SHAHEED ZULFIKAR ALI BHUTTO
INSTITUTE OF SCIENCE AND TECHNOLOGY

Diabetes Data Analysis

Submitted by

Arslan Ali
(2012110)
Naveen kingrani
(2012127)

SECTION 7-A

Course: DATA SCIENCE

Instructor
Dr Imran Amin,
Sir Ahsan Nisar

Department of Computer Science

[04-January-2024]

TABLE OF CONTENTS

INTRODUCTON

Diabetes is a widespread disease with a significant impact on human health. In this project we are working on Diabetes data set which is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.The objective of the data set is to diagnostically predict whether a patient has diabetes, based on certain diagnostic measurements included in the data set 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.The project aims to contribute to early diagnosis and predict diabetes based on data set information about patient record by machine learning models and data visualization. .

Executive Summary

This report presents a comprehensive analysis of a data science project focused on diabetes prediction using Diabetes data set which is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The goal of this project is to use machine learning techniques in Python to build a robust model that can predict the presence of diabetes in patient based on give information in data set.

Data Collection and Exploration

1: **Data set:**

The data set used in this project comprises patient health record with features such as blood pressure, number of pregnancies, glucose, insulin, age, BMI.

and the target variable is Outcome by which we can predict whether patient has diabetes or not.

2: **Data Exploration:**

Exploratory Data Analysis (EDA) techniques were used to understand the distribution and characteristics of the data. Data cleaning, filtering, Visualizations, Model evaluation, correlation analyses and predictions are performed.

Data Cleaning and Filter

Initially data set was unclean like there were many empty rows, null values, duplicate rows, unnamed columns. So we cleaned data set and filtered it accordingly.

*Data set initially:

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
path="/content/drive/MyDrive/DataScience/diabetes_dataset.csv"
df = pd.read_csv(path)
print(df)
```

```
     Column1  Column2  Column3  Column4  Column4.1  Column5  Column6  Column7  \
0        NaN      NaN      NaN      NaN        NaN      NaN      NaN      NaN   
1        NaN      NaN      NaN      NaN        NaN      NaN      NaN      NaN   
2        6.0    148.0     72.0     35.0        0.0     33.6    0.627     50.0   
3        1.0     85.0     66.0     29.0        0.0     26.6    0.351     31.0   
4        8.0    183.0     64.0      0.0        0.0     23.3    0.672     32.0   
..       ...      ...      ...      ...        ...      ...      ...      ...   
793     10.0    101.0     76.0     48.0      180.0     32.9    0.171     63.0   
794      2.0    122.0     70.0     27.0        0.0     36.8    0.340     27.0   
795      5.0    121.0     72.0     23.0      112.0     26.2    0.245     30.0   
796      1.0    126.0     60.0      0.0        0.0     30.1    0.349     47.0   
797      1.0     93.0     70.0     31.0        0.0     30.4    0.315     23.0   

     Column8     Cclumn9
0        NaN         NaN
1        NaN         NaN
2        1.0    1/2/1960
3        0.0  11/12/1961
4        1.0   21/2/1962
```

*Column names renaming

```
[ ] df.rename(columns={"Column1":'Number_of_Pregnancies',"Column2":"Glucose","Column3" : "BloodPressure", "Column4":"SkinThicl
                      "Column4.1":"Insulin","Column5":"BMI","Column6":"DiabetesPedigreeFunction","Column7":"Age","Column8":"(
```

```
df.head()
```

	Number_of_Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	1.0	1/
3	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	0.0	11/1
4	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	1.0	21/

*Data types conversion

```
#Original Data Types
df.dtypes
```

```
Number_of_Pregnancies       float64
Glucose                     float64
BloodPressure               float64
SkinThickness               float64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                         float64
Outcome                     float64
DOB                          object
dtype: object
```

```
[ ] #After Data Type conversion
    df['Outcome'] = df['Outcome'].astype(str)
    df.dtypes
```

```
Number_of_Pregnancies       float64
Glucose                     float64
BloodPressure               float64
SkinThickness               float64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                         float64
Outcome                      object
DOB                          object
dtype: object
```

```
[ ] df['Age'] = df['Age'].astype(int)
    df['Number_of_Pregnancies'] = df['Number_of_Pregnancies'].astype(int)
    df.dtypes

    Number_of_Pregnancies         int64
    Glucose                     float64
    BloodPressure               float64
    SkinThickness               float64
    Insulin                     float64
    BMI                         float64
    DiabetesPedigreeFunction    float64
    Age                           int64
    Outcome                      object
    dtype: object
```

*Identifying Null values in data set

```
print(df.isnull().sum())

Number_of_Pregnancies        26
Glucose                      26
BloodPressure                26
SkinThickness                26
Insulin                      26
BMI                          26
DiabetesPedigreeFunction     26
Age                          26
Outcome                       0
DOB                         746
dtype: int64

[ ] print(df['DOB'].isnull().sum())

746
```

* Removing rows with null values in specific column & then replacing those with particular value

```
# Remove rows with missing values in a specific column
df_cleaned = df.dropna(subset=['DOB'])

[ ] df['DOB'].fillna("1/9/1973", inplace=True)
```

```
print(df)
```

```
     Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  \
0                      NaN      NaN            NaN            NaN      NaN
1                      NaN      NaN            NaN            NaN      NaN
2                      6.0    148.0           72.0           35.0      0.0
3                      1.0     85.0           66.0           29.0      0.0
4                      8.0    183.0           64.0            0.0      0.0
..                     ...      ...            ...            ...      ...
793                   10.0    101.0           76.0           48.0    180.0
794                    2.0    122.0           70.0           27.0      0.0
795                    5.0    121.0           72.0           23.0    112.0
796                    1.0    126.0           60.0            0.0      0.0
797                    1.0     93.0           70.0           31.0      0.0

