}%'.format(round(sum(scores)*100/
↪len(scores)), 3))

## Support Vector Machine
from sklearn.model_selection import cross_val_score
scores = cross_val_score(SVC(gamma='auto', random_state=0), X_train, y_train,␣
↪cv=5)
print('Average Accuracy (Support Vector Machine) : {}%'.
↪format(round(sum(scores)*100/len(scores)), 3))
```

```
Average Accuracy (Decision Tree) : 68%
Average Accuracy (RandomForest) : 76%
Average Accuracy (Support Vector Machine) : 74%
```

6.2 CREATING MODEL FOR EACH CLASSIFIER

6.2.1 For Decision Tree

[31]:
```
# Creating Decision Tree Model
from sklearn.tree import DecisionTreeClassifier
classifier1 = DecisionTreeClassifier(criterion='gini', max_depth=5)
classifier1.fit(X_train, y_train)
```

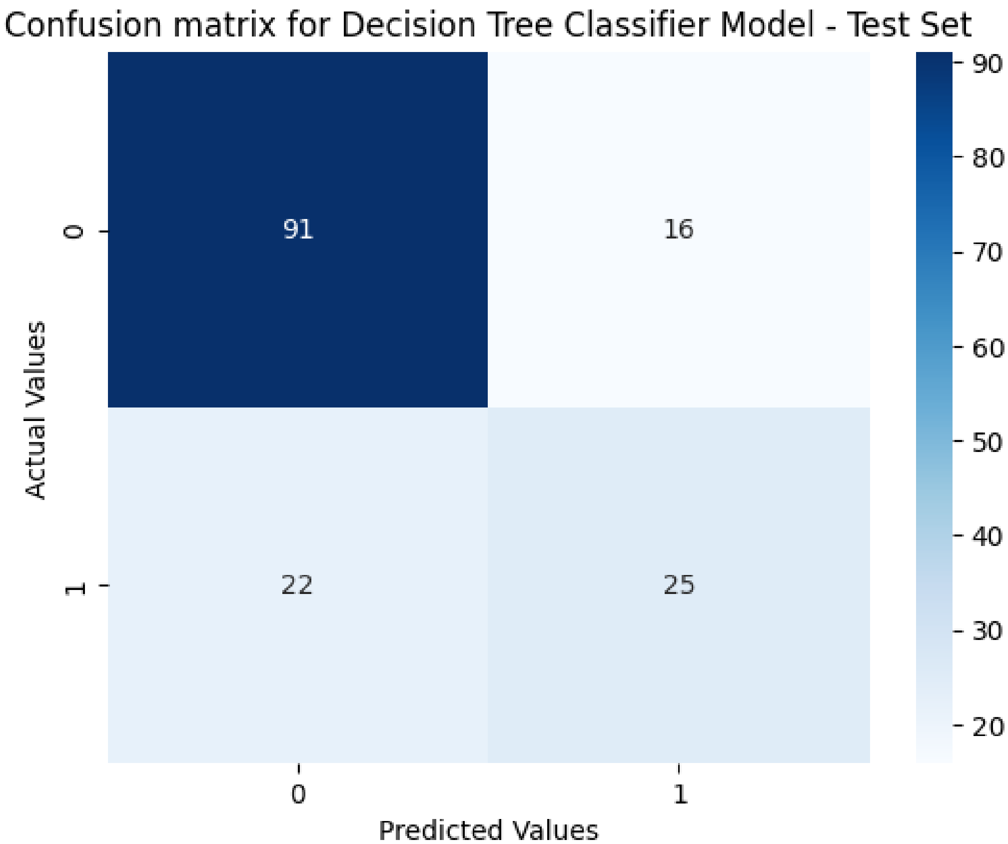
[31]:
```
DecisionTreeClassifier(max_depth=5)
```

6.2.2 Model Evaluation

[32]:
```
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
y_pred1 = classifier1.predict(X_test)
cm = confusion_matrix(y_test, y_pred1)
cm
```

[32]:
```
array([[91, 16],
       [22, 25]], dtype=int64)
```

[33]:
```
# Plotting the confusion matrix
# plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Decision Tree Classifier Model - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



[34]:
```
# Accuracy Score
score = round(accuracy_score(y_test, y_pred1),4)*100
print("Accuracy on test set: {}%".format(score))
```

```
Accuracy on test set: 75.32%
```

[35]:
```
# Creating a confusion matrix for training set
y_train_pred1 = classifier1.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred1)
cm
```

[35]:
```
array([[352,  41],
       [ 68, 153]], dtype=int64)
```

[36]:
```
# Accuracy Score
score = round(accuracy_score(y_train, y_train_pred1),4)*100
print("Accuracy on trainning set: {}%".format(score))
```

```
Accuracy on trainning set: 82.25%
```

6.2.3 For Support Vector Machine

[37]:
```
# Creating Support Vector Machine Model
from sklearn.svm import SVC
classifier2 = SVC(C=1, kernel='rbf', gamma='auto')
classifier2.fit(X_train, y_train)
```

