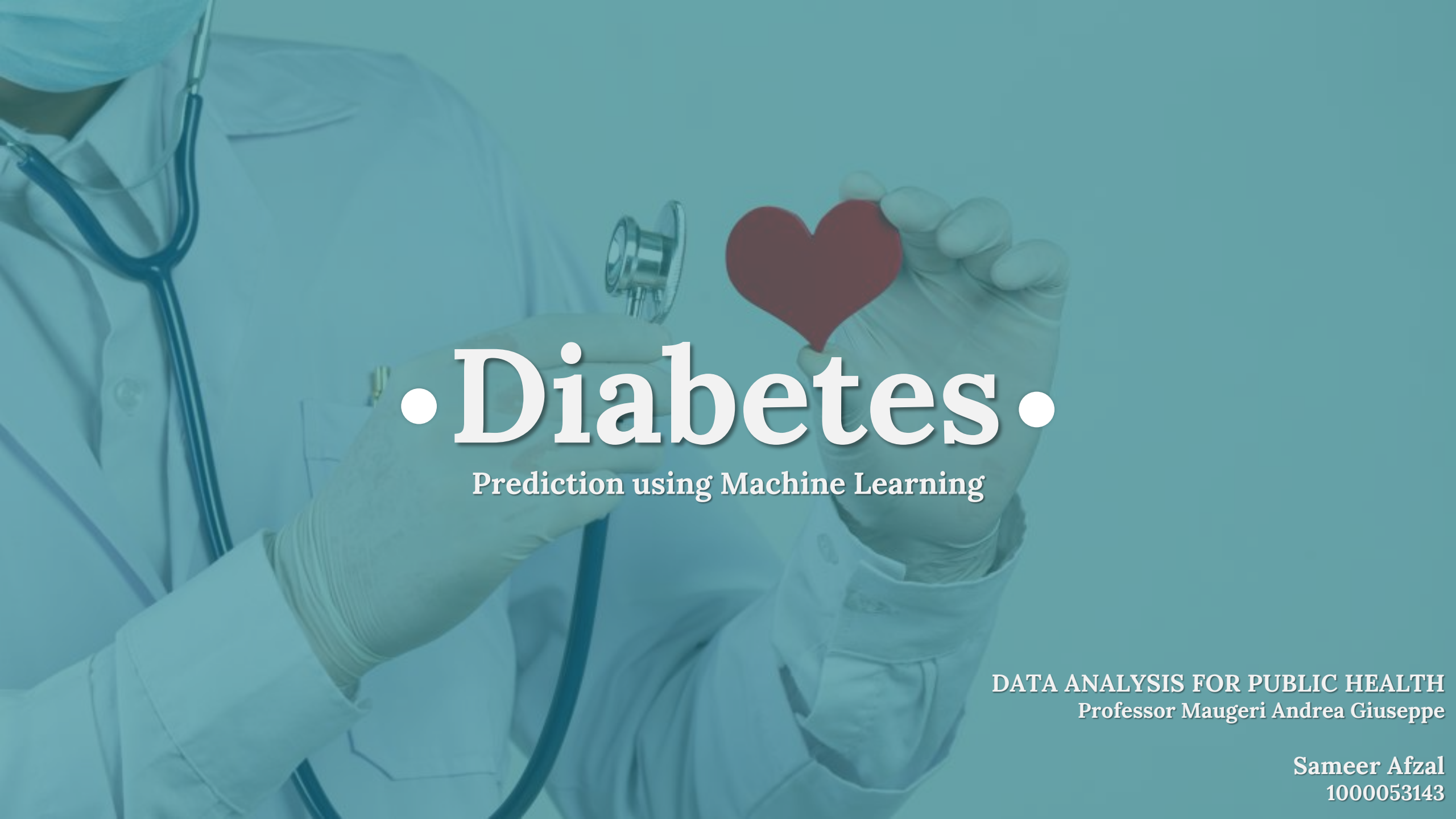


Diabetes

Prediction using Machine Learning

DATA ANALYSIS FOR PUBLIC HEALTH
Professor Maugeri Andrea Giuseppe

Sameer Afzal
1000053143

Motivations in Predicting Diabetes

Early Diagnosis & Intervention: Catching diabetes early for better outcomes.

