Importing Libraries and Data

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

```
df=pd.read_csv('/content/diabetes.csv')
df
```

	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
...	...	...	...	...	...	...	...	...	...
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

768 rows × 9 columns

EDL (Exploratory Data Analysis)

```
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
```

```
df.size
```

```
6912
```

```
df.shape
```

```
(768, 9)
```

```
df.columns
```

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

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Checking The Nulll Values

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

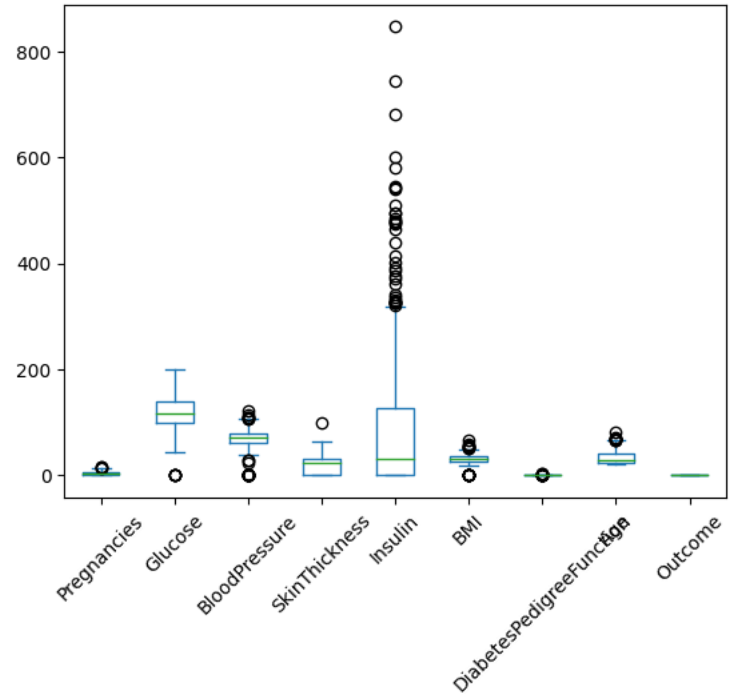
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
for col in df.describe():
  print(col)
  print(df[col].unique())
  print('-'*50)
```

```
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]
--------------------------------------------------
DiabetesPedigreeFunction
```

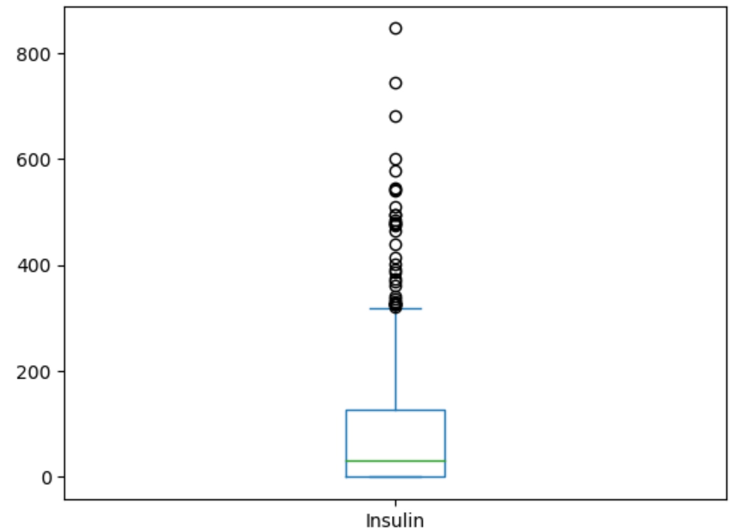
```
df.plot.box()
plt.xticks(rotation=45)
```