      BMI  DiabetesPedigreeFunction   Age Outcome        DOB
0     NaN                       NaN   NaN     nan   1/9/1973
1     NaN                       NaN   NaN     nan   1/9/1973
2    33.6                     0.627  50.0     1.0   1/2/1960
3    26.6                     0.351  31.0     0.0 11/12/1961
4    23.3                     0.672  32.0     1.0  21/2/1962
..    ...                       ...   ...     ...        ...
793  32.9                     0.171  63.0     0.0   1/9/1973
794  36.8                     0.340  27.0     0.0   1/9/1973
795  26.2                     0.245  30.0     0.0   1/9/1973
796  30.1                     0.349  47.0     1.0   1/9/1973
797  30.4                     0.315  23.0     0.0   1/9/1973
```

* Identifying number of empty rows then dropping it

```
# Number of rows before dropping empty rows
num_rows = len(df)
print("Number of rows:", num_rows)
```

```
Number of rows: 798
```

```
# Identify missing values and sum them by row
empty_rows = df.isnull().sum(axis=1)

num_empty_rows = (empty_rows > 0).sum()

print("Number of empty rows:", num_empty_rows)
```

```
Number of empty rows: 26
```

```
df.dropna(inplace=True)
```

* Identifying duplicate rows in data set

```
duplicate_rows = df[df.duplicated()]
print("Rows with duplicate values:")
print(duplicate_rows)
```

```
Rows with duplicate values:
    Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  \
5                     8.0    183.0           64.0            0.0      0.0
10                    3.0     78.0           50.0           32.0     88.0
18                   10.0    168.0           74.0            0.0      0.0
24                    0.0    118.0           84.0           47.0    230.0

     BMI  DiabetesPedigreeFunction   Age Outcome        DOB
5   23.3                     0.672  32.0     1.0  21/2/1962
10  31.0                     0.248  26.0     1.0  18/2/1965
18  38.0                     0.537  34.0     1.0  11/2/1975
24  45.8                     0.551  31.0     1.0  31/2/1965
```

* Removing duplicate rows

```
# remove duplicate rows based on all columns
df.drop_duplicates(inplace=True)

df.head(30)
```

	Number_of_Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	1.0
3	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	0.0
4	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	1.0
6	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	0.0
7	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	1.0
8	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	0.0
9	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	1.0
11	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	0.0
12	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	1.0
13	8.0	125.0	96.0	0.0	0.0	0.0	0.232	54.0	1.0
16	4.0	110.0	92.0	0.0	0.0	37.6	0.191	30.0	0.0

* Dropping unnecessary columns

```
[ ] print("Dataset before dropping DOB Column",df)
```

```
Dataset before dropping DOB Column      Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  \
2                          6.0     148.0          72.0           35.0      0.0
3                          1.0      85.0          66.0           29.0      0.0
4                          8.0     183.0          64.0            0.0      0.0
6                          1.0      89.0          66.0           23.0     94.0
7                          0.0     137.0          40.0           35.0    168.0
..                         ...       ...           ...            ...      ...
793                       10.0     101.0          76.0           48.0    180.0
794                        2.0     122.0          70.0           27.0      0.0
795                        5.0     121.0          72.0           23.0    112.0
796                        1.0     126.0          60.0            0.0      0.0
797                        1.0      93.0          70.0           31.0      0.0

      BMI  DiabetesPedigreeFunction   Age  Outcome         DOB
2    33.6                     0.627  50.0      1.0    1/2/1960
3    26.6                     0.351  31.0      0.0  11/12/1961
4    23.3                     0.672  32.0      1.0   21/2/1962
6    28.1                     0.167  21.0      0.0    1/9/1973
7    43.1                     2.288  33.0      1.0   11/5/1965
..    ...                       ...   ...      ...         ...
793  32.9                     0.171  63.0      0.0    1/9/1973
794  36.8                     0.340  27.0      0.0    1/9/1973
795  26.2                     0.245  30.0      0.0    1/9/1973
796  30.1                     0.349  47.0      1.0    1/9/1973
797  30.4                     0.315  23.0      0.0    1/9/1973
```

```
[ ] df.drop('DOB', axis=1, inplace=True)
```

```
[ ] print("Dataset after dropping DOB Column",df)
```

```
Dataset after dropping DOB Column      Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  \
2                          6.0     148.0          72.0           35.0      0.0
3                          1.0      85.0          66.0           29.0      0.0
4                          8.0     183.0          64.0            0.0      0.0
6                          1.0      89.0          66.0           23.0     94.0
7                          0.0     137.0          40.0           35.0    168.0
..                         ...       ...           ...            ...      ...
793                       10.0     101.0          76.0           48.0    180.0
794                        2.0     122.0          70.0           27.0      0.0
795                        5.0     121.0          72.0           23.0    112.0
796                        1.0     126.0          60.0            0.0      0.0
797                        1.0      93.0          70.0           31.0      0.0

      BMI  DiabetesPedigreeFunction   Age  Outcome
2    33.6                     0.627  50.0      1.0
3    26.6                     0.351  31.0      0.0
4    23.3                     0.672  32.0      1.0
6    28.1                     0.167  21.0      0.0
7    43.1                     2.288  33.0      1.0
..    ...                       ...   ...      ...
793  32.9                     0.171  63.0      0.0
```

* Filtering

```
[22] #Average Age of Diabities Patient
     average_age = df['Age'].mean()

     print("Average Age of Patient:", average_age)
```

```
Average Age of Patient: 33.240885416666664
```

```
[23] max_age = df['Age'].max()
     min_age = df['Age'].min()

     # Display the results
     print("Maximum Age of Diabities an Patient:", max_age)
     print("Minimum Age of Diabities an Patient:", min_age)
```

```
Maximum Age of Diabities an Patient: 81.0
Minimum Age of Diabities an Patient: 21.0
```

```
mode_age = df['Outcome'].mode()

# Display the most frequent value
print("Most frequent Outcome:", mode_age.values[0])
```

```
Most frequent Outcome: 0.0
```

* Sorting

```
[30] df_sorted = df.apply(lambda x: x.sort_values().values)

     print("\nDataFrame with columns sorted in ascending order:")
     print(df_sorted)
```