Personalized Care: Tailored plans based on individual risk factors. Resource

Allocation: Targeting high-risk populations for efficient care.

Public Health Planning: Predicting future trends to guide policy and programs.

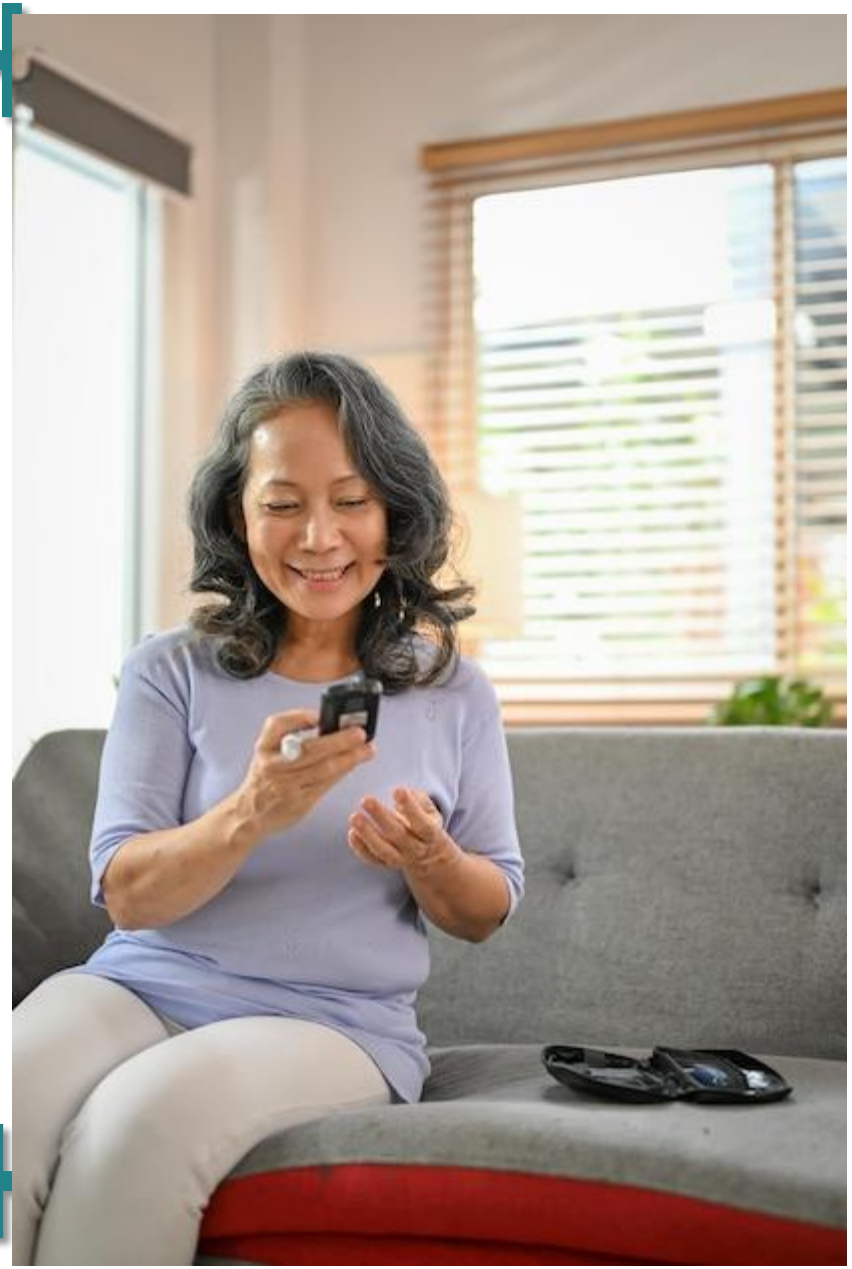
Research & Drug Development: Identifying targets for new treatments.