```
(array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(1, 0, 'Pregnancies'),
  Text(2, 0, 'Glucose'),
  Text(3, 0, 'BloodPressure'),
  Text(4, 0, 'SkinThickness'),
  Text(5, 0, 'Insulin'),
  Text(6, 0, 'BMI'),
  Text(7, 0, 'DiabetesPedigreeFunction'),
  Text(8, 0, 'Age'),
  Text(9, 0, 'Outcome')])
```



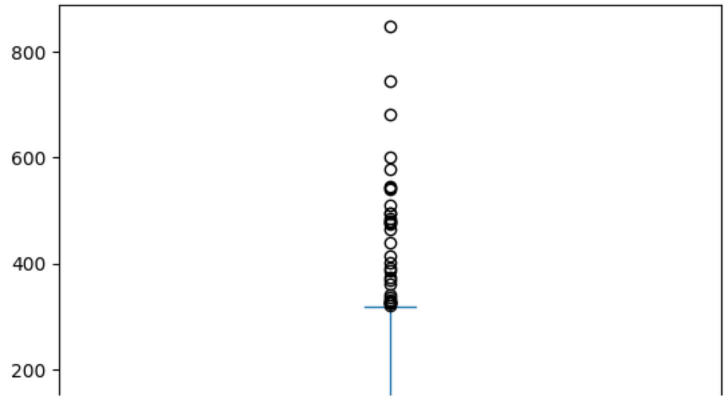

```
df['Insulin'].plot(kind='box')
```

```
<Axes: >
```



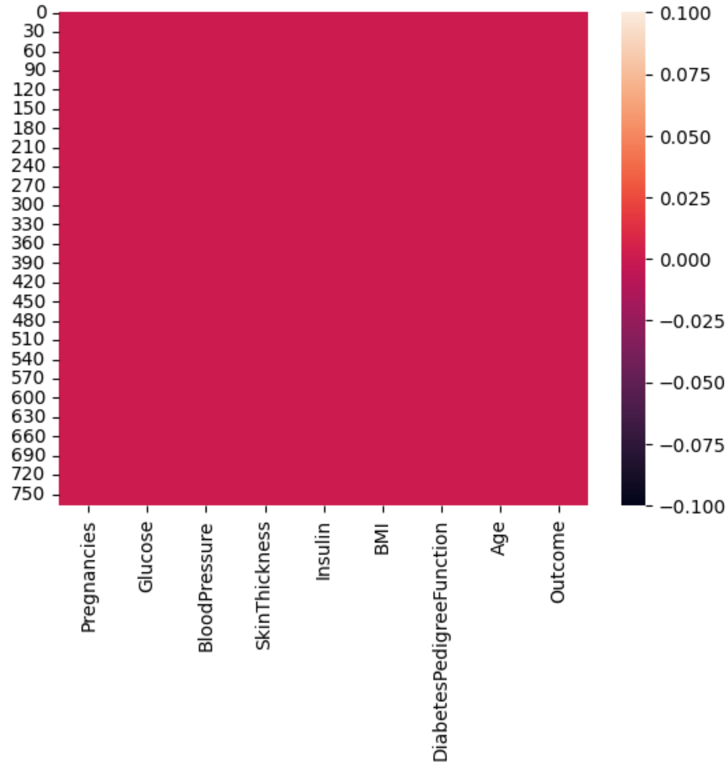

```
a=df['Insulin'].describe()
b=df['Insulin'].plot(kind='box')
print(a)
print(b)
```

```
count    768.000000
mean      79.799479
std      115.244002
min        0.000000
25%        0.000000
50%       30.500000
75%      127.250000
max      846.000000
Name: Insulin, dtype: float64
Axes(0.125,0.11;0.775x0.77)
```