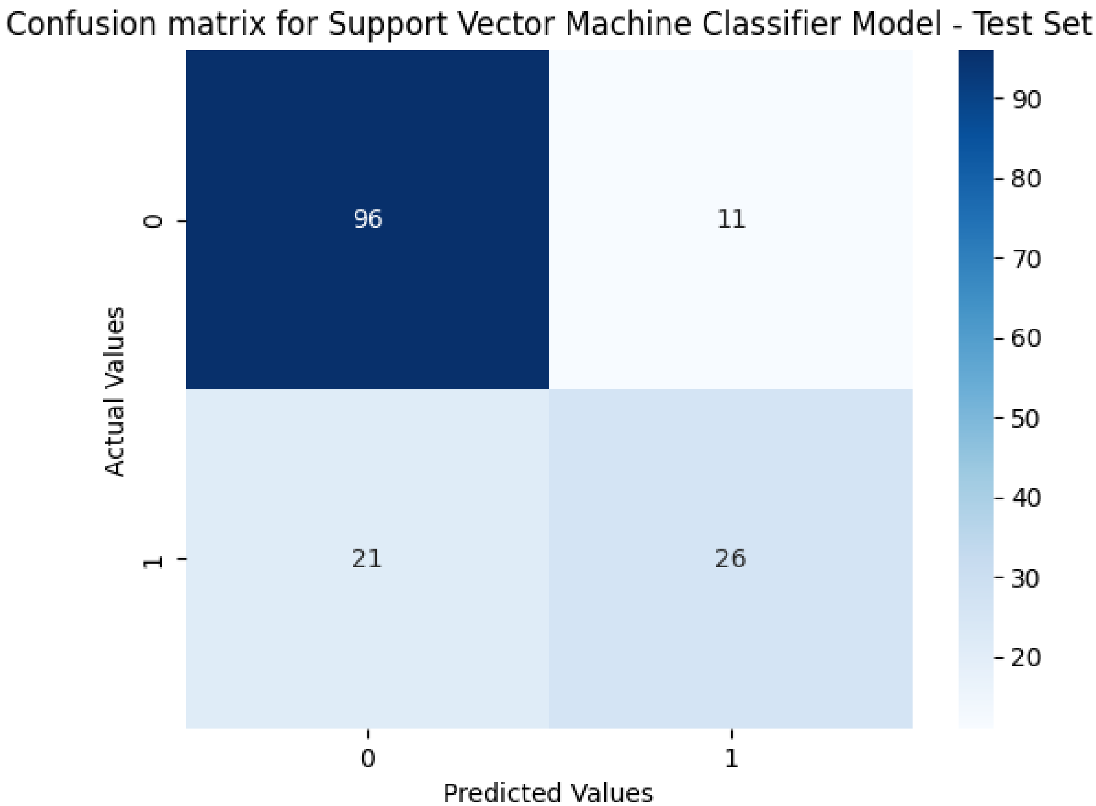
[37]:
```
SVC(C=1, gamma='auto')
```

6.2.4 Model Evaluation

[38]:
```
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
y_pred2 = classifier2.predict(X_test)
cm = confusion_matrix(y_test, y_pred2)
cm
```

[38]:
```
array([[96, 11],
       [21, 26]], dtype=int64)
```

[39]:
```
# Plotting the confusion matrix
# plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Support Vector Machine Classifier Model - Test␣
 ↪Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



[40]:
```
# Accuracy Score
score = round(accuracy_score(y_test, y_pred2),4)*100
print("Accuracy on test set: {}%".format(score))
```

```
Accuracy on test set: 79.22%
```

[41]:
```
# Creating a confusion matrix for training set
y_train_pred2 = classifier2.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred2)
cm
```

[41]:
```
array([[363,  30],
       [ 81, 140]], dtype=int64)
```

[42]:
```
# Accuracy Score
score = round(accuracy_score(y_train, y_train_pred2),4)*100
print("Accuracy on trainning set: {}%".format(score))
```