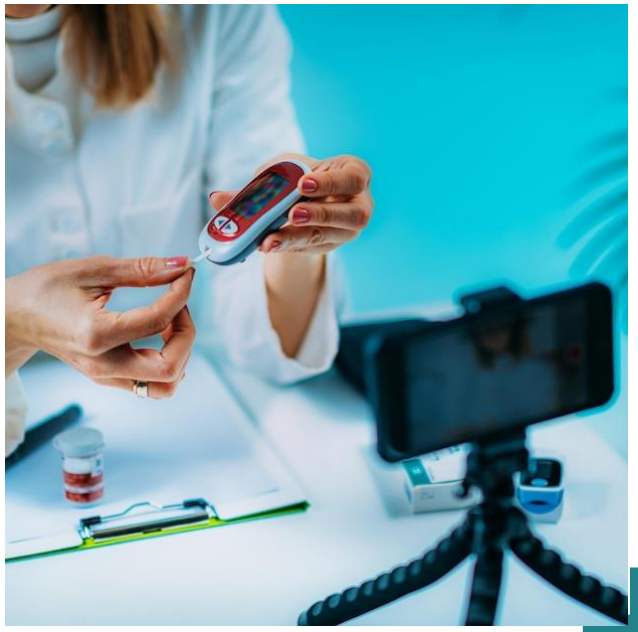
Empowering Individuals: Taking control of health with risk assessments.

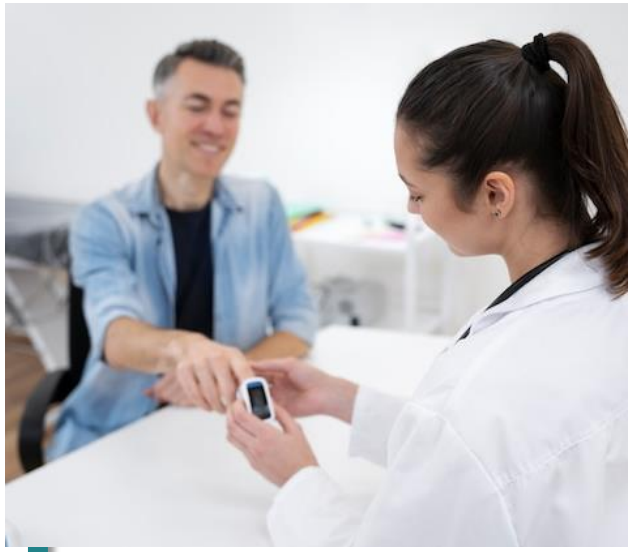
Economic Impact: Reducing healthcare costs through better management.