```
sns.heatmap(df.isnull())
```

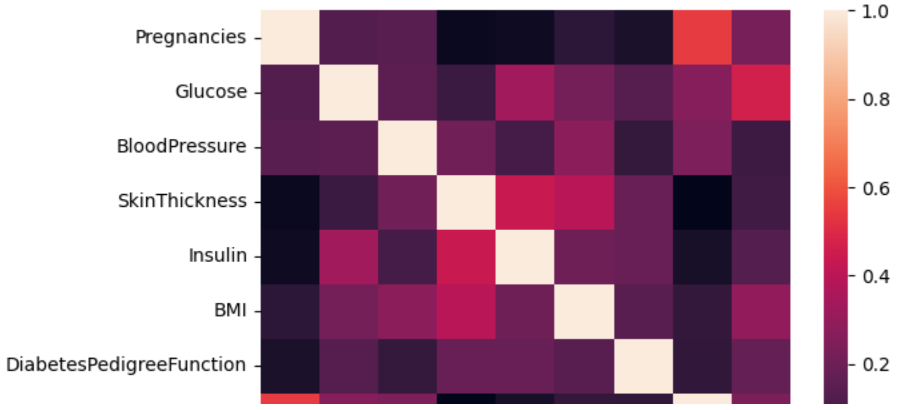
```
<Axes: >
```



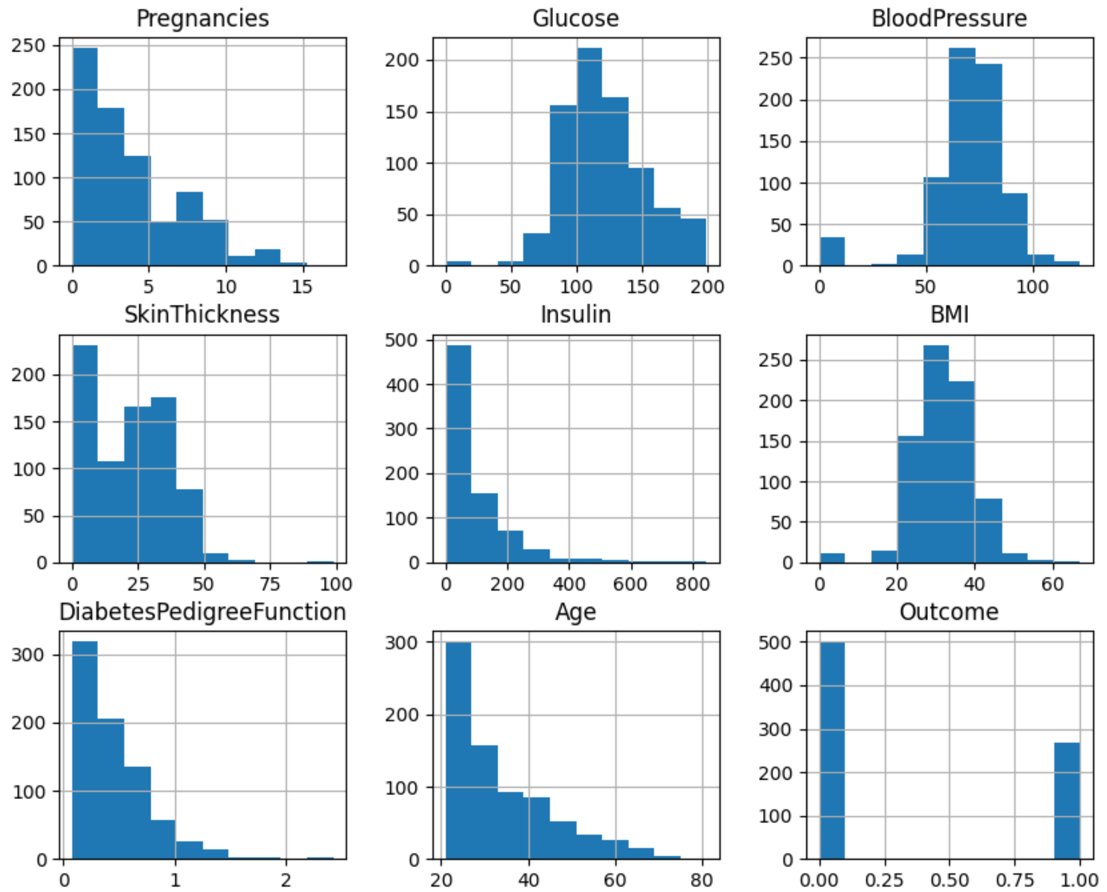
Correlation Matrix

