Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence and Machine Learning
Department of Artificial Intelligence and Data Science
B.Tech. Sem: IV Subject: Statistics for Engineers Laboratory
Experiment 7

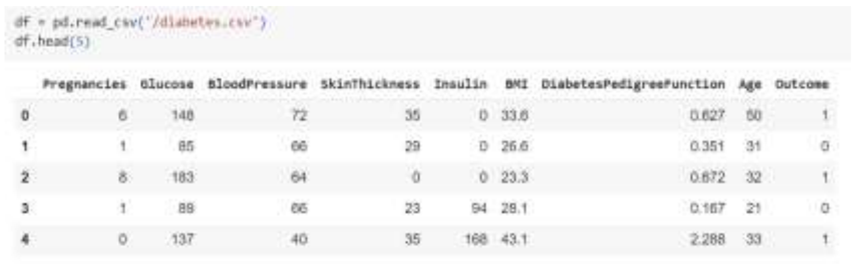
Name: Parth Karia **SAP ID: 60018230108**

Date:	Title: Descriptive Statistics, handling missing values and Correlation Matrix for diabetes dataset.
Aim	To do descriptive Statistics and plot a correlation matrix for a dataset of your choice.
Software	Google Colab, Kaggle/ UCI Machine Learning Repository to download a dataset
Theory	**Data preprocessing on diabetes dataset**

1. **Import numpy, seaborn, matplotlib and pandas**

```
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import pandas as pd
```

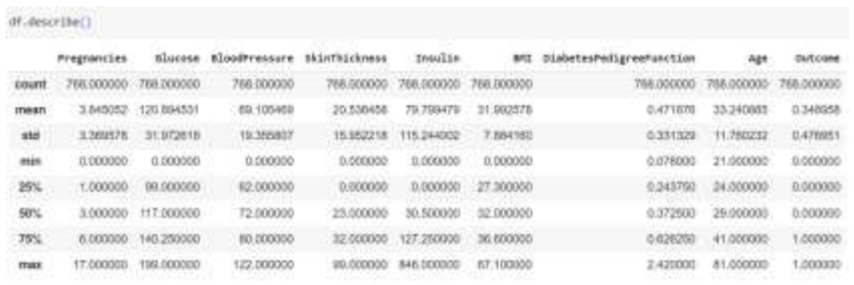
2. **Read dataset and print first 5 rows**



```
df = pd.read_csv('/diabetes.csv')
df.head(5)
```

	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

3. **describe()**

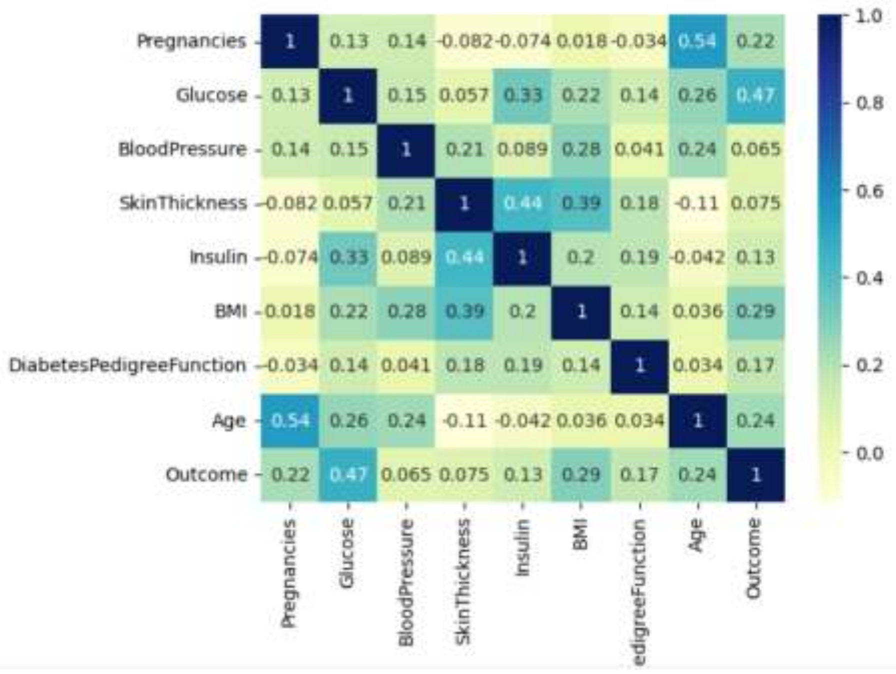


```
df.describe()
```

	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	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	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

