

▼ Diabetic Prediction



Dataset Link: <https://www.kaggle.com/datasets/aemyjutt/diabetesdataanalysis/data>

▼ Import necessary libraries

```
1 import sklearn
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import numpy as np
6
```

▼ Data Preprocessing

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

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

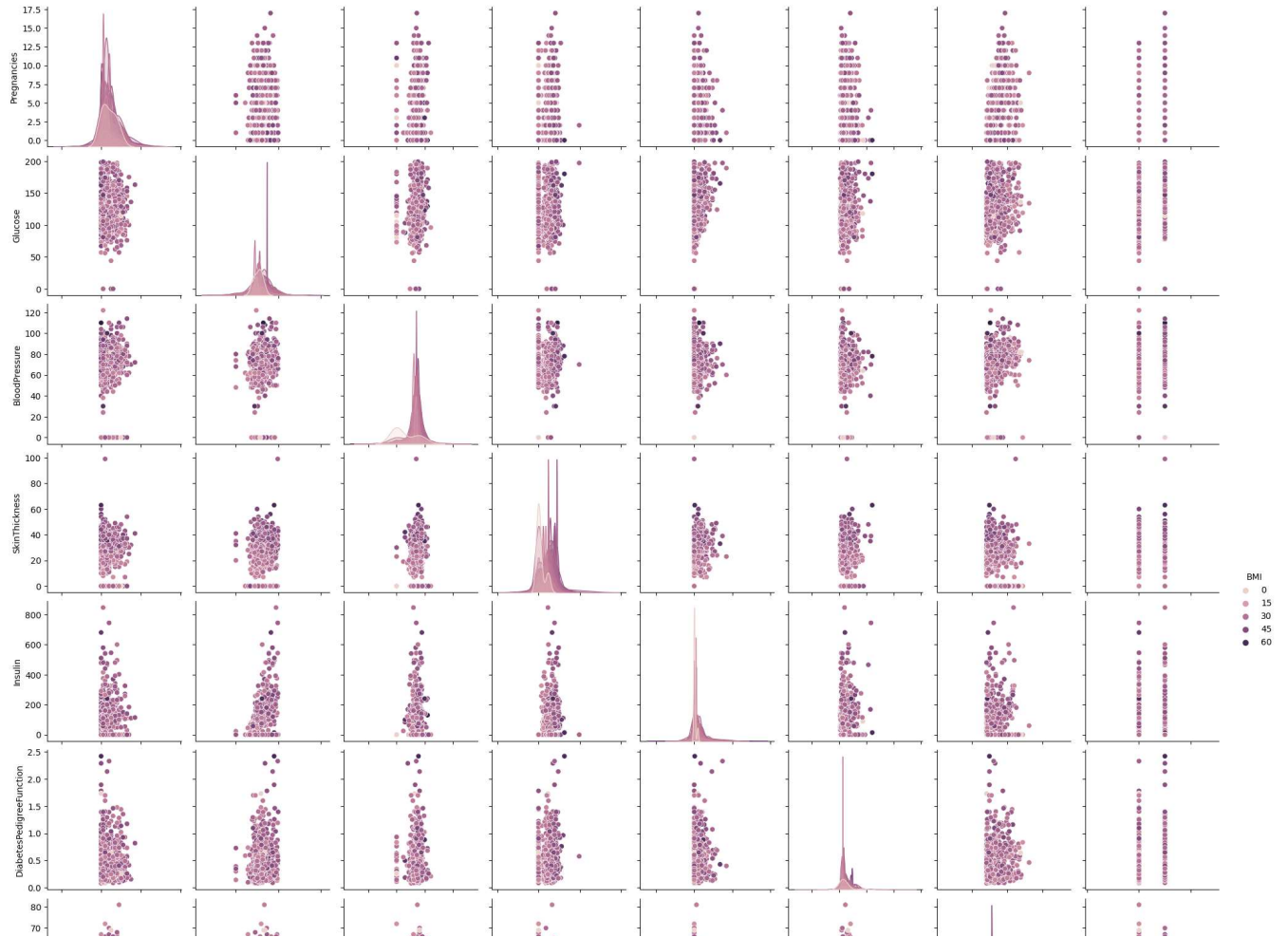
| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|-------------|---------|---------------|---------------|---------|------|--------------------------|-----|---------|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 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 |

```
1 df.shape
```

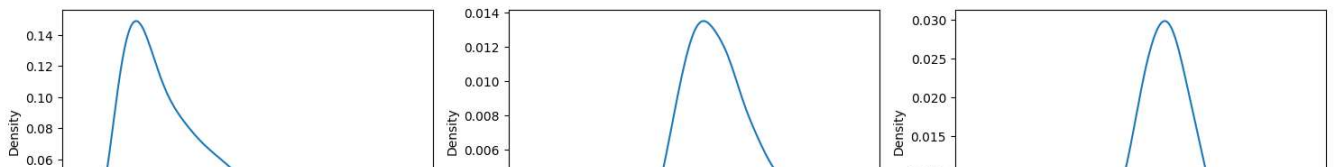
```
(768, 9)
```

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

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



```
1
2 #display distribution of data attributes
3 #we need to draw kernel density estimation to saw 0 values in all features except outcome feature
4 int_vars = df.select_dtypes(include = ['int','float'])
5 fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
6 axs = axs.flatten()
7
8 for i, var in enumerate (int_vars):
9     sns.kdeplot(x=var,data=df,ax=axs[i])
10
11 plt.tight_layout()
12 plt.show()
```



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

```
1 df.replace({'Pregnancies':0,
2           'Glucose':0,
3           'BloodPressure':0,
4           'SkinThickness':0,
5           'Insulin':0,
6           'BMI':0,
7           'DiabetesPedigreeFunction':0,
8           'Age':0},np.nan,inplace=True)
```

```
1 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
```

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

```
1 df.isnull().sum()
```