```
DataFrame with columns sorted in ascending order:
     Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  \
2                        0      0.0            0.0            0.0      0.0
3                        0      0.0            0.0            0.0      0.0
4                        0      0.0            0.0            0.0      0.0
6                        0      0.0            0.0            0.0      0.0
7                        0      0.0            0.0            0.0      0.0
..                     ...      ...            ...            ...      ...
793                     13    197.0          110.0           54.0    579.0
794                     14    197.0          110.0           56.0    600.0
795                     14    197.0          110.0           60.0    680.0
796                     15    198.0          114.0           63.0    744.0
797                     17    199.0          122.0           99.0    846.0

      BMI  DiabetesPedigreeFunction  Age Outcome
2     0.0                     0.078   21     0.0
3     0.0                     0.084   21     0.0
4     0.0                     0.085   21     0.0
6     0.0                     0.085   21     0.0
7     0.0                     0.088   21     0.0
```

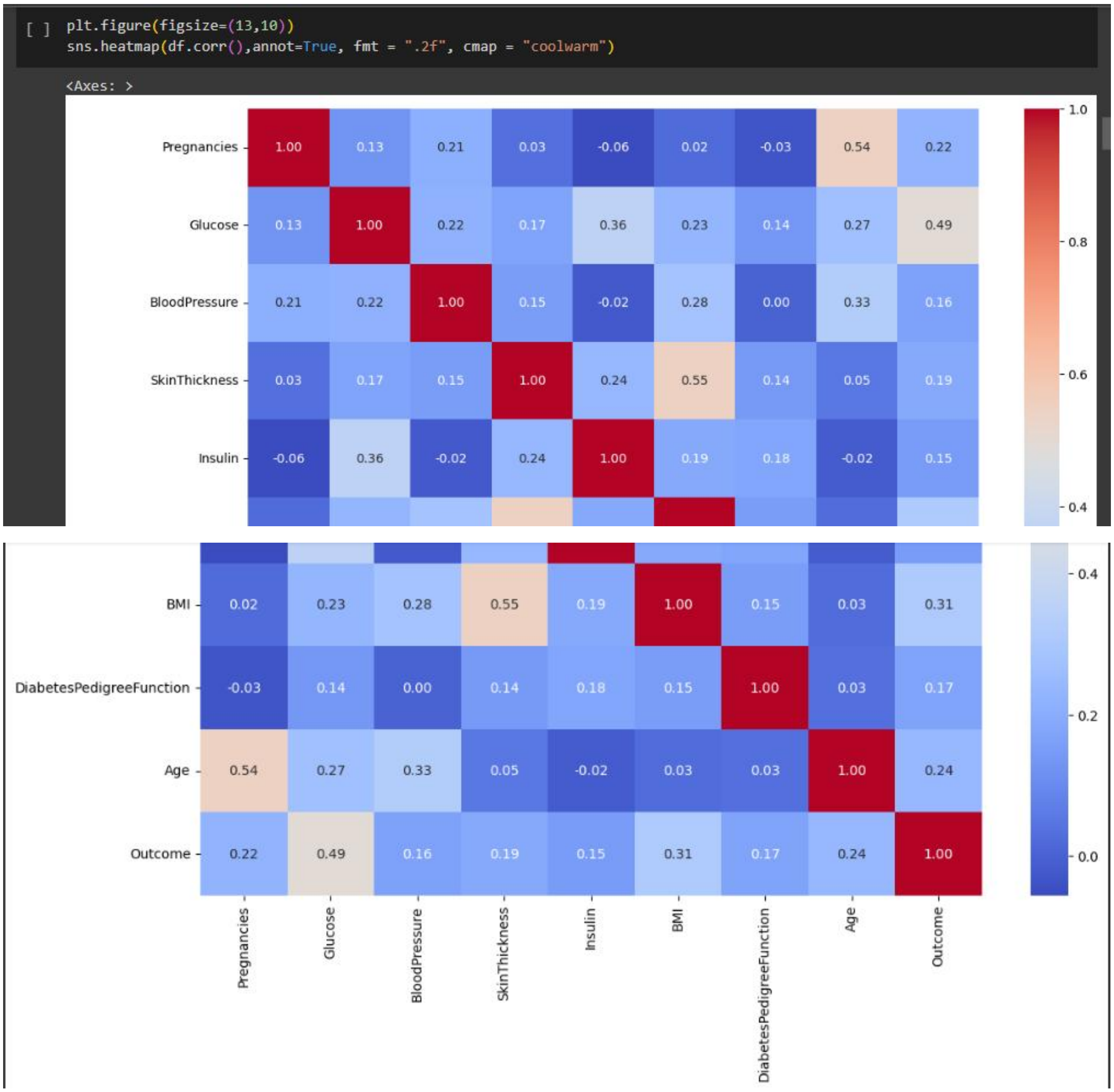
* Saving filtered data set

```
[32] # output_file_path = 'Desktop/modified_dataset.csv'
     df.to_csv("C:\\Users\\HP\\Desktop\\filter_dataset.csv")
     df=pd.read_csv("C:\\Users\\HP\\Desktop\\filter_dataset.csv")
     print(df)
```

```
     Unnamed: 0  Number_of_Pregnancies  Glucose  BloodPressure  SkinThickness  \
0             2                      6    148.0           72.0           35.0
1             3                      1     85.0           66.0           29.0
2             4                      8    183.0           64.0            0.0
3             6                      1     89.0           66.0           23.0
4             7                      0    137.0           40.0           35.0
..          ...                    ...      ...            ...            ...
763         793                     10    101.0           76.0           48.0
764         794                      2    122.0           70.0           27.0
765         795                      5    121.0           72.0           23.0
766         796                      1    126.0           60.0            0.0
767         797                      1     93.0           70.0           31.0