4. Plot correlation matrix of the diabetes dataset.

```
dataplot = sn.heatmap(df.corr(), cmap="YlGnBu", annot=True)

plt.show()
```



5. Sum of null values in every column

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

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

6. Drop columns Pregnancies and Outcome.

```
df_new = df.drop(['Pregnancies','Outcome'], axis=1)
```

7. Replacing 0 with nan value in all columns in the dataset

```
df_new.replace(0, np.nan, inplace=True)
print(df_new)
```

```
     Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0      148.0           72.0           35.0      NaN  33.6
1       85.0           66.0           29.0      NaN  26.6
2      183.0           64.0            NaN      NaN  23.3
3       89.0           66.0           23.0     94.0  28.1
4      137.0           40.0           35.0    168.0  43.1
..       ...            ...            ...      ...   ...
763    101.0           76.0           48.0    180.0  32.9
764    122.0           70.0           27.0      NaN  36.8
765    121.0           72.0           23.0    112.0  26.2
766    126.0           60.0            NaN      NaN  30.1
767     93.0           70.0           31.0      NaN  30.4
```

8. Sort the columns, with descending order of nan values

```
null_count_new = df_new.isnull().sum()
missing_data = pd.DataFrame({'Missing Values': null_count_new})
missing_data.sort_values(by='Missing Values', ascending=False)
```

	Missing Values
Insulin	374
SkinThickness	227
BloodPressure	35
BMI	11
Glucose	5
DiabetesPedigreeFunction	0
Age	0

9. describe()

```
df_new.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	763.000000	733.000000	541.000000	394.000000	757.000000	768.000000	768.000000
mean	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885
std	30.535641	12.382158	10.476982	118.775855	6.924988	0.331329	11.760232
min	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000
25%	99.000000	64.000000	22.000000	76.250000	27.500000	0.243750	24.000000
50%	117.000000	72.000000	29.000000	125.000000	32.300000	0.372500	29.000000
75%	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250	41.000000
max	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

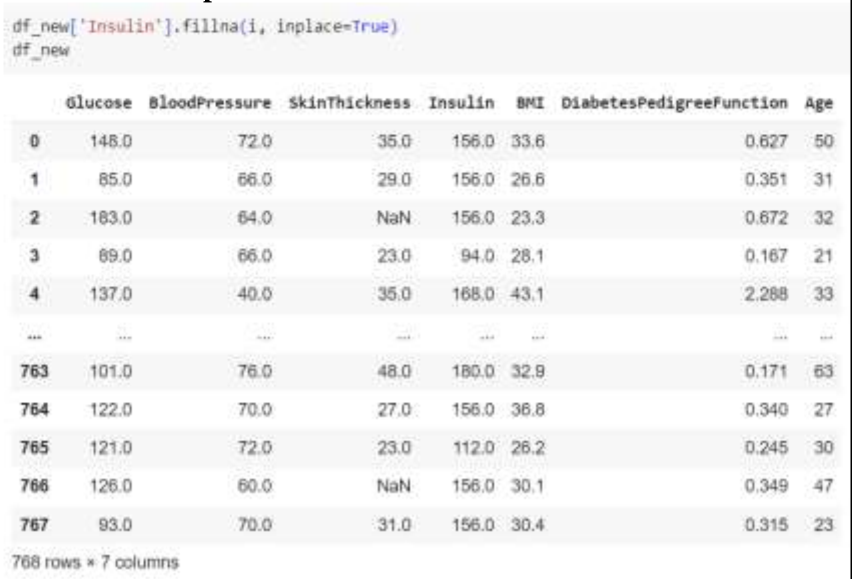
Handling missing values:
Using mean of the column

1. **Calculate the mean of the 'Insulin' column using NumPy's mean() function and round the result using round(). The rounded mean is stored in the variable i.**