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

```
1 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

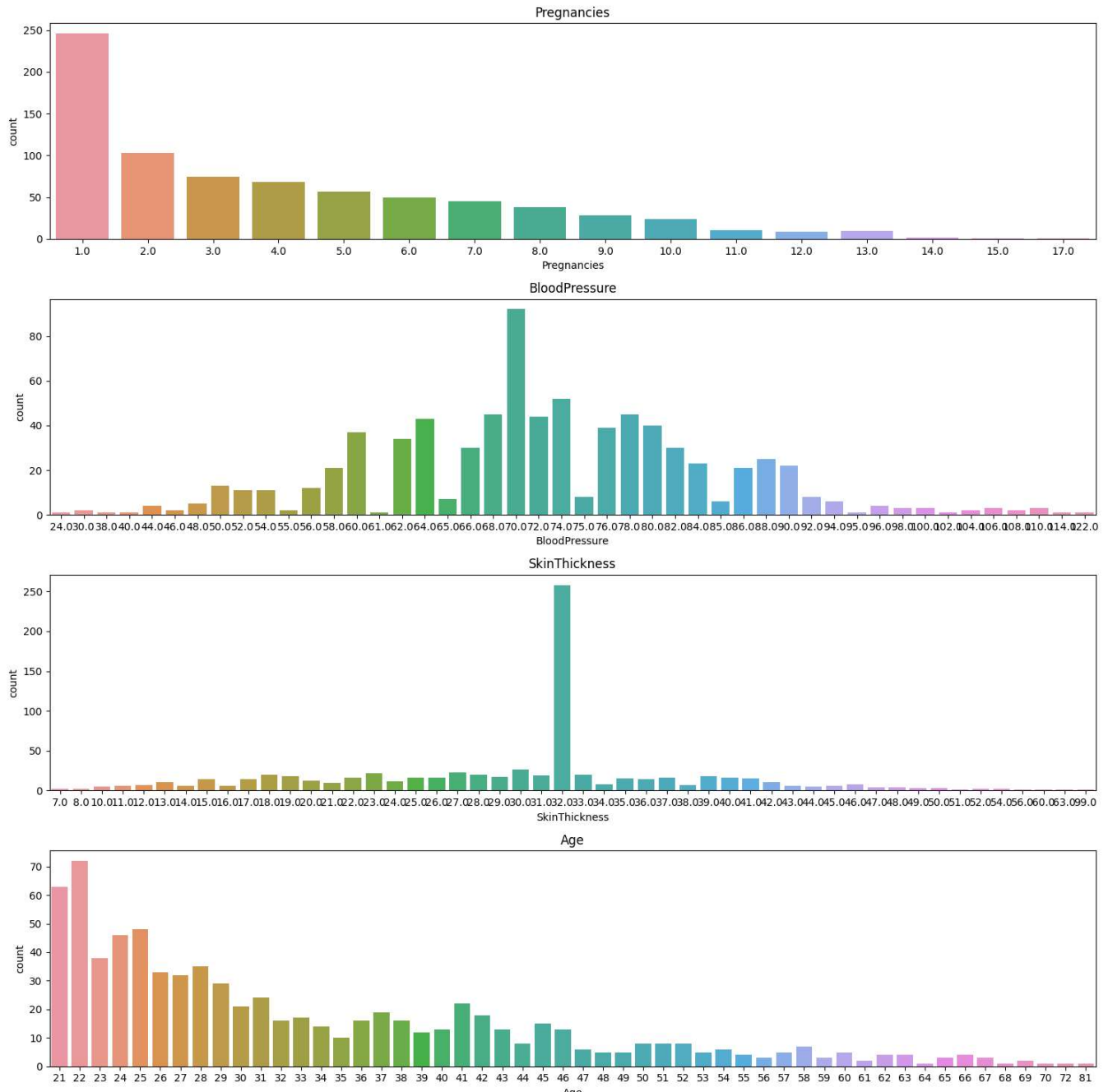
```
1 df.describe() #non graphical EDA
```

| | Pregnancies | 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 | 768.000000 |
| mean | 3.989583 | 121.539062 | 72.295573 | 29.994792 | 130.932292 | 32.450911 | 0.471876 | 33.240885 | 0.348951 |
| std | 3.219464 | 30.490660 | 12.106756 | 8.886506 | 88.700443 | 6.875366 | 0.331329 | 11.760232 | 0.476951 |
| min | 1.000000 | 44.000000 | 24.000000 | 7.000000 | 14.000000 | 18.200000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 64.000000 | 25.000000 | 105.000000 | 27.500000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 32.000000 | 105.000000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

```

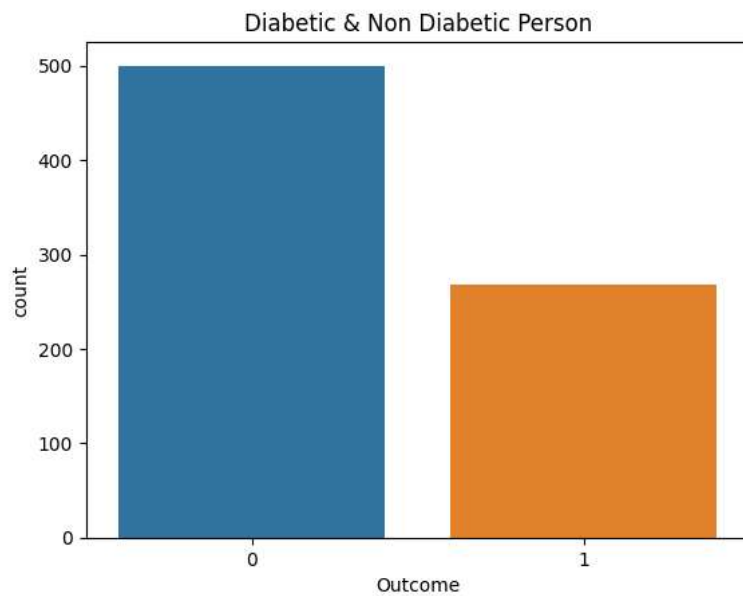
1 #graphical EDA
2 #get the count of these major features
3 #univariate analysis
4 int_vars = ['Pregnancies','BloodPressure','SkinThickness','Age']
5 fig,axs = plt.subplots(nrows=4, ncols=1, figsize=(15,15))
6 axs=axs.flatten()
7 for i, var in enumerate (int_vars):
8     sns.countplot(x=var,data=df,ax=axs[i])
9     axs[i].set_title(var)
10
11 plt.tight_layout()
12 plt.show()

```

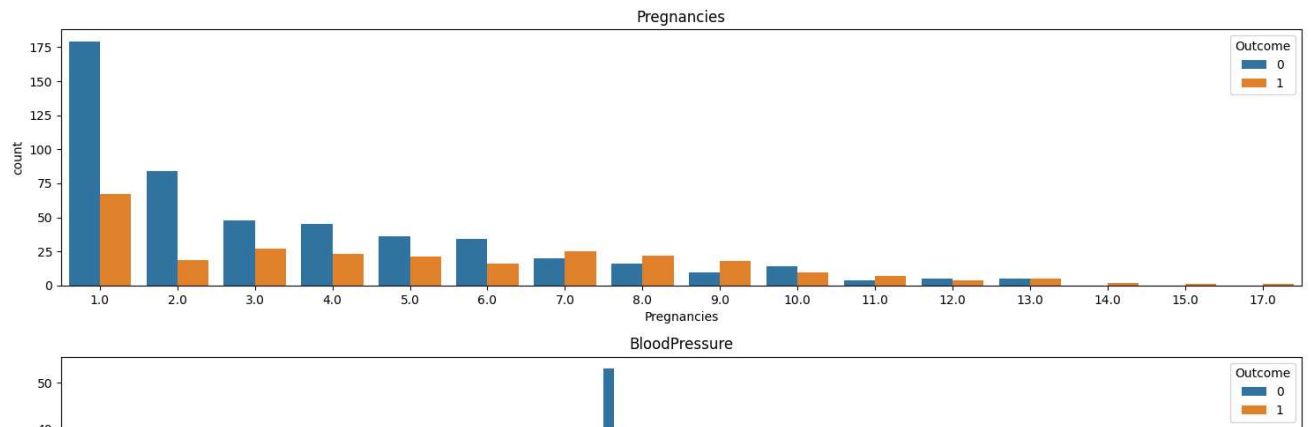


We see the different factor's occurrence now let's see the occurrence of diabetic & non-diabetic

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