     Insulin   BMI  DiabetesPedigreeFunction  Age  Outcome
0        0.0  33.6                     0.627   50      1.0
1        0.0  26.6                     0.351   31      0.0
2        0.0  23.3                     0.672   32      1.0
3       94.0  28.1                     0.167   21      0.0
4      168.0  43.1                     2.288   33      1.0
..       ...   ...                       ...  ...      ...
763    180.0  32.9                     0.171   63      0.0
764      0.0  36.8                     0.340   27      0.0
```

Data Visualization

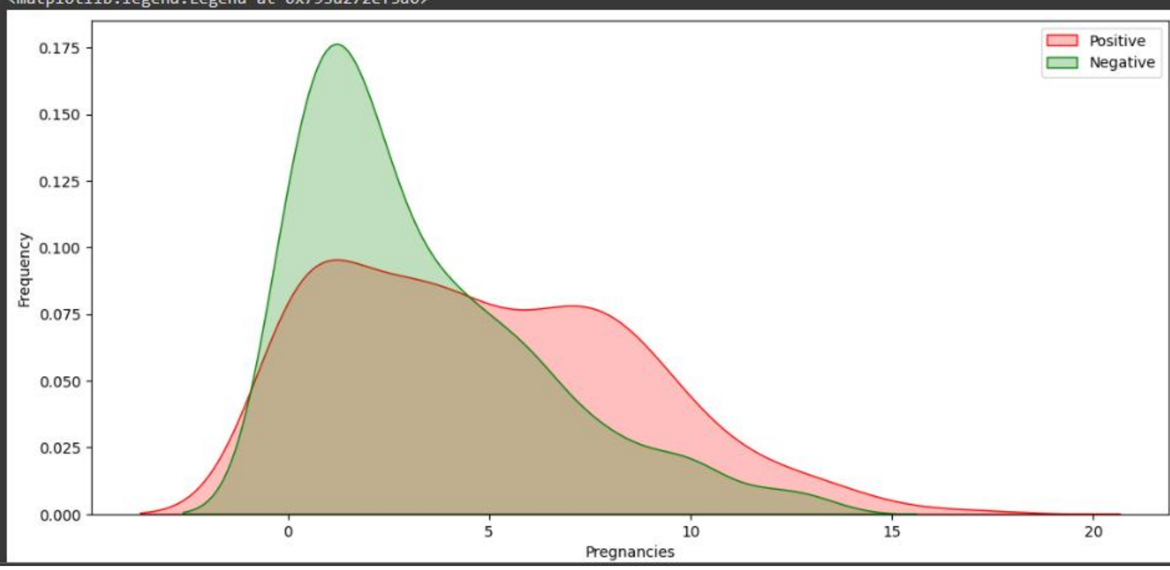
In this part we have visualize data by showing relation between columns.



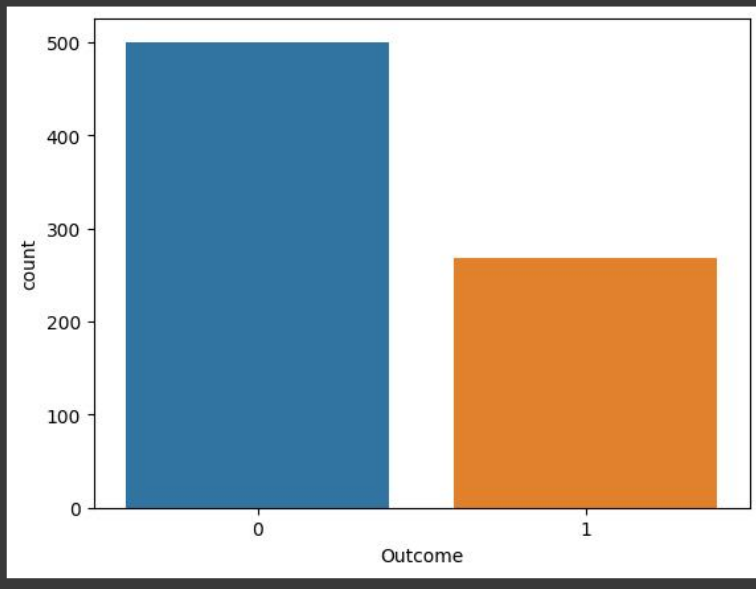
```
[ ] # Explore Pregnancies vs Outcome
    plt.figure(figsize=(13,6))
    g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1], color="Red", shade = True)
    g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
    g.set_xlabel("Pregnancies")
    g.set_ylabel("Frequency")
    g.legend(["Positive","Negative"])
```

```
<ipython-input-9-d86aa68bbb10>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1], color="Red", shade = True)
<ipython-input-9-d86aa68bbb10>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
<matplotlib.legend.Legend at 0x793d272ef3d0>
```

```
[ ]  g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
<matplotlib.legend.Legend at 0x793d272ef3d0>
```

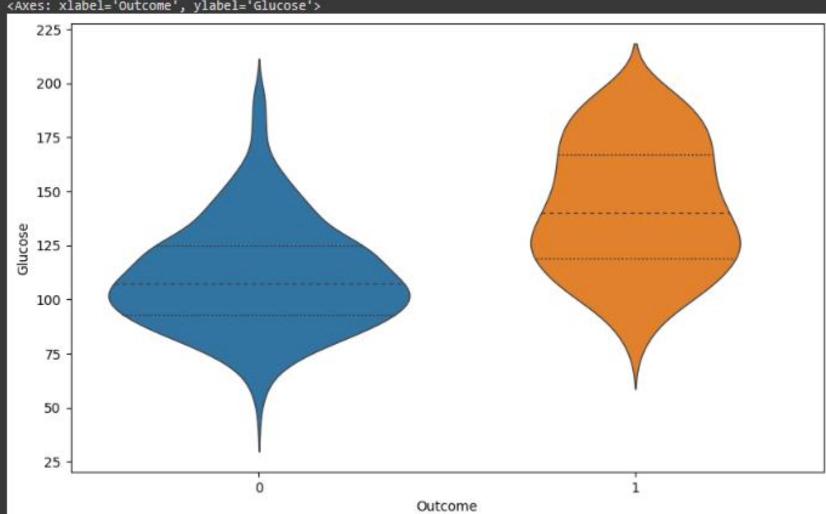