```
insulin_mean = np.mean(df_new['Insulin'])
i = np.round_(insulin_mean)
print(i)
```

```
156.0
```

2. **Replace all NaN values in the 'Insulin' column with the rounded mean i. The inplace=True parameter modifies the DataFrame in place.**

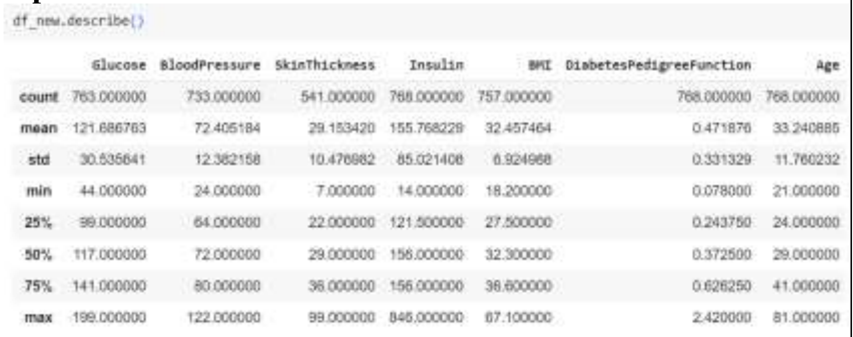


```
df_new['Insulin'].fillna(i, inplace=True)
df_new
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148.0	72.0	35.0	156.0	33.6	0.627	50
1	85.0	66.0	29.0	156.0	26.6	0.351	31
2	183.0	64.0	NaN	156.0	23.3	0.672	32
3	89.0	66.0	23.0	94.0	28.1	0.167	21
4	137.0	40.0	35.0	168.0	43.1	2.288	33
...	...	...	...	...	...	...	...
763	101.0	76.0	48.0	180.0	32.9	0.171	63
764	122.0	70.0	27.0	156.0	36.8	0.340	27
765	121.0	72.0	23.0	112.0	26.2	0.245	30
766	126.0	60.0	NaN	156.0	30.1	0.349	47
767	93.0	70.0	31.0	156.0	30.4	0.315	23

768 rows × 7 columns

3. **Generate descriptive statistics of the DataFrame after the replacement.**



```
df_new.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	763.000000	733.000000	541.000000	768.000000	757.000000	768.000000	768.000000
mean	121.686763	72.405184	29.153420	155.768229	32.457464	0.471876	33.240885
std	30.535641	12.382158	10.476982	85.021408	6.924988	0.331329	11.760232
min	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000
25%	99.000000	64.000000	22.000000	121.500000	27.500000	0.243750	24.000000
50%	117.000000	72.000000	29.000000	156.000000	32.300000	0.372500	29.000000
75%	141.000000	80.000000	36.000000	156.000000	36.600000	0.626250	41.000000
max	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Using SimpleImputer

1. **Import the SimpleImputer class from the scikit-learn library, which provides tools for handling missing data.**

```
from sklearn.impute import SimpleImputer
```

2. **Create an instance of SimpleImputer with the strategy set to 'median'. This means that missing values in the dataset will be replaced with the median value of each column.**

```
imp_median = SimpleImputer(missing_values=np.nan, strategy='median')
imp_median
```

```
▾        SimpleImputer
SimpleImputer(strategy='median')
```

3. **Fit the imputer to your dataset (data) and simultaneously transform it by replacing missing values with the median. The result is stored in data_array.**
4. **data_array: This variable now contains the transformed dataset with missing values imputed using the median strategy.**

```
imp_median.fit(df_new)
data_array = imp_median.transform(df_new)
data_array
```