```
1 #get the count of these major features with respect to Outcome
2 #Multivariate analysis
3 int_vars = ['Pregnancies','BloodPressure','SkinThickness','Age']
4 fig,axs = plt.subplots(nrows=4, ncols=1, figsize=(15,15))
5 axs=axs.flatten()
6 for i, var in enumerate (int_vars):
7     sns.countplot(x=var,hue='Outcome',data=df,ax=axs[i])
8     axs[i].set_title(var)
9
10 plt.tight_layout()
11 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

8 |

1 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
7.0 8.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 29.0 30.0 31.0 32.0 33.0 34.0 35.0 36.0 37.0 38.0 39.0 40.0 41.0 42.0 43.0 44.0 45.0 46.0 47.0 48.0 49.0 50.0 51.0 52.0 53.0 54.0 55.0 56.0 57.0 58.0 59.0 60.0 61.0 62.0 63.0 64.0 65.0 66.0 67.0 68.0 69.0 70.0 71.0 72.0 73.0 74.0 75.0 76.0 77.0 78.0 79.0 80.0 81.0 82.0 83.0 84.0 85.0 86.0 87.0 88.0 89.0 90.0 91.0 92.0 93.0 94.0 95.0 96.0 97.0 98.0 99.0 100.0 101.0 102.0 103.0 104.0 105.0 106.0 107.0 108.0 109.0 110.0 111.0 112.0 113.0 114.0 115.0 116.0 117.0 118.0 119.0 120.0 121.0 122.0 123.0 124.0 125.0 126.0 127.0 128.0 129.0 130.0 131.0 132.0 133.0 134.0 135.0 136.0 137.0 138.0 139.0 140.0 141.0 142.0 143.0 144.0 145.0 146.0 147.0 148.0 149.0 150.0 151.0 152.0 153.0 154.0 155.0 156.0 157.0 158.0 159.0 160.0 161.0 162.0 163.0 164.0 165.0 166.0 167.0 168.0 169.0 170.0 171.0 172.0 173.0 174.0 175.0 176.0 177.0 178.0 179.0 180.0 181.0 182.0 183.0 184.0 185.0 186.0 187.0 188.0 189.0 190.0 191.0 192.0 193.0 194.0 195.0 196.0 197.0 198.0 199.0 200.0 201.0 202.0 203.0 204.0 205.0 206.0 207.0 208.0 209.0 210.0 211.0 212.0 213.0 214.0 215.0 216.0 217.0 218.0 219.0 220.0 221.0 222.0 223.0 224.0 225.0 226.0 227.0 228.0 229.0 230.0 231.0 232.0 233.0 234.0 235.0 236.0 237.0 238.0 239.0 240.0 241.0 242.0 243.0 244.0 245.0 246.0 247.0 248.0 249.0 250.0 251.0 252.0 253.0 254.0 255.0 256.0 257.0 258.0 259.0 260.0 261.0 262.0 263.0 264.0 265.0 266.0 267.0 268.0 269.0 270.0 271.0 272.0 273.0 274.0 275.0 276.0 277.0 278.0 279.0 280.0 281.0 282.0 283.0 284.0 285.0 286.0 287.0 288.0 289.0 290.0 291.0 292.0 293.0 294.0 295.0 296.0 297.0 298.0 299.0 300.0 301.0 302.0 303.0 304.0 305.0 306.0 307.0 308.0 309.0 310.0 311.0 312.0 313.0 314.0 315.0 316.0 317.0 318.0 319.0 320.0 321.0 322.0 323.0 324.0 325.0 326.0 327.0 328.0 329.0 330.0 331.0 332.0 333.0 334.0 335.0 336.0 337.0 338.0 339.0 340.0 341.0 342.0 343.0 344.0 345.0 346.0 347.0 348.0 349.0 350.0 351.0 352.0 353.0 354.0 355.0 356.0 357.0 358.0 359.0 360.0 361.0 362.0 363.0 364.0 365.0 366.0 367.0 368.0 369.0 370.0 371.0 372.0 373.0 374.0 375.0 