About Dataset

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







Import Dataset


```
df.head()
```

✓ 0.0s

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

```
df.tail()
```

✓ 0.0s

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

Dataset Content

```
df.info()
```

✓ 0.0s

```
<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
```

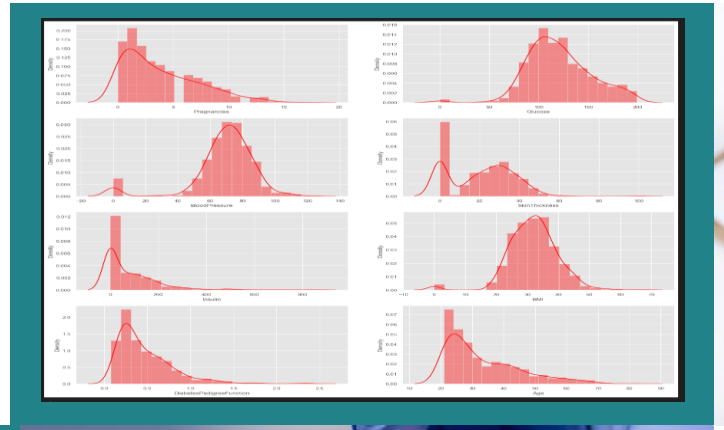
```
df.columns
```

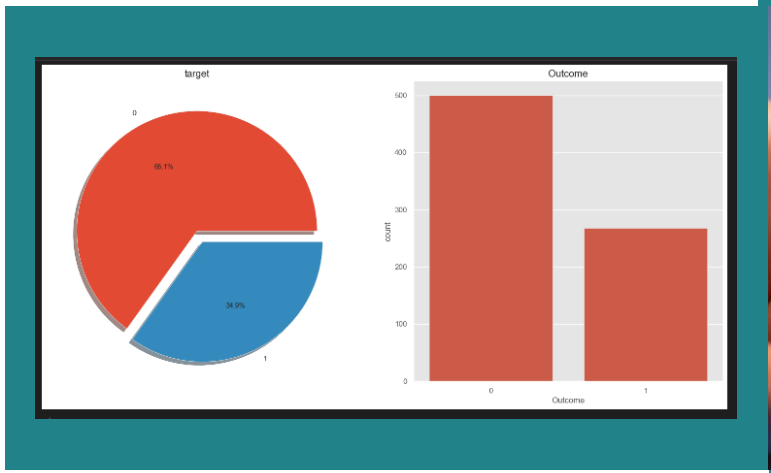
✓ 0.0s

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

- **Pregnancies**: Number of times pregnant.
- **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
- **Blood Pressure**: Diastolic blood pressure (mm Hg).
- **Skin Thickness**: Triceps skin fold thickness (mm).
- **Insulin**: 2-Hour serum insulin (mu U/ml).
- **BMI**: Body mass index (weight in kg/(height in m)^2).
- **DiabetesPedigreeFunction :** Diabetes pedigree function
- **Age**: Age (years).
- **Outcome**: '1' denotes patient having diabetes and '0' denotes patient not having diabetes.

Exploratory Data Analysis



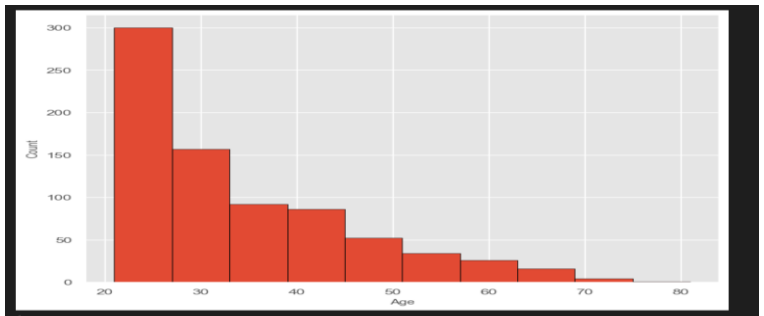


```
df.groupby("Outcome").agg({'Pregnancies':'max'})
```

Outcome	Pregnancies
0	13
1	17

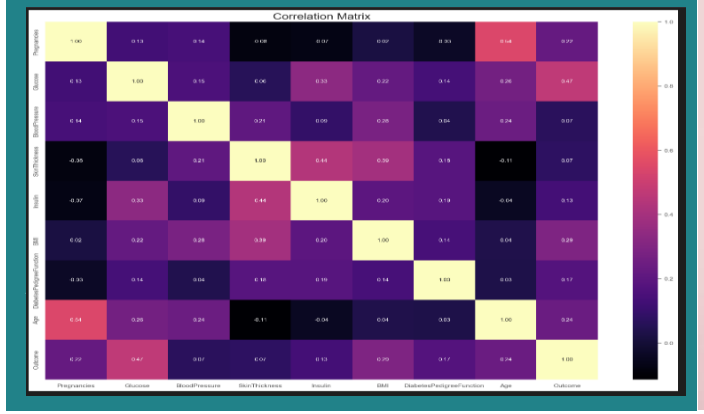
```
df.groupby("Outcome").agg({'Glucose':'max'})
```