```
array([[148.   ,  72.   ,  35.   , ...,  33.6  ,   0.627,  50.   ],
       [ 85.   ,  66.   ,  29.   , ...,  26.6  ,   0.351,  31.   ],
       [183.   ,  64.   ,  29.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [121.   ,  72.   ,  23.   , ...,  26.2  ,   0.245,  30.   ],
       [126.   ,  60.   ,  29.   , ...,  30.1  ,   0.349,  47.   ],
       [ 93.   ,  70.   ,  31.   , ...,  30.4  ,   0.315,  23.   ]])
```

5. **Print column names.**

```
print(diabetes_new.columns)
```

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

Form a new dataframe "diabetes_new" from the above array.

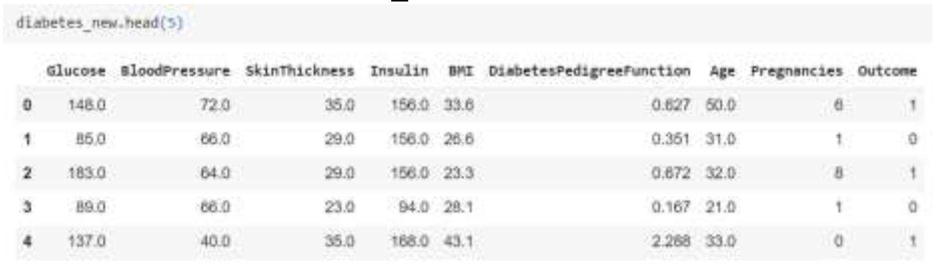
```
diabetes_new = pd.DataFrame(data_array, columns=df_new.columns)
diabetes_new
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148.0	72.0	35.0	156.0	33.6	0.627	50.0
1	85.0	66.0	29.0	156.0	26.6	0.351	31.0
2	183.0	64.0	29.0	156.0	23.3	0.672	32.0
3	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	137.0	40.0	35.0	168.0	43.1	2.288	33.0

Add columns "pregnancies" and "outcome" from "diabetes" dataframe to "diabetes_new" dataframe.

```
diabetes_new['Pregnancies'] = df['Pregnancies']
diabetes_new['Outcome'] = df['Outcome']
```

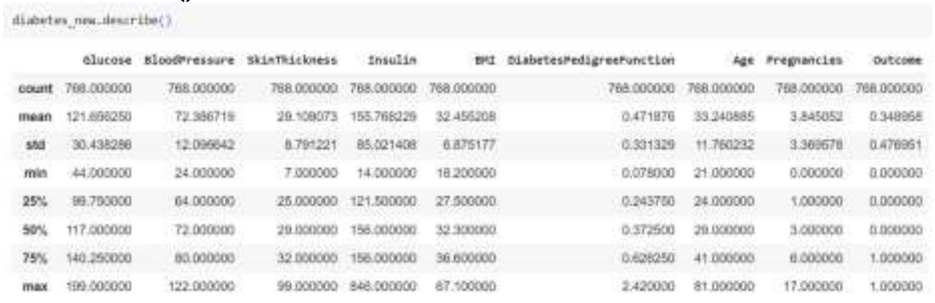
Print first 5 rows of diabetes_new.



```
diabetes_new.head(5)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Outcome
0	148.0	72.0	35.0	156.0	33.6	0.627	50.0	6	1
1	85.0	66.0	29.0	156.0	26.6	0.351	31.0	1	0
2	183.0	64.0	29.0	156.0	23.3	0.672	32.0	8	1
3	89.0	66.0	23.0	94.0	28.1	0.167	21.0	1	0
4	137.0	40.0	35.0	168.0	43.1	2.288	33.0	0	1

Use describe() for diabetes_new.