376.0 377.0 378.0 379.0 380.0 381.0 382.0 383.0 384.0 385.0 386.0 387.0 388.0 389.0 390.0 391.0 392.0 393.0 394.0 395.0 396.0 397.0 398.0 399.0 400.0 401.0 402.0 403.0 404.0 405.0 406.0 407.0 408.0 409.0 410.0 411.0 412.0 413.0 414.0 415.0 416.0 417.0 418.0 419.0 420.0 421.0 422.0 423.0 424.0 425.0 426.0 427.0 428.0 429.0 430.0 431.0 432.0 433.0 434.0 435.0 436.0 437.0 438.0 439.0 440.0 441.0 442.0 443.0 444.0 445.0 446.0 447.0 448.0 449.0 450.0 451.0 452.0 453.0 454.0 455.0 456.0 457.0 458.0 459.0 460.0 461.0 462.0 463.0 464.0 465.0 466.0 467.0 468.0 469.0 470.0 471.0 472.0 473.0 474.0 475.0 476.0 477.0 478.0 479.0 480.0 481.0 482.0 483.0 484.0 485.0 486.0 487.0 488.0 489.0 490.0 491.0 492.0 493.0 494.0 495.0 496.0 497.0 498.0 499.0 500.0 501.0 502.0 503.0 504.0 505.0 506.0 507.0 508.0 509.0 510.0 511.0 512.0 513.0 514.0 515.0 516.0 517.0 518.0 519.0 520.0 521.0 522.0 523.0 524.0 525.0 526.0 527.0 528.0 529.0 530.0 531.0 532.0 533.0 534.0 535.0 536.0 537.0 538.0 539.0 540.0 541.0 542.0 543.0 544.0 545.0 546.0 547.0 548.0 549.0 550.0 551.0 552.0 553.0 554.0 555.0 556.0 557.0 558.0 559.0 560.0 561.0 562.0 563.0 564.0 565.0 566.0 567.0 568.0 569.0 570.0 571.0 572.0 573.0 574.0 575.0 576.0 577.0 578.0 579.0 580.0 581.0 582.0 583.0 584.0 585.0 586.0 587.0 588.0 589.0 590.0 591.0 592.0 593.0 594.0 595.0 596.0 597.0 598.0 599.0 600.0 601.0 602.0 603.0 604.0 605.0 606.0 607.0 608.0 609.0 610.0 611.0 612.0 613.0 614.0 615.0 616.0 617.0 618.0 619.0 620.0 621.0 622.0 623.0 624.0 625.0 626.0 627.0 628.0 629.0 630.0 631.0 632.0 633.0 634.0 635.0 636.0 637.0 638.0 639.0 640.0 641.0 642.0 643.0 644.0 645.0 646.0 647.0 648.0 649.0 650.0 651.0 652.0 653.0 654.0 655.0 656.0 657.0 658.0 659.0 660.0 661.0 662.0 663.0 664.0 665.0 666.0 667.0 668.0 669.0 670.0 671.0 672.0 673.0 674.0 675.0 676.0 677.0 678.0 679.0 680.0 681.0 682.0 683.0 684.0 685.0 686.0 687.0 688.0 689.0 690.0 691.0 692.0 693.0 694.0 695.0 696.0 697.0 698.0 699.0 700.0 701.0 702.0 703.0 704.0 705.0 706.0 707.0 708.0 709.0 710.0 711.0 712.0 713.0 714.0 715.0 716.0 717.0 718.0 719.0 720.0 721.0 722.0 723.0 724.0 725.0 726.0 727.0 728.0 729.0 730.0 731.0 732.0 733.0 734.0 735.0 736.0 737.0 738.0 739.0 740.0 741.0 742.0 743.0 744.0 745.0 746.0 747.0 748.0 749.0 750.0 751.0 752.0 753.0 754.0 755.0 756.0 757.0 758.0 759.0 760.0 761.0 762.0 763.0 764.0 765.0 766.0 767.0 768.0
```

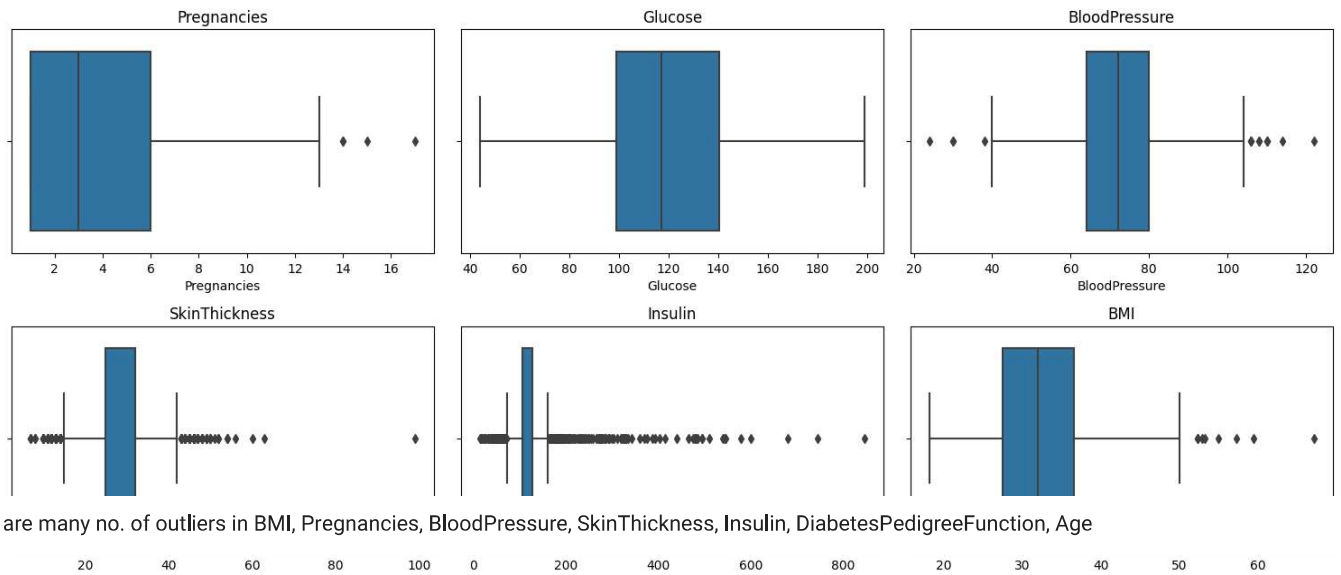
As we see the BMI & DiabetesPedigreeFunction is in float with this we are detecting the outliers

10 |

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

Outlier detection