```
[ ] sns.countplot(x='Outcome', data=df)

    # Show the plot
    plt.show()
```




```
# Glucose against Outcome
plt.figure(figsize=(10,6))
sns.violinplot(data=df, x="Outcome", y="Glucose",
               split=True, inner="quart", linewidth=1)
```

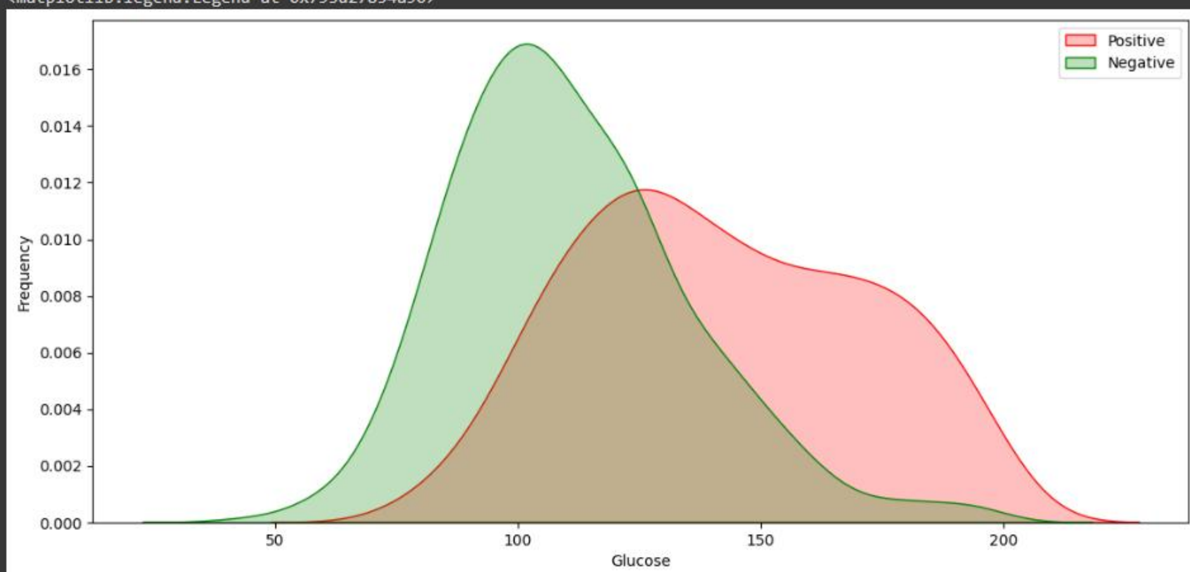
```
<Axes: xlabel='Outcome', ylabel='Glucose'>
```



```python
plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
g.set_xlabel("Glucose")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```

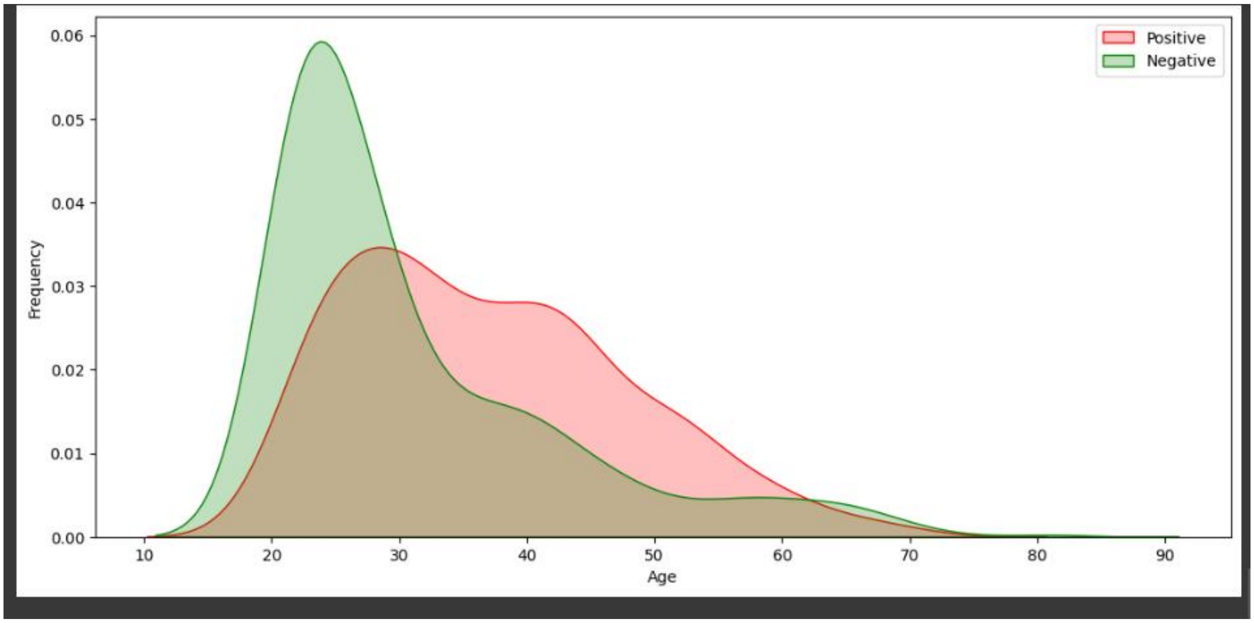
```
<ipython-input-13-02dee0bca3f2>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)
<ipython-input-13-02dee0bca3f2>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
<matplotlib.legend.Legend at 0x793d27854a90>
```

```
  g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
<matplotlib.legend.Legend at 0x793d27854a90>
```