```
Accuracy on trainning set: 81.92%
```

6.2.5 For Random Forest

[43]:
```
# Creating Random Forest Model
classifier3 = RandomForestClassifier(n_estimators=58, random_state=0)
classifier3.fit(X_train, y_train)
```

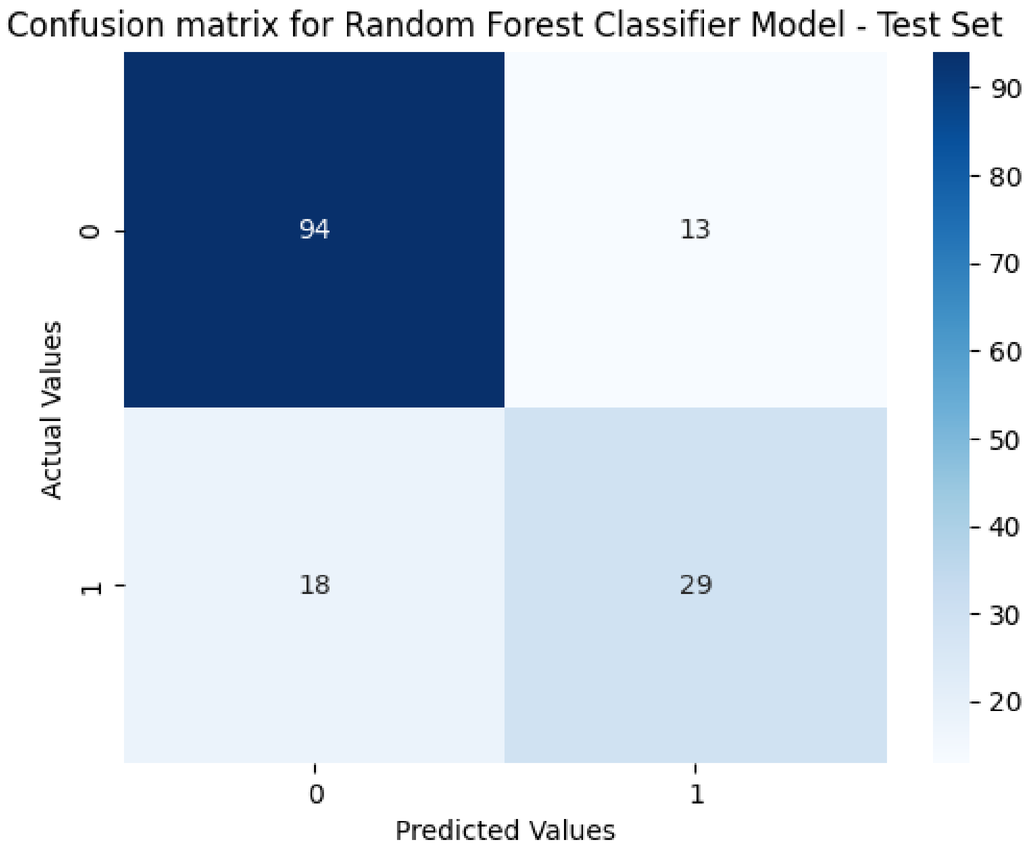
[43]:
```
RandomForestClassifier(n_estimators=58, random_state=0)
```

6.2.6 Model Evaluation

[44]:
```
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score
y_pred3 = classifier3.predict(X_test)
cm = confusion_matrix(y_test, y_pred3)
cm
```

[44]:
```
array([[94, 13],
       [18, 29]], dtype=int64)
```

[45]:
```
# Plotting the confusion matrix
# plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Random Forest Classifier Model - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



[46]:
```
# Accuracy Score
score = round(accuracy_score(y_test, y_pred3),4)*100
print("Accuracy on test set: {}%".format(score))
```

```
Accuracy on test set: 79.86999999999999%
```

[47]:
```
# Classification Report
# print(classification_report(y_test, y_pred))
```

[48]:
```
# Creating a confusion matrix for training set
y_train_pred3 = classifier3.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred3)
cm
```

[48]:
```
array([[393,   0],
       [  0, 221]], dtype=int64)
```

[49]:
```
##Plotting the confusion matrix

# plt.figure(figsize=(10,7))
```

```
# p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
# plt.title('Confusion matrix for Random Forest Classifier Model - Train Set')
# plt.xlabel('Predicted Values')
# plt.ylabel('Actual Values')
# plt.show()
```

[50]:
```
# Accuracy Score
score = round(accuracy_score(y_train, y_train_pred3),4)*100
print("Accuracy on trainning set: {}%".format(score))
```

```
Accuracy on trainning set: 100.0%
```

[51]:
```
# Classification Report
# print(classification_report(y_train, y_train_pred))
```

[52]:
```
# ensemble models
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Initialize other models
lr = LogisticRegression()
svc = SVC(probability=True, kernel='linear')

# Voting Classifier
voting_clf = VotingClassifier(estimators=[
    ('rf', rf),
    ('lr', lr),
    ('svc', svc)
], voting='soft')

# Fit Voting Classifier
voting_clf.fit(X_train, y_train)
y_pred_voting = voting_clf.predict(X_test)

# Evaluate Voting Classifier
print("Voting Classifier Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_voting):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred_voting))
```

```
Voting Classifier Performance:
Accuracy: 0.8247
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47
```

```
    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154
```

7 Predictions

[53]:
```python
# Creating a function for prediction
def predict_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness,
 ↪Insulin, BMI, DPF, Age):
    preg = int(Pregnancies)
    glucose = float(Glucose)
    bp = float(BloodPressure)
    st = float(SkinThickness)
    insulin = float(Insulin)
    bmi = float(BMI)
    dpf = float(DPF)
    age = int(Age)

    x = [[preg, glucose, bp, st, insulin, bmi, dpf, age]]
    x = sc.transform(x)

    return classifier3.predict(x)
```

7.1 Prediction 1

[54]:
```python
# Prediction 2
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
 ↪BMI, DPF, Age
prediction = predict_diabetes(1, 117, 88, 24, 145, 34.5, 0.403, 40)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Oops! You have diabetes.
```

```
C:\Users\sonuk\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```

8 Prediction 2

[55]:
```
# Prediction 3
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,␣
 ↪BMI, DPF, Age
prediction = predict_diabetes(5, 120, 92, 10, 81, 26.1, 0.551, 67)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Great! You don't have diabetes.
```

```
C:\Users\sonuk\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```