```
diabetes_new.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	121.656250	72.386719	29.108073	155.768229	32.456208	0.471876	33.240885	3.845052	0.348958
std	30.438286	12.096642	8.791221	85.021408	6.875177	0.331329	11.760232	3.369578	0.476951
min	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000	0.000000
25%	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	1.000000	0.000000
50%	117.000000	72.000000	29.000000	156.000000	32.300000	0.372500	29.000000	3.000000	0.000000
75%	140.250000	80.000000	32.000000	156.000000	36.600000	0.626250	41.000000	6.000000	1.000000
max	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	17.000000	1.000000

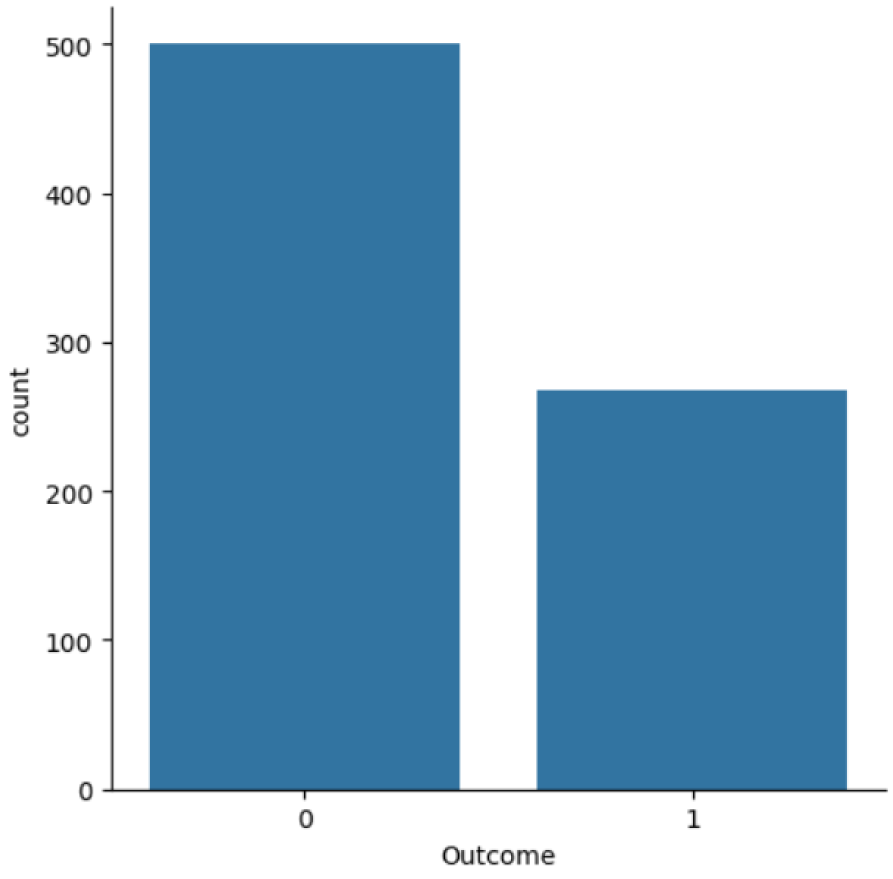
Find sum of null values of all columns in diabetes_new.

```
diabetes_new.isnull().sum()
```

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

Use Seaborn's catplot function to create a count plot for the 'Outcome' variable in the diabetes_new DataFrame.

```
sn.catplot(x="Outcome", kind="count", data=diabetes_new)
plt.show()
```



Count the occurrences of unique values in the 'Outcome' column of the DataFrame diabetes_new.

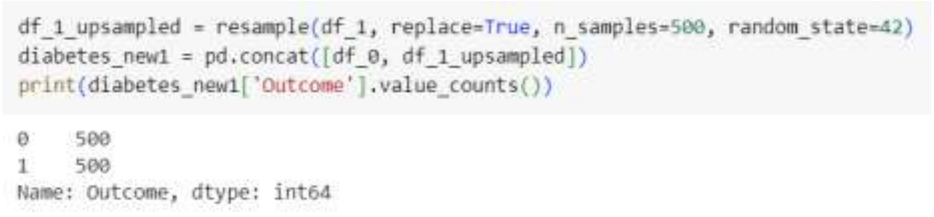
```
outcome_counts = diabetes_new['Outcome'].value_counts()
print(outcome_counts)
```

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

Apply resample using bootstrapping. To make both 0 and 1 outcomes to be 500.