```
df.corr()
sns.heatmap(df.corr())
```

```
<Axes: >
```




```
df.hist(figsize=(10,8))
plt.show()
```

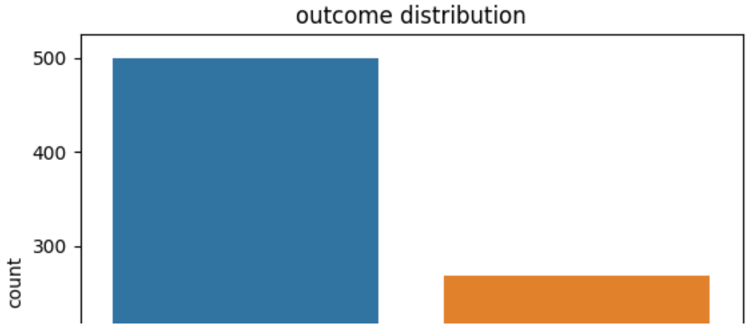



```
a=df['Outcome'].value_counts()
a
```

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

```
sns.countplot(data=df,x='Outcome')
plt.xlabel('Outcome')
plt.ylabel('count')
plt.title('outcome distribution')
```

```
Text(0.5, 1.0, 'outcome distribution')
```



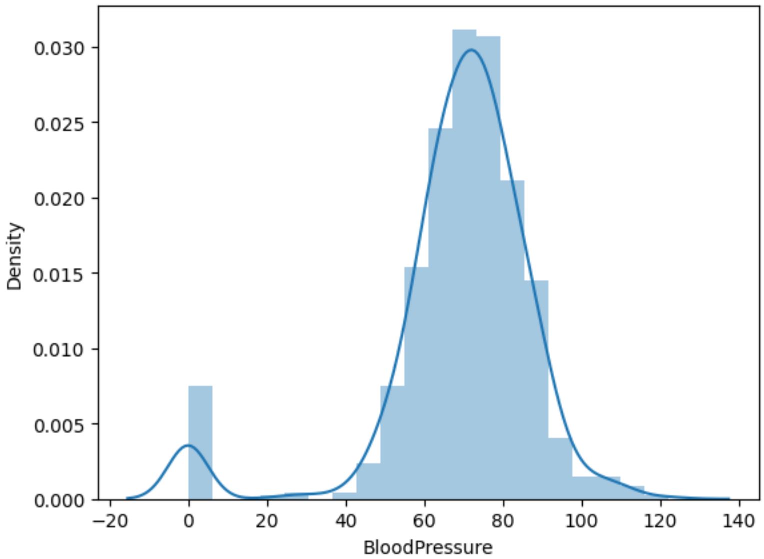

```
median=df['BloodPressure'].median()
print('median:',median)
mean=df['BloodPressure'].mean()
print('mean:',mean)
```

```
median: 72.0
mean: 69.10546875
```

0

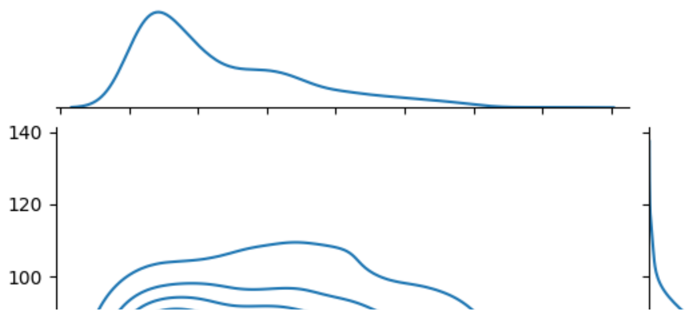
```
sns.distplot(df['BloodPressure'],bins=20)
plt.show(median)
```