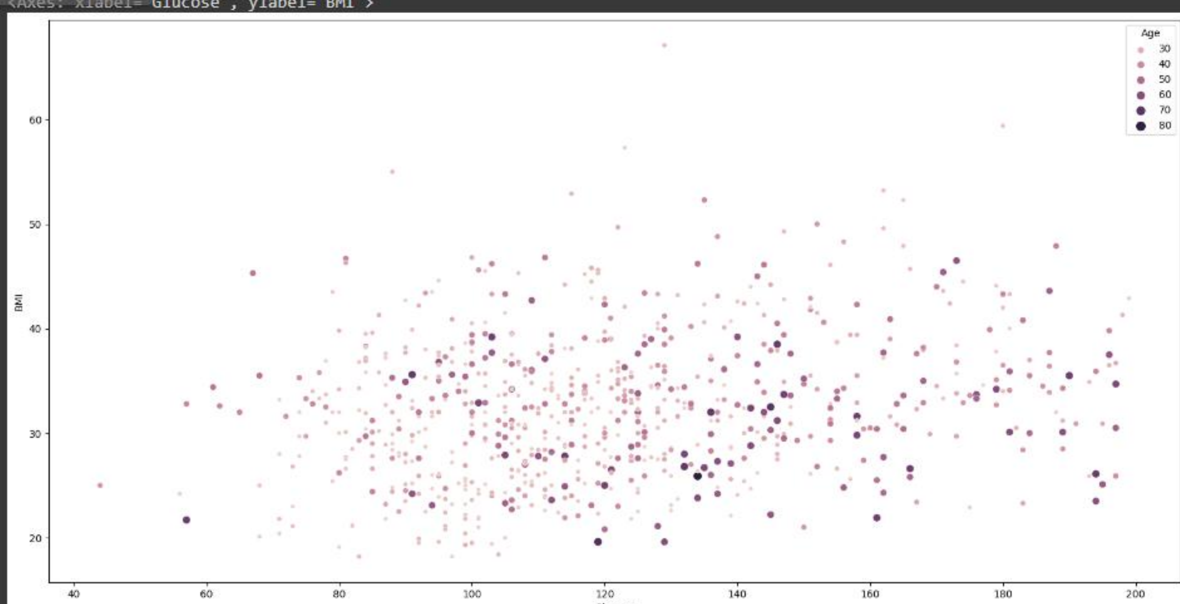

```python
# Explore Pregnancies vs Outcome
plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Age"][df["Outcome"] == 1], color="Red", shade = True)
g = sns.kdeplot(df["Age"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
g.set_xlabel("Age")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```

```
<ipython-input-14-49db55352beb>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Age"][df["Outcome"] == 1], color="Red", shade = True)
<ipython-input-14-49db55352beb>:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["Age"][df["Outcome"] == 0], ax =g, color="Green", shade= True)
<matplotlib.legend.Legend at 0x793d276ee1d0>
```




```
plt.figure(figsize=(20,10))
sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")
```

```
<Axes: xlabel='Glucose', ylabel='BMI'>
```



Outliers Detection

The reason behind removing outliers detection is that it can effect accuracy of an algorithm.

Outliers Detection

```
def detect_outliers(df,n,features):
    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col],75)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
```

```
    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction',
```

```
[ ]  df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

Data Modelling

In modeling part we have evaluated various machine learning models.

Modeling

```
[ ] # Data Transformation
    q  = QuantileTransformer()
    X = q.fit_transform(df)
    transformedDF = q.transform(X)
    transformedDF = pd.DataFrame(X)
    transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFu
    # Show top 5 rows
    transformedDF.head()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater thai
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but Quantile
  warnings.warn(
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.747718	0.810300	0.516949	0.801825	0.000000	0.591265	0.750978	0.889831	1.0
1	0.232725	0.097784	0.336375	0.644720	0.000000	0.227510	0.475880	0.558670	0.0
2	0.863755	0.956975	0.279009	0.000000	0.000000	0.091917	0.782269	0.585398	1.0
3	0.232725	0.131030	0.336375	0.505867	0.662973	0.298566	0.106258	0.000000	0.0

Data Split

```
[ ] features = df.drop(["Outcome"], axis=1)
    labels = df["Outcome"]
    x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)
```

MODEL EVALUATIONS

Model Evaluations

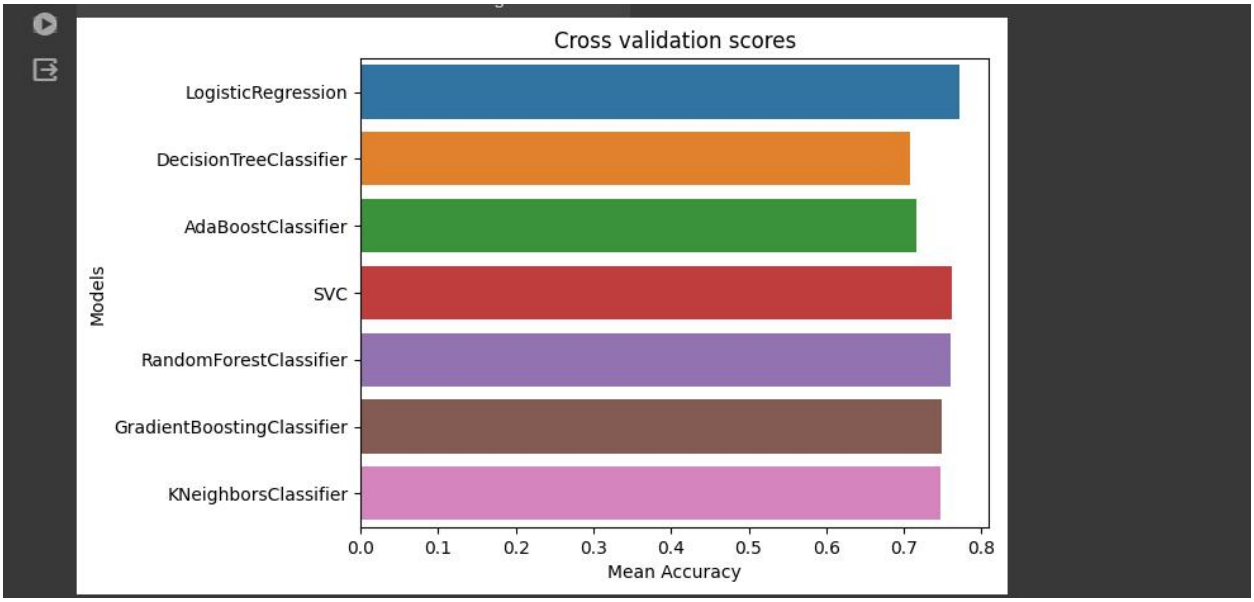
```
[ ] def evaluate_model(models):