```
1
2 #Outlier Detection
3 #Boxplot is help to clearly see the outliers
4 int_vars = df.select_dtypes(include = ['int','float'])
5 fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
6 axs = axs.flatten()
7 for i, var in enumerate(int_vars):
8     sns.boxplot(x=var,data=df,ax=axs[i])
9     axs[i].set_title(var)
10
11 plt.tight_layout()
12 plt.show()
13
```

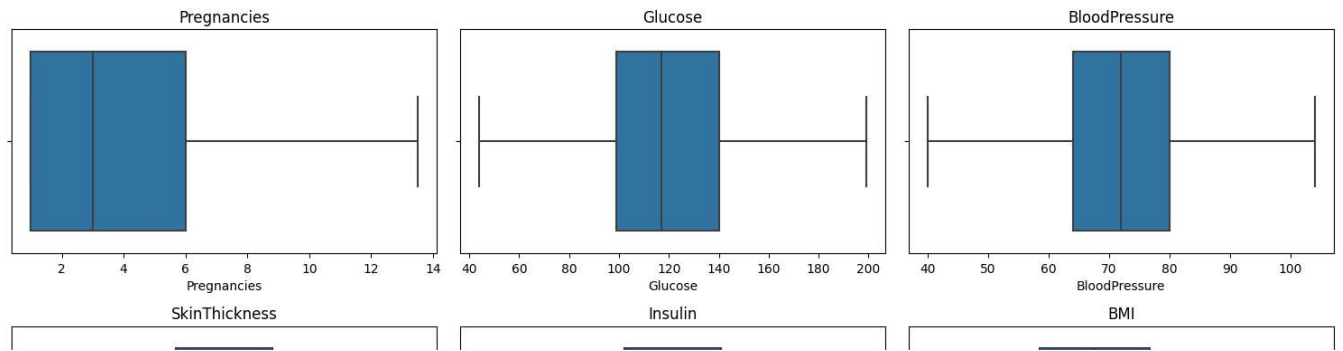


there are many no. of outliers in BMI, Pregnancies, BloodPressure, SkinThickness, Insulin, DiabetesPedigreeFunction, Age

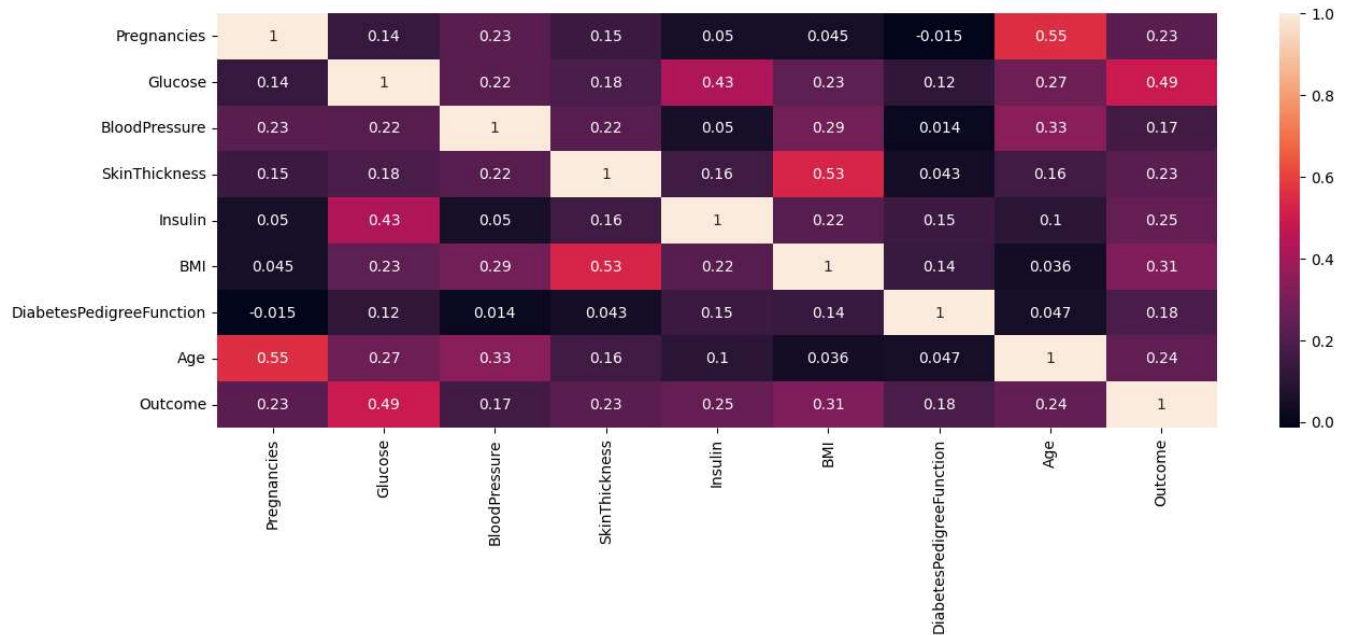
```
1 fea_outlier=['BMI', 'Pregnancies', 'BloodPressure', 'SkinThickness', 'Insulin', 'DiabetesPedigreeFunction', 'Age']
```

```
2 #remove outliers
3 #calculate the Q1, Q2, Q3 ... if data is not in inter quartile range then it eliminate
4 #make user defined function
5 def remove_outliers_iqr(data):
6     # Calculate the first quartile (Q1) and third quartile (Q3)
7     Q1 = np.percentile(data, 25)
8     Q3 = np.percentile(data, 75)
9     # Calculate the interquartile range (IQR)
10    IQR = Q3 - Q1
11    # Define the lower and upper bounds for outliers
12    lower_bound = Q1 - 1.5 * IQR
13    upper_bound = Q3 + 1.5 * IQR
14    # Remove outliers
15    data = np.where(data>upper_bound, upper_bound, np.where(data<lower_bound,lower_bound,data))
16
17    return data[(data >= lower_bound) & (data <= upper_bound)]
18
19 #apply for those coulms which contain outliers
20 for col in fea_outlier:
21     df[col]= remove_outliers_iqr(df[col])
```

```
1 #now again see the any outlier is remain or not
2 int_vars = df.select_dtypes(include = ['int','float'])
3 fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15,10))
4 axs = axs.flatten()
5 for i, var in enumerate (int_vars):
6     sns.boxplot(x=var,data=df,ax=axs[i])
7     axs[i].set_title(var)
8
9 plt.tight_layout()
10 plt.show()
11
```



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



Cleary see that all the outliers are remove

▼ Data Modeling

```
1 correlation_matrix = df.corr()
2 print(correlation_matrix['Outcome'].abs().sort_values(ascending=False))
3
```

```
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

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

Data splitting


```

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

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

1 # scaling data
2 from sklearn.preprocessing import MinMaxScaler
3 scale = MinMaxScaler()
4 x_train = scale.fit_transform(x_train)
5 x_test = scale.transform(x_test)
6 print('Shape of x_train:', x_train.shape)
7 print('Shape of x_test:',x_test.shape)
8

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.