```
<ipython-input-19-2148420b5fe4>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['BloodPressure'],bins=20)
```



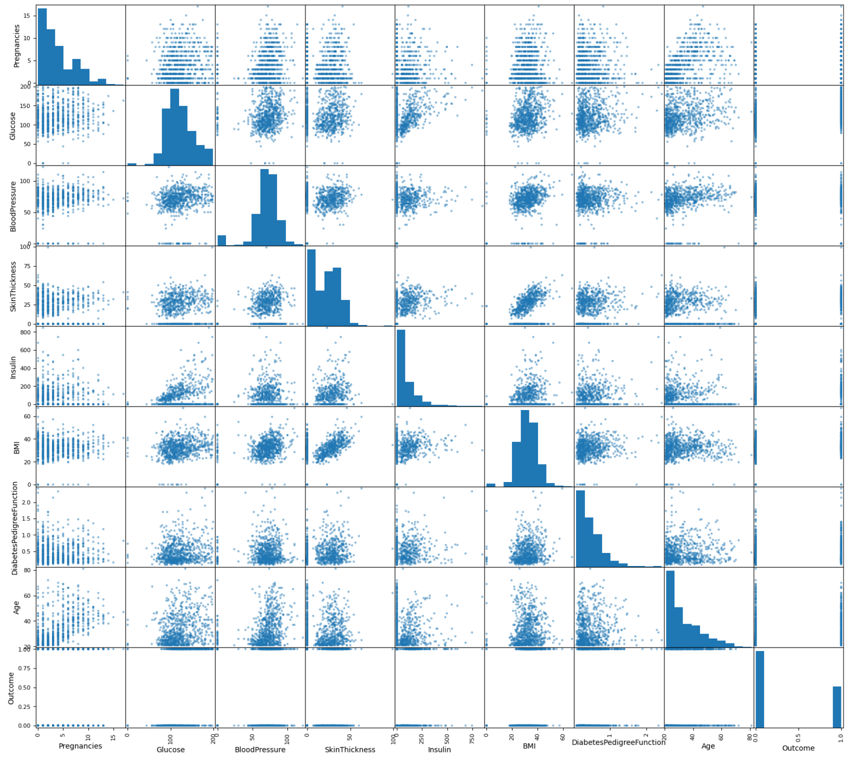

```
sns.jointplot(data=df,x='Age',y='BloodPressure',kind='kde')
```

```
<seaborn.axisgrid.JointGrid at 0x7e0ec4c1c2b0>
```



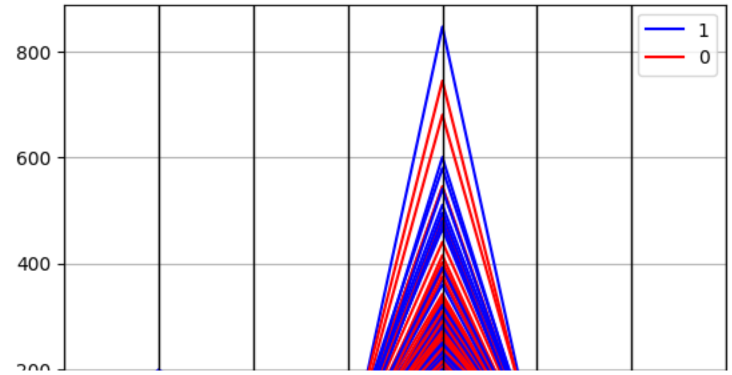