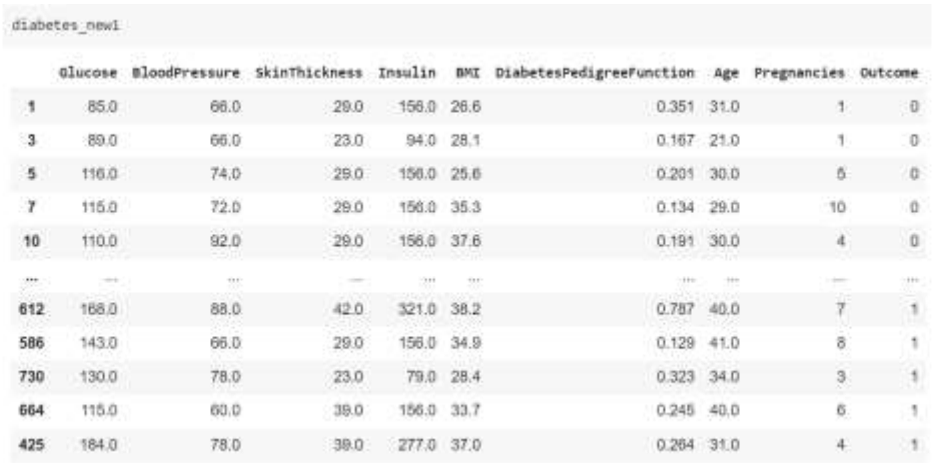
```
from sklearn.utils import resample

df_0 = diabetes_new[diabetes_new['Outcome'] == 0]
df_1 = diabetes_new[diabetes_new['Outcome'] == 1]
```

Concatanate “0” outcome with “1” outcome. And name that dataframe as diabetes_new1.



```
df_1_upsampled = resample(df_1, replace=True, n_samples=500, random_state=42)
diabetes_new1 = pd.concat([df_0, df_1_upsampled])
print(diabetes_new1['Outcome'].value_counts())
```

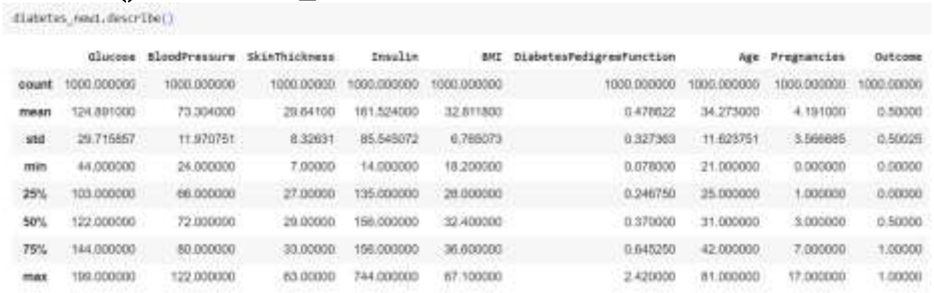
```
0    500
1    500
Name: Outcome, dtype: int64
```



```
diabetes_new1
```