```

1 from sklearn.linear_model import LogisticRegression
2 lr=LogisticRegression()
3 lr.fit(x_train,y_train)
4 yL_train_pred=lr.predict(x_train)
5 yL_test_pred=lr.predict(x_test)

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

Accuracy : 0.7559055118110236

1 # importing evaluation metrics
2 from sklearn.metrics import r2_score, mean_squared_error, classification_report, confusion_matrix
3
4 print('Train data results')
5 print('RMSE:', np.sqrt(mean_squared_error(yL_train_pred, y_train)))
6 print('R-squared:',r2_score(yL_train_pred,y_train))
7 print('Test data results')
8 print('RMSE:', np.sqrt(mean_squared_error(yL_test_pred, y_test)))
9 print('R-squared:',r2_score(yL_test_pred, y_test))
10 print("\n Classification Report:\n",classification_report(y_test,yL_test_pred))
11 print("Confusion Matrix : \n", confusion_matrix(y_test,yL_test_pred))
12

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]]

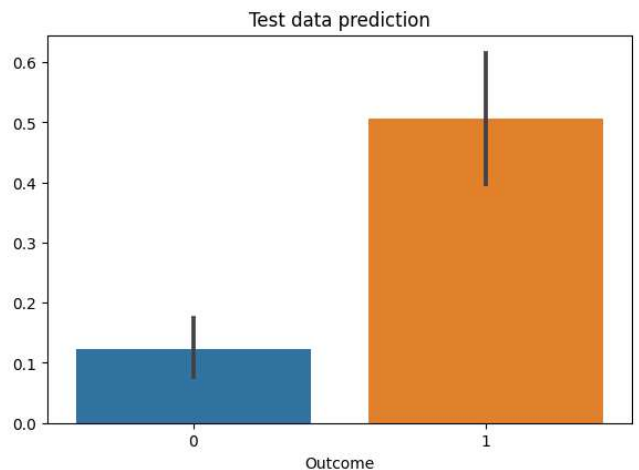
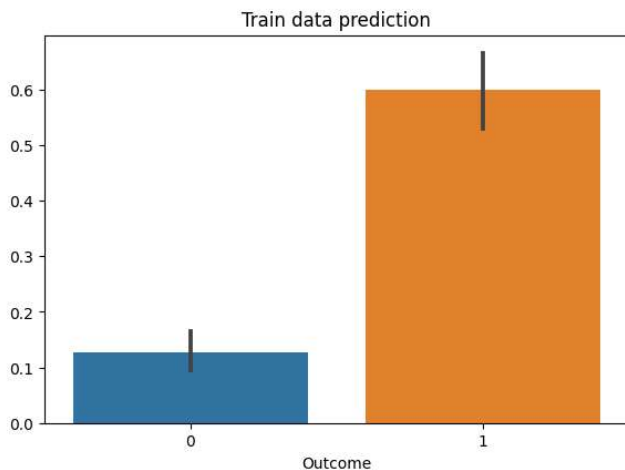
1 # plotting results for visual
2 plt.figure(figsize=(15,10))
3 plt.subplot(2,2,1)
4 plt.title('Train data prediction')
5 sns.barplot(x=y_train,y=yL_train_pred)
6 plt.subplot(2,2,2)
7 plt.title('Test data prediction')

```

```

8 sns.barplot(x=y_test,y=yL_test_pred)
9 plt.show()
10

```



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

```

1 from sklearn.ensemble import RandomForestClassifier
2 RF = RandomForestClassifier()
3 RF.fit(x_train, y_train)
4 yR_train_pred=RF.predict(x_train)
5 yR_test_pred=RF.predict(x_test)

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

    Accuracy :  0.7834645669291339

1 #Evaluation metrics
2 print('Train data results')
3 print('RMSE:', np.sqrt(mean_squared_error(yR_train_pred, y_train)))
4 print('R-squared:',r2_score(yR_train_pred,y_train))
5 print('Test data results')
6 print('RMSE:', np.sqrt(mean_squared_error(yR_test_pred, y_test)))
7 print('R-squared:',r2_score(yR_test_pred, y_test))
8 print("\n Classification Report:\n",classification_report(y_test,yR_test_pred))
9 print("Confusion Matrix : \n", confusion_matrix(y_test,yR_test_pred))
10

```

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 :

```

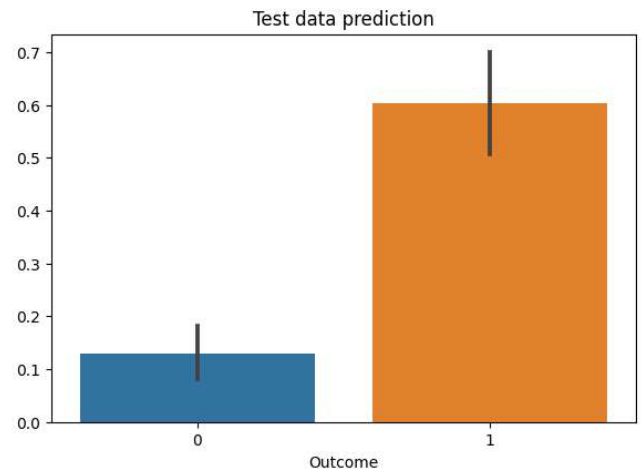
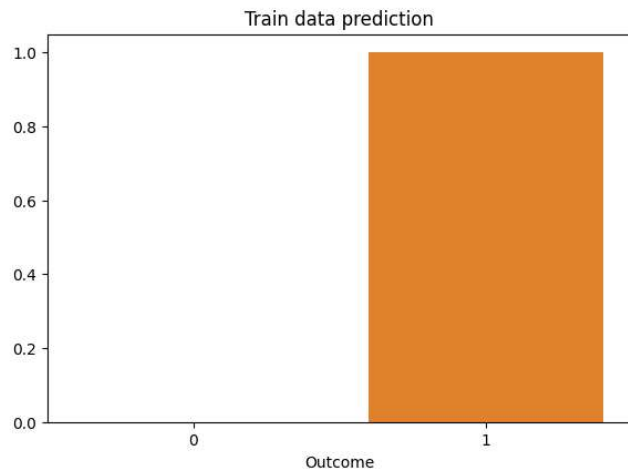
[[149  22]
 [ 33  50]]

```