```
pd.plotting.scatter_matrix(df,figsize=(20,18))
```

```
        <Axes: xlabel='Insulin', ylabel='DiabetesPedigreeFunction'>,
        <Axes: xlabel='BMI', ylabel='DiabetesPedigreeFunction'>,
        <Axes: xlabel='DiabetesPedigreeFunction',
ylabel='DiabetesPedigreeFunction'>,
        <Axes: xlabel='Age', ylabel='DiabetesPedigreeFunction'>,
        <Axes: xlabel='Outcome', ylabel='DiabetesPedigreeFunction'>],
       [<Axes: xlabel='Pregnancies', ylabel='Age'>,
        <Axes: xlabel='Glucose', ylabel='Age'>,
        <Axes: xlabel='BloodPressure', ylabel='Age'>,
        <Axes: xlabel='SkinThickness', ylabel='Age'>,
        <Axes: xlabel='Insulin', ylabel='Age'>,
        <Axes: xlabel='BMI', ylabel='Age'>,
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Age'>,
        <Axes: xlabel='Age', ylabel='Age'>,
        <Axes: xlabel='Outcome', ylabel='Age'>],
       [<Axes: xlabel='Pregnancies', ylabel='Outcome'>,
        <Axes: xlabel='Glucose', ylabel='Outcome'>,
        <Axes: xlabel='BloodPressure', ylabel='Outcome'>,
        <Axes: xlabel='SkinThickness', ylabel='Outcome'>,
        <Axes: xlabel='Insulin', ylabel='Outcome'>,
        <Axes: xlabel='BMI', ylabel='Outcome'>,
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Outcome'>,
        <Axes: xlabel='Age', ylabel='Outcome'>,
        <Axes: xlabel='Outcome', ylabel='Outcome'>]], dtype=object)
```




```
pd.plotting.parallel_coordinates(df,'Outcome',color='br')
plt.xticks(rotation=45)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, 'Pregnancies'),
  Text(1, 0, 'Glucose'),
  Text(2, 0, 'BloodPressure'),
  Text(3, 0, 'SkinThickness'),
  Text(4, 0, 'Insulin'),
  Text(5, 0, 'BMI'),
  Text(6, 0, 'DiabetesPedigreeFunction'),
  Text(7, 0, 'Age')])
```



Training the model with the help of train test split.

importing new libraries.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Train-test-split

```
x=df.drop('Outcome',axis=1)
y=df['Outcome']
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size=0.25)
x_test,x_train,y_test,y_train
```

```
(     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
 626            0      125             68              0        0  24.7
 361            5      158             70              0        0  29.8
 231            6      134             80             37      370  46.2
 379            0       93            100             39       72  43.4
 210            2       81             60             22        0  27.7
 ..           ...      ...            ...            ...      ...   ...
 25            10      125             70             26      115  31.1
 667           10      111             70             27        0  27.5
 333           12      106             80              0        0  23.6
 278            5      114             74              0        0  24.9
 124            0      113             76              0        0  33.3

      DiabetesPedigreeFunction  Age
 626                     0.206   21
 361                     0.207   63
 231                     0.238   46
 379                     1.021   35
 210                     0.290   25
 ..                        ...  ...
 25                      0.205   41
 667                     0.141   40
 333                     0.137   44
 278                     0.744   57
 124                     0.278   23

 [576 rows x 8 columns],
      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
 364            4      147             74             25      293  34.9
 459            9      134             74             33       60  25.9
 336            0      117              0              0        0  33.8
 635           13      104             72              0        0  31.2
 401            6      137             61              0        0  24.2
 ..           ...      ...            ...            ...      ...   ...
 258            1      193             50             16      375  25.9
 718            1      108             60             46      178  35.5
 269            2      146              0              0        0  27.5
 692            2      121             70             32       95  39.1
 521            3      124             80             33      130  33.2

      DiabetesPedigreeFunction  Age
 364                     0.385   30
 459                     0.460   81
 336                     0.932   44
 635                     0.465   38
```

```
401                     0.151   55
..                        ...  ...
258                     0.655   24
718                     0.415   24
269                     0.240   28
692                     0.886   23
521                     0.305   26

[192 rows x 8 columns],
626    0
361    0
231    1
379    0
```

Training The Model

```
model=LogisticRegression()
model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

Making prediction

```
prediction=model.predict(x_test)
prediction
```

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

After training the model, predictions are made using the test data, which comprises 25% of the total dataset.

```
accuracy=accuracy_score(prediction,y_test)
percentage=f'{accuracy:.0%}'
print('accuracy of train-test-model is', accuracy)
print('percentage of train-test-model is', percentage)
```

```
accuracy of train-test-model is 0.7586805555555556
percentage of train-test-model is 76%
```

Hence Our Train Test Model is Correct upto 76%