Outcome	Glucose
0	197
1	199




```
print("MAX AGE: "+str(df['Age'].max()))
print("MIN AGE: "+str(df['Age'].min()))
```

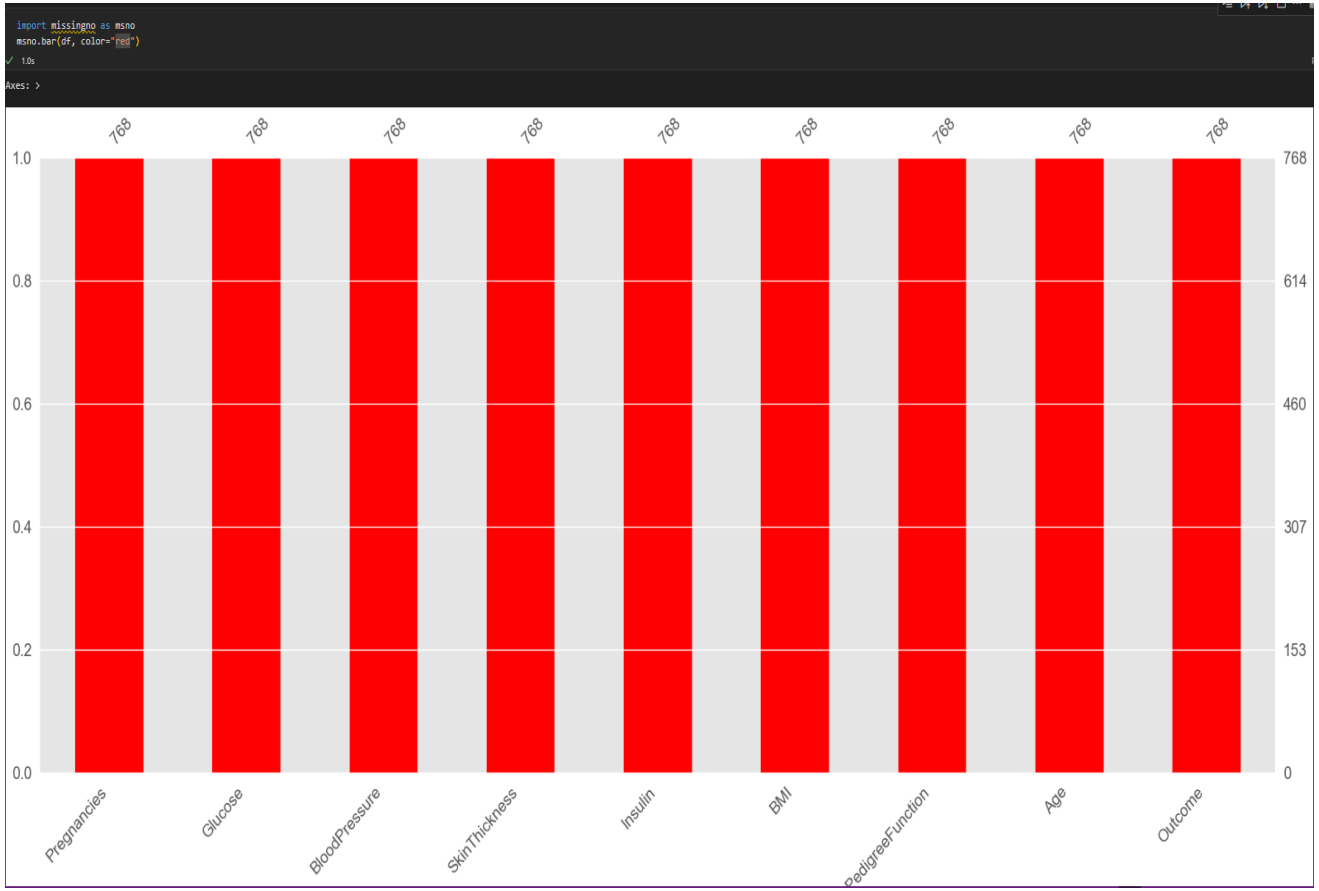
```
MAX AGE: 81
MIN AGE: 21
```



Data Preprocessing

1. Missing Observation Analysis


```
import missingno as msno
msno.bar(df, color="red")
```



2. Outlier Observation Analysis

Data Preprocessing

3. Local Outlier Factor (LOF)

```
# LOF
# local outlier factor
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(n_neighbors=10)
lof.fit_predict(df)
```

```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
...
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1])
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Data Preprocessing

Feature Engineering

Creating new variables is important for models. But you need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

BMI

```
    NewBMI

0    Underweight
1         Normal
2     Overweight
3      Obesity 1
4      Obesity 2
5      Obesity 3
dtype: category
Categories (6, object): ['Normal', 'Obesity 1', 'Obesity 2', 'Obesity 3', 'Overweight', 'Underweight']
```

```
    df['NewBMI'] = NewBMI
    df.loc[df["BMI"]<18.5, "NewBMI"] = NewBMI[0]
    df.loc[(df["BMI"]>18.5) & df["BMI"]<=24.9, "NewBMI"] = NewBMI[1]
    df.loc[(df["BMI"]>24.9) & df["BMI"]<=29.9, "NewBMI"] = NewBMI[2]
    df.loc[(df["BMI"]>29.9) & df["BMI"]<=34.9, "NewBMI"] = NewBMI[3]
    df.loc[(df["BMI"]>34.9) & df["BMI"]<=39.9, "NewBMI"] = NewBMI[4]
    df.loc[df["BMI"]>39.9, "NewBMI"] = NewBMI[5]
```

Feature Engineering

Creating new variables is important for models. But you need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

Insulin

```
# if insulin>=16 & insuline<=166->normal
def set_insuline(row):
    if row["Insulin"]>=16 and row["Insulin"]<=166:
        return "Normal"
    else:
        return "Abnormal"
```

```
df = df.assign(NewInsulinScore=df.apply(set_insuline, axis=1))
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore
0	6.0	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 2	Abnormal
1	1.0	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Obesity 2	Normal
2	8.0	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Obesity 2	Abnormal
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Obesity 2	Normal
4	5.0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3	Abnormal