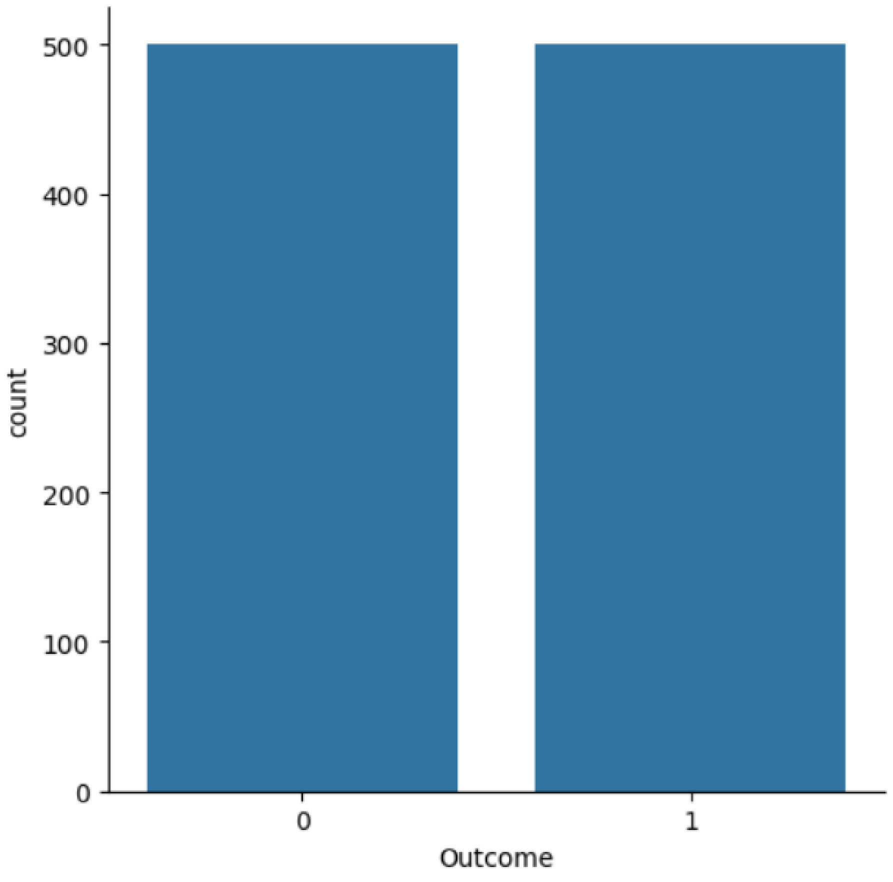
	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Outcome
1	85.0	66.0	29.0	156.0	26.6	0.351	31.0	1	0
3	89.0	66.0	23.0	94.0	28.1	0.167	21.0	1	0
5	116.0	74.0	29.0	156.0	25.6	0.201	30.0	5	0
7	115.0	72.0	29.0	156.0	35.3	0.134	29.0	10	0
10	110.0	92.0	29.0	156.0	37.6	0.191	30.0	4	0
...	...	...	...	...	...	...	...	...	...
612	168.0	88.0	42.0	321.0	38.2	0.787	40.0	7	1
586	143.0	66.0	29.0	156.0	34.9	0.129	41.0	8	1
730	130.0	78.0	23.0	79.0	28.4	0.323	34.0	3	1
664	115.0	60.0	39.0	156.0	33.7	0.245	40.0	6	1
425	184.0	78.0	39.0	277.0	37.0	0.264	31.0	4	1

describe() on diabetes_new.1.



```
diabetes_new1.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Outcome
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	124.891000	73.304000	28.64100	161.524000	32.811800	0.478622	34.273000	4.191000	0.50000
std	29.715857	11.970751	8.32631	85.545072	6.768073	0.327363	11.623751	3.566685	0.50025
min	44.000000	24.000000	7.00000	14.000000	18.200000	0.078000	21.000000	0.000000	0.00000
25%	103.000000	66.000000	27.00000	135.000000	28.000000	0.246750	25.000000	1.000000	0.00000
50%	122.000000	72.000000	29.00000	156.000000	32.400000	0.370000	31.000000	3.000000	0.50000
75%	144.000000	80.000000	30.00000	156.000000	36.600000	0.645250	42.000000	7.000000	1.00000
max	199.000000	122.000000	63.00000	744.000000	67.100000	2.420000	81.000000	17.000000	1.00000

	Use Seaborn's catplot function again to create a count plot for the 'Outcome' variable in the diabetes_new1 DataFrame. Now you will see both 0 and 1 outcome count as 500.
	 `sn.catplot(x="Outcome", kind="count", data=diabetes_new1)`
	 `plt.show()`
	 

Conclusion	Thus, we implemented various data preprocessing steps on diabetes dataset.

Signature of Faculty