        kfold = StratifiedKFold(n_splits = 10)

        result = []
        for model in models :
            result.append(cross_val_score(estimator = model, X = x_train, y = y_train, scoring = "accuracy", cv = kfold, n_

        cv_means = []
        cv_std = []
        for cv_result in result:
            cv_means.append(cv_result.mean())
            cv_std.append(cv_result.std())

        result_df = pd.DataFrame({
            "CrossValMeans":cv_means,
            "CrossValerrors": cv_std,
            "Models":[
                "LogisticRegression",
                "DecisionTreeClassifier",
                "AdaBoostClassifier",
```

```
[ ]             "GradientBoostingClassifier",
                "KNeighborsClassifier"
            ]
        })

        # Generate chart
        bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df, orient = "h")
        bar.set_xlabel("Mean Accuracy")
        bar.set_title("Cross validation scores")
        return result_df
```

```
[ ] # Modeling step Test differents algorithms
    random_state = 30
    models = [
        LogisticRegression(random_state = random_state, solver='liblinear'),
        DecisionTreeClassifier(random_state = random_state),
        AdaBoostClassifier(DecisionTreeClassifier(random_state = random_state), random_state = random_state, learning_rat
        SVC(random_state = random_state),
        RandomForestClassifier(random_state = random_state),
        GradientBoostingClassifier(random_state = random_state),
        KNeighborsClassifier(),
    ]
    evaluate_model(models)
```

[ ]

	CrossValMeans	CrossValerrors	Models
0	0.770964	0.058524	LogisticRegression
1	0.707687	0.065109	DecisionTreeClassifier
2	0.717016	0.061262	AdaBoostClassifier
3	0.761670	0.033724	SVC
4	0.759748	0.055512	RandomForestClassifier
5	0.748532	0.065303	GradientBoostingClassifier
6	0.746506	0.054171	KNeighborsClassifier




```
[ ] from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import classification_report
    def analyze_grid_result(grid_result):
        '''
        Analysis of GridCV result and predicting with test dataset
        Show classification report at last
        '''
        # Best parameters and accuracy
        print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
        print("Accuracy :", grid_result.best_score_)

        means = grid_result.cv_results_["mean_test_score"]
        stds = grid_result.cv_results_["std_test_score"]
        for mean, std, params in zip(means, stds, grid_result.cv_results_["params"]):
            print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
        print()
        print("Detailed classification report:")
        y_true, y_pred = y_test, grid_result.predict(x_test)
        print(classification_report(y_true, y_pred))
        print()
```

Logistic Regression

```
[ ] # Define models and parameters for LogisticRegression
    model = LogisticRegression(solver='liblinear')
    solvers = ['newton-cg', 'liblinear']
    penalty = ['l2']
    c_values = [100, 10, 1.0, 0.1, 0.01]
    # Define grid search
    grid = dict(solver = solvers, penalty = penalty, C = c_values)
    cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
    grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv, scoring = 'accuracy', error_score = 0)
    logi_result = grid_search.fit(x_train, y_train)
    # Logistic Regression Hyperparameter Result
    analyze_grid_result(logi_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy : 0.774909090909091
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.241) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.775 (+/-0.226) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.240) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.224) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.242) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.720 (+/-0.225) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.245) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.687 (+/-0.256) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

SVC

```
[ ] # Define models and parameters for LogisticRegression
    model = SVC()
    # Define grid search
    tuned_parameters = [
        {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
        {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
    ]
    cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
    grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring = 'accuracy', error_score =
    scv_result = grid_search.fit(x_train, y_train)
    # SVC Hyperparameter Result
    analyze_grid_result(scv_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 100, 'kernel': 'linear'}
Accuracy : 0.7765286023414526
0.715 (+/-0.005) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.747 (+/-0.023) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.670 (+/-0.020) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.737 (+/-0.003) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.663 (+/-0.005) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.728 (+/-0.036) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.648 (+/-0.002) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.680 (+/-0.031) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
0.762 (+/-0.001) for {'C': 1, 'kernel': 'linear'}
0.769 (+/-0.021) for {'C': 10, 'kernel': 'linear'}
0.777 (+/-0.008) for {'C': 100, 'kernel': 'linear'}
0.764 (+/-0.003) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       147
           1       0.70      0.60      0.65        84

    accuracy                           0.76       231
   macro avg       0.75      0.73      0.73       231
weighted avg       0.76      0.76      0.76       231
```