Feature Engineering

Creating new variables is important for models. But you need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

Glucose

```
# Some intervals were determined according to the glucose variable and these were assigned categorical variables.
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype = "category")
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose[1]
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucose[2]
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore	NewGlucose
0	6.0	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 2	Abnormal	Secret
1	1.0	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Obesity 2	Normal	Normal
2	8.0	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Obesity 2	Abnormal	Secret
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Obesity 2	Normal	Normal
4	5.0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3	Abnormal	Secret

One Hot Encoding

Categorical variables in the data set should be converted into numerical values.
For this reason, these transformation processes are performed with Label Encoding and One Hot Encoding method.

by making One Hot Encoding transformation, categorical variables were converted into numerical values. It is also protected from the Dummy variable trap.

```
# One hot encoding
df = pd.get_dummies(df, columns = ["NewBMI", "NewInsulinScore", "NewGlucose"], drop_first=True)
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI_Underweight	NewInsulinScore_Normal	NewGlucose_Low	NewGlucose_Normal	NewGlucose_Overweight	NewGlucose_Secret
0	6.0	148.0	72.0	35.0	169.5	33.6	0.627	50	1	False	True	False	False	False	False	False	False	False	True
1	1.0	85.0	66.0	29.0	102.5	26.6	0.351	31	0	False	True	False	False	False	True	False	True	False	False
2	8.0	183.0	64.0	32.0	169.5	23.3	0.672	32	1	False	True	False	False	False	False	False	False	False	True
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21	0	False	True	False	False	False	True	False	True	False	False
4	5.0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	False	False	True	False	False	False	False	False	False	True

One Hot Encoding

Categorical variables in the data set should be converted into numerical values.
For this reason, these transformation processes are performed with Label Encoding and One Hot Encoding method.