```

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

```



Support Vector Machine : This classifies the elements on the basis of hyperplane & with good margin

```
1 from sklearn.svm import SVC
2 sv = SVC()
3 sv.fit(x_train, y_train)
4 yS_train_pred=sv.predict(x_train)
5 yS_test_pred=sv.predict(x_test)

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

Accuracy : 0.7755905511811023

1 #Evaluation metrics
2 print('Train data results')
3 print('RMSE:', np.sqrt(mean_squared_error(yS_train_pred, y_train)))
4 print('R-squared:', r2_score(yS_train_pred, y_train))
5 print('Test data results')
6 print('RMSE:', np.sqrt(mean_squared_error(yS_test_pred, y_test)))
7 print('R-squared:', r2_score(yS_test_pred, y_test))
8 print("\n Classification Report:\n", classification_report(y_test, yS_test_pred))
9 print("Confusion Matrix : \n", confusion_matrix(y_test, yS_test_pred))
10
```

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

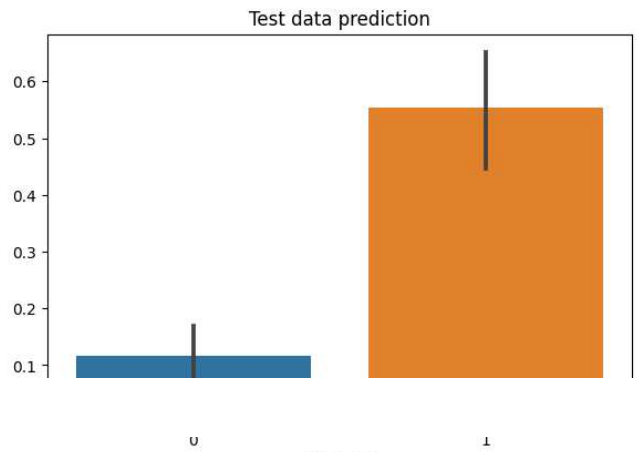
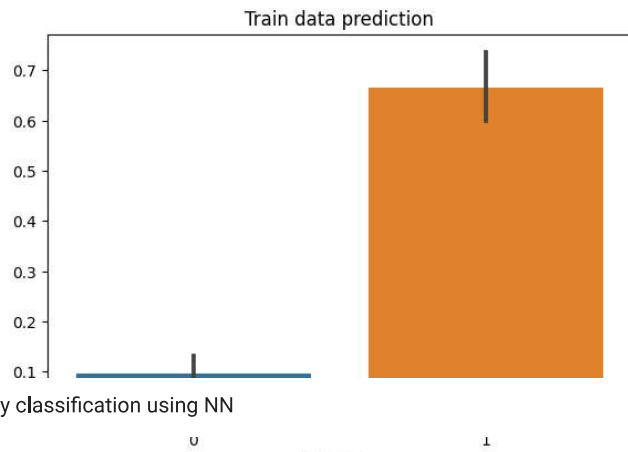
```
Classification Report:
      precision    recall  f1-score   support

     0       0.80      0.88      0.84        171
     1       0.70      0.55      0.62         83

 accuracy          0.75
 macro avg          0.75
weighted avg          0.75
```

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

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



binary classification using NN

```

1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense
3 from tensorflow.keras.layers import Dropout
4
5 # Initialize the model
6 model = Sequential()
7
8 # Add input layer and first hidden layer
9 model.add(Dense(units=128, activation='relu', input_dim=x_train.shape[1]))
10 model.add(Dropout(0.2)) # Dropout layer with a dropout rate of 0.2
11 # Add more hidden layers if needed
12 model.add(Dense(units=64, activation='relu'))
13 model.add(Dropout(0.2)) # Dropout layer with a dropout rate of 0.2
14 # Add the output layer with sigmoid activation for binary classification
15 model.add(Dense(units=1, activation='sigmoid'))
16 model.add(Dropout(0.2)) # Dropout layer with a dropout rate of 0.2
17 # Compile the model
18 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
19 #train
20 # Train the model
21 model.fit(x_train, y_train, epochs=100, batch_size=32, validation_data=(x_test, y_test))
22 yNN_train_pred=model.predict(x_train)
23 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: 0.7555
Epoch 98/100
17/17 [=====] - 0s 4ms/step - loss: 1.6362 - accuracy: 0.7490 - val_loss: 0.4644 - val_accuracy: 0.7555
Epoch 99/100
17/17 [=====] - 0s 5ms/step - loss: 1.3002 - accuracy: 0.7821 - val_loss: 0.4644 - val_accuracy: 0.7526
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

```

```

1 # Evaluate the model on the test set
2 loss, accuracy = model.evaluate(x_test, y_test)
3 print("Test Loss:", loss)
4 print("Test Accuracy:", accuracy)
5 #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

```

```

1 #Evaluation metrics
2 print('Train data results')
3 print('RMSE:', np.sqrt(mean_squared_error(yNN_train_pred, y_train)))
4 print('R-squared:', r2_score(yNN_train_pred,y_train))
5 print('Test data results')
6 print('RMSE:', np.sqrt(mean_squared_error(yNN_test_pred, y_test)))
7 print('R-squared:', r2_score(yNN_test_pred, y_test))
8
9 #We can't generate the classification report & confusion matrix because 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.

```

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

```

```

Accuracy : 0.7716535433070866

```

```

1 #Evaluation metrics
2 print('Train data results')
3 print('RMSE:', np.sqrt(mean_squared_error(ySC_train_pred, y_train)))
4 print('R-squared:', r2_score(ySC_train_pred,y_train))
5 print('Test data results')
6 print('RMSE:', np.sqrt(mean_squared_error(ySC_test_pred, y_test)))
7 print('R-squared:', r2_score(ySC_test_pred, y_test))
8 print("\n Classification Report:\n",classification_report(y_test,ySC_test_pred))
9 print("Confusion Matrix : \n", confusion_matrix(y_test,ySC_test_pred))
10