Prediction

Prediction

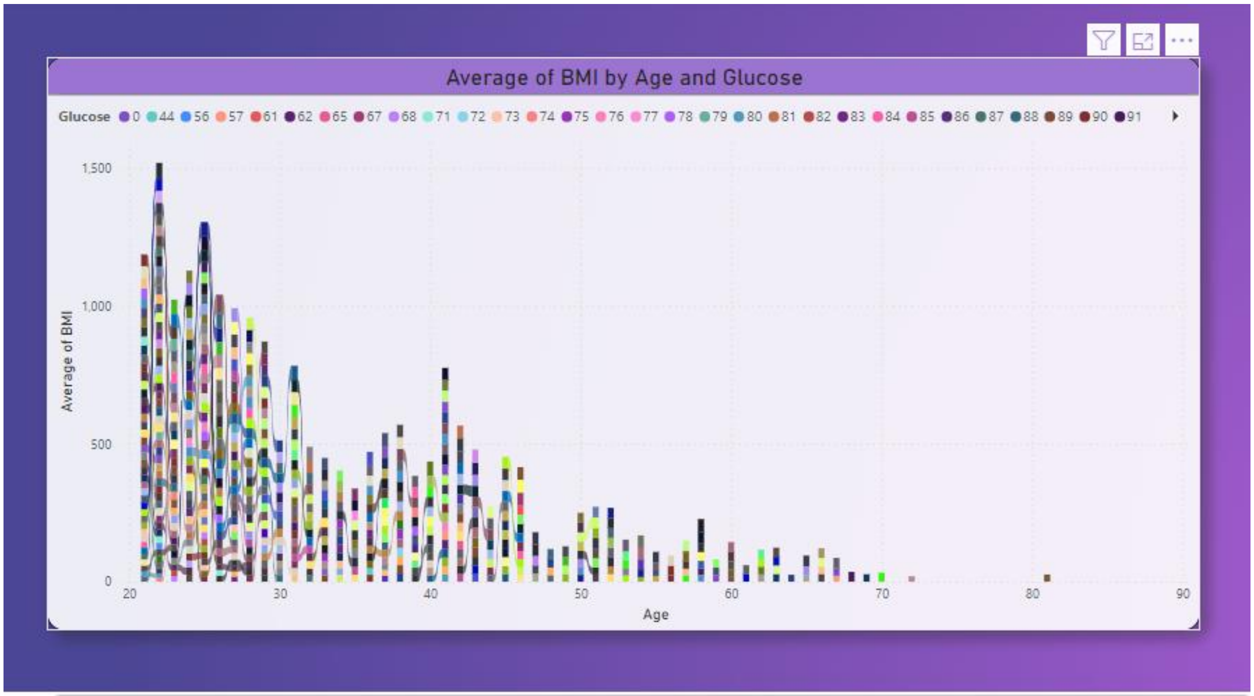
```
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

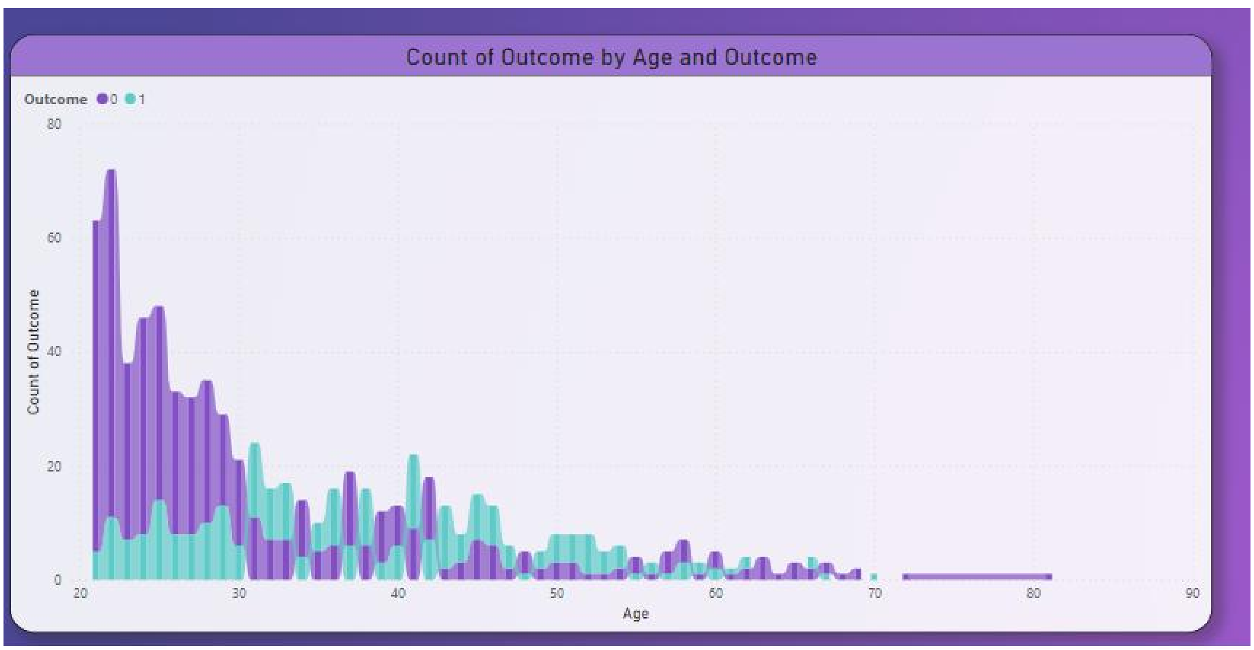
```
x_test['pred'] = y_pred
print(x_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
353            1       90             62             12       43  27.2
236            7      181             84             21      192  35.9
323           13      152             90             33       29  26.8
98             6       93             50             30       64  28.7
701            6      125             78             31        0  27.6
..           ...      ...            ...            ...      ...   ...
188            8      109             76             39      114  27.9
351            4      137             84              0        0  31.2
120            0      162             76             56      100  53.2
108            3       83             58             31       18  34.3
616            6      117             96              0        0  28.7

     DiabetesPedigreeFunction  Age  pred
353                     0.580   24     0
236                     0.586   51     1
323                     0.731   43     1
98                      0.356   23     0
701                     0.565   49     0
..                        ...  ...   ...
188                     0.640   31     0
351                     0.252   30     0
120                     0.759   25     1
108                     0.336   25     0
616                     0.157   30     0

[231 rows x 9 columns]
```

Power Bi Dashboard



Average of BMI by Age and Glucose
Glucose 0 44 56 57 61 62 65 67 68 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
1,500
1,000
500
0
Average of BMI
20
30
40
50
60
70
80
90
Age



Count of Outcome by Age and Outcome
Outcome 0 1
80
60
40
20
0
Count of Outcome
20
30
40
50
60
70
80
90
Age

Project Scope

Project scope is very vast like you can use it in various ways like for disease prediction, analysis. .Suggestions for future work may include exploring additional features, refining the model, and incorporating real-time data.

Conclusion

The project successfully developed a predictive model for diabetes analysis.Insights gained from the project can contribute to early intervention and improved patient outcomes.

References

https://www.kaggle.com/datasets