Make new data frame in which all newly created variable are present

```
categorical_df = df[['NewBMI_Obesity 1',
       'NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight',
       'NewBMI_Underweight', 'NewInsulinScore_Normal', 'NewGlucose_Low',
       'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret']]
```

```
categorical_df.head()
```

	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI_Underweight	NewInsulinScore_Normal	NewGlucose_Low	NewGlucose_Normal	NewGlucose_Overweight	NewGlucose_Secret
0	False	True	False	False	False	False	False	False	False	True
1	False	True	False	False	False	True	False	True	False	False
2	False	True	False	False	False	False	False	False	False	True
3	False	True	False	False	False	True	False	True	False	False
4	False	False	True	False	False	False	False	False	False	True

One Hot Encoding

Categorical variables in the data set should be converted into numerical values.
For this reason, these transformation processes are performed with Label Encoding and One Hot Encoding method.

The variables in the data set are an effective factor in increasing the performance of the models by standardization. There are multiple standardization methods. These are methods such as " Normalize"," MinMax"," Robust" and "Scale".

```
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
X=transformer.transform(X)
X=pd.DataFrame(X, columns = cols, index = index)
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.75	0.775	0.000	1.000000	1.000000	0.177778	0.669707	1.235294
1	-0.50	-0.800	-0.375	0.142857	0.000000	-0.600000	-0.049511	0.117647
2	1.25	1.650	-0.500	0.571429	1.000000	-0.966667	0.786971	0.176471
3	-0.50	-0.700	-0.375	-0.714286	-0.126866	-0.433333	-0.528990	-0.470588
4	0.50	0.500	-2.000	1.000000	0.977612	1.233333	4.998046	0.235294

```
X = pd.concat([X, categorical_df], axis=1)
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI_Underweight	NewInsulinScore_Normal	NewGlucose_Low	NewGlucose_Normal	NewGlucose_Overweight	NewGlucose_Secret
0	0.75	0.775	0.000	1.000000	1.000000	0.177778	0.669707	1.235294	False	True	False	False	False	False	False	False	False	True
1	-0.50	-0.800	-0.375	0.142857	0.000000	-0.600000	-0.049511	0.117647	False	True	False	False	False	True	False	True	False	False
2	1.25	1.650	-0.500	0.571429	1.000000	-0.966667	0.786971	0.176471	False	True	False	False	False	False	False	False	False	True
3	-0.50	-0.700	-0.375	-0.714286	-0.126866	-0.433333	-0.528990	-0.470588	False	True	False	False	False	True	False	True	False	False
4	0.50	0.500	-2.000	1.000000	0.977612	1.233333	4.998046	0.235294	False	False	True	False	False	False	False	False	False	True

Splitting and Scaling Data for Modelling

Splitting Data (train_test_split):

- This code splits our data (features X and target variable y) into training and testing sets.
- test_size=0.2 allocates 20% of the data for testing and the remaining 80% for training.
- random_state=0 sets a seed for reproducibility (ensuring the same split each time you run the code).

Scaling Data (StandardScaler):

- A StandardScaler object is created (scaler).
- scaler.fit_transform(X_train) standardizes the training data (X_train).
    - This removes the mean and scales the features to unit variance.
- The same scaler (scaler) is then used to transform the testing data (X_test).
    - It applies the same mean and scaling factors learned from the training data.

```
X_train, X_test, y_train , y_test = train_test_split(X,y, test_size=0.2, random_state=0)
```

```
scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

```
confusion_matrix(y_test, y_pred)
```

```
array([[88, 10],
       [ 6, 48]], dtype=int64)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        98
           1       0.83      0.89      0.86        54

    accuracy                           0.89       152
   macro avg       0.88      0.89      0.89       152
weighted avg       0.90      0.89      0.90       152
```

Modelling

SVM