```

```

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

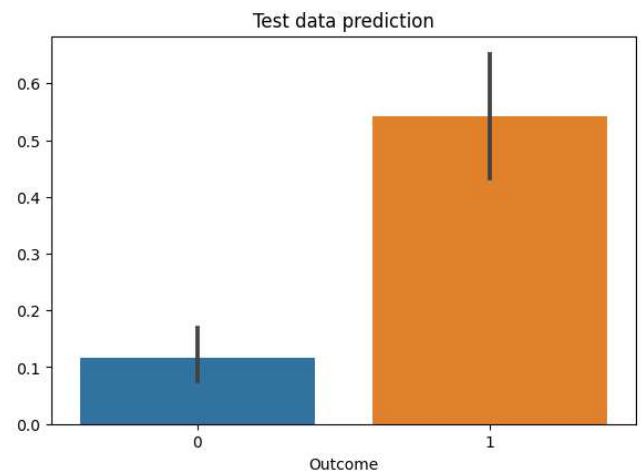
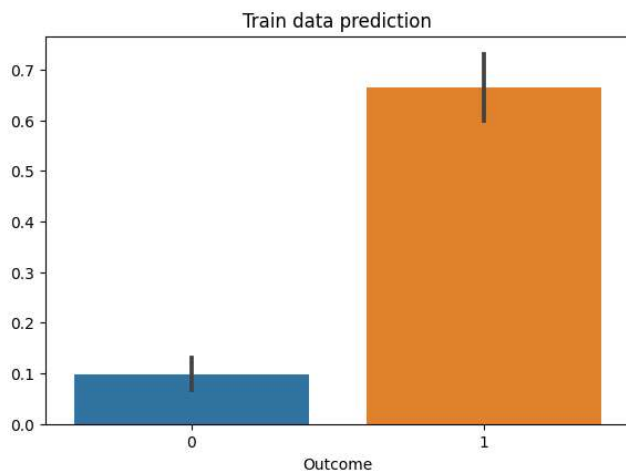
     0       0.80      0.88      0.84       171
     1       0.69      0.54      0.61        83

 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]]

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

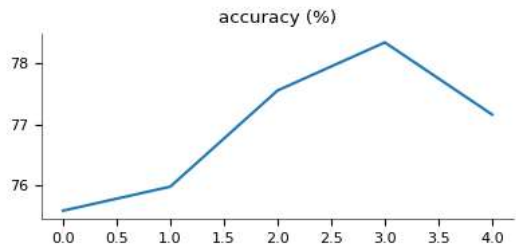


Final Report

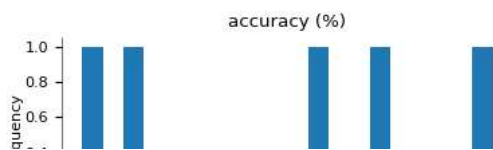
```
1 from sklearn.metrics import r2_score, mean_squared_error
2 data = [['Logistic_Regression',accuracy_score(y_test,yL_test_pred)*100],
3         ['Artificial_Neural_Network',accuracy*100],
4         ['SVM',accuracy_score(y_test,yS_test_pred)*100],
5         ['Random_Forest',accuracy_score(y_test,yR_test_pred)*100],
6         ['Stacking',accuracy_score(y_test,ySC_test_pred)*100]]
7 result = pd.DataFrame(data, columns=['algo','accuracy (%)'])
8 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

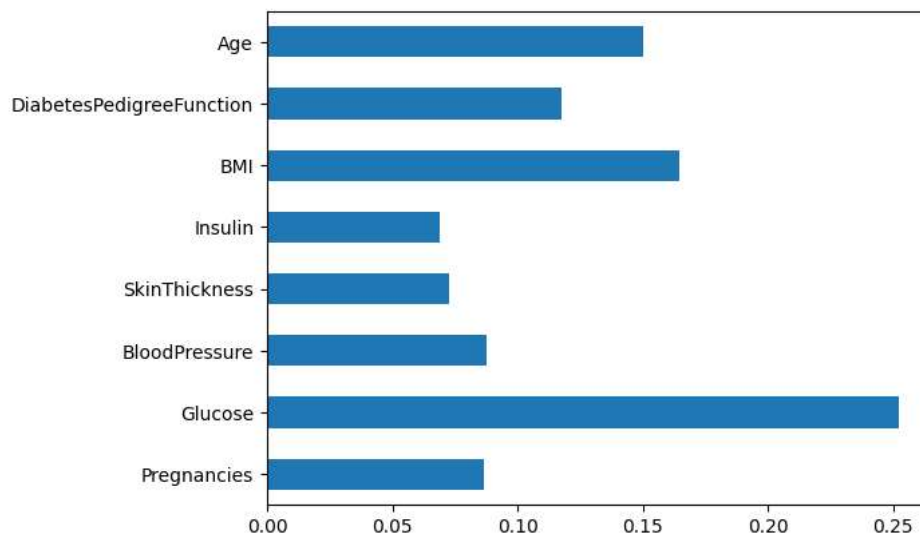
Feature Importance

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

```
1 #through bar graph visualize the important feature in whole model
2 (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

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

```

3 #Then we will be loading that saved model
4 SC_from_pickle = pickle.loads(save_model)
5
6 # lastly, after loading that model we will use this to make predictions
7 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, 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, 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, 0, 1, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
        0, 1, 1, 1, 0, 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, 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

```
1 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 |

```

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

array([0])

```

1

Result of stacking might be somewhat wrong

Because in dataset its positive result but model show negative

```

1 from sklearn.preprocessing import StandardScaler
2 input=(5,180,100,60,570,25.5,0.128,40)
3 input2array=np.asarray(input)
4 input_reshape=input2array.reshape(1,-1)
5 sd_data=scale.transform(input_reshape)
6 prediction=RF.predict(sd_data)
7 prediction
8 if prediction == 0:
9     print("Patient is not diabetic ")
10 else:
11     print("Patient is diabetic ")
12
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