```
[[90  8]
 [ 6 48]]
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        98
           1       0.86      0.89      0.87        54

    accuracy                           0.91       152
   macro avg       0.90      0.90      0.90       152
weighted avg       0.91      0.91      0.91       152
```

Modelling

Decision Tree Classifier

```
[[87 11]
 [ 9 45]]
              precision    recall  f1-score   support

           0       0.91      0.89      0.90        98
           1       0.80      0.83      0.82        54

    accuracy                           0.87       152
   macro avg       0.85      0.86      0.86       152
weighted avg       0.87      0.87      0.87       152
```

Modelling

Model Comparison

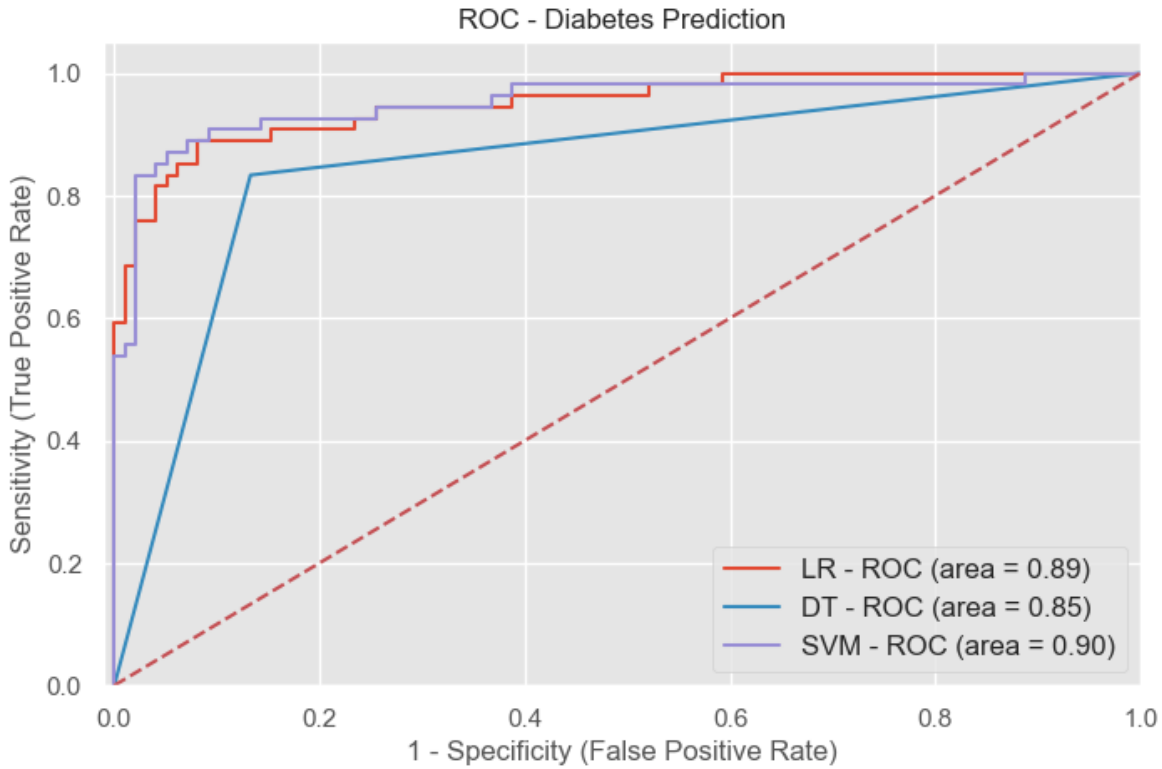
```python
# Model Comparison
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'SVM', 'Decision Tree Classifier'],
    'Score': [100*round(log_reg_acc,4), 100*round(svc_acc,4), 100*round(dt_acc,4)]
})
models.sort_values(by = 'Score', ascending = False)
```

✓ 0.0s

	Model	Score
1	SVM	90.79
0	Logistic Regression	89.47
2	Decision Tree Classifier	86.84

ROC

Receiver Operating Characteristic



Performance Evaluation