```
from google.colab import files
uploaded = files.upload()
```

Choose Files Dataset of Diabetes .csv

- **Dataset of Diabetes .csv**(text/csv) - 49511 bytes, last modified: 3/3/2025 - 100% done

```
Saving Dataset of Diabetes .csv to Dataset of Diabetes .csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
```

```
import pandas as pd

# Replace 'your_file.csv' with the name of the file you just uploaded
df = pd.read_csv('Dataset of Diabetes .csv')
df.head()  # Display the first few rows
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.head(10)
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
5	634	34224	F	45	2.3	24	4.0	2.9	1.0	1.0	1.5	0.4	21.0	N
6	721	34225	F	50	2.0	50	4.0	3.6	1.3	0.9	2.1	0.6	24.0	N
7	421	34227	M	48	4.7	47	4.0	2.9	0.8	0.9	1.6	0.4	24.0	N
8	670	34229	M	43	2.6	67	4.0	3.8	0.9	2.4	3.7	1.0	21.0	N
9	759	34230	F	32	3.6	28	4.0	3.8	2.0	2.4	3.8	1.0	24.0	N

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(1000, 14)
```

```
#df.loc[5, 'Age'] = np.nan
#df.loc[10, 'Salary'] = np.nan
df.head(10)
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
5	634	34224	F	45	2.3	24	4.0	2.9	1.0	1.0	1.5	0.4	21.0	N
6	721	34225	F	50	2.0	50	4.0	3.6	1.3	0.9	2.1	0.6	24.0	N
7	421	34227	M	48	4.7	47	4.0	2.9	0.8	0.9	1.6	0.4	24.0	N
8	670	34229	M	43	2.6	67	4.0	3.8	0.9	2.4	3.7	1.0	21.0	N
9	759	34230	F	32	3.6	28	4.0	3.8	2.0	2.4	3.8	1.0	24.0	N

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
print(df.describe())
```

```
                ID     No_Pation          AGE         Urea           Cr  \
count  1000.000000  1.000000e+03  1000.000000  1000.000000  1000.000000   
mean    340.500000  2.705514e+05    53.528000     5.124743    68.943000   
std     240.397673  3.380758e+06     8.799241     2.935165    59.984747   
min       1.000000  1.230000e+02    20.000000     0.500000     6.000000   
25%     125.750000  2.406375e+04    51.000000     3.700000    48.000000   
50%     300.500000  3.439550e+04    55.000000     4.600000    60.000000   
75%     550.250000  4.538425e+04    59.000000     5.700000    73.000000   
max     800.000000  7.543566e+07    79.000000    38.900000   800.000000   

             HbA1c         Chol           TG          HDL          LDL  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000   
mean      8.281160     4.862820     2.349610     1.204750     2.609790   
std       2.534003     1.301738     1.401176     0.660414     1.115102   
min       0.900000     0.000000     0.300000     0.200000     0.300000   
25%       6.500000     4.000000     1.500000     0.900000     1.800000   
50%       8.000000     4.800000     2.000000     1.100000     2.500000   
75%      10.200000     5.600000     2.900000     1.300000     3.300000   
max      16.000000    10.300000    13.800000     9.900000     9.900000   

              VLDL          BMI  
count  1000.000000  1000.000000  
mean      1.854700    29.578020  
std       3.663599     4.962388  
min       0.100000    19.000000  
25%       0.700000    26.000000  
50%       0.900000    30.000000  
75%       1.500000    33.000000  
max      35.000000    47.750000  
```

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Series([], dtype: int64)
```

```
import numpy as np

# Introduce missing values at specific locations
df.loc[5, 'AGE'] = np.nan  # Set missing value for 'AGE' at row index 5
df.loc[10, 'BMI'] = np.nan  # Set missing value for 'BMI' at row index 10

# Display the first 10 rows to check the changes
print(df.head(10))
df
```

```
    ID  No_Pation Gender   AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M  26.0   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M  33.0   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4
5  634      34224      F   NaN   2.3  24    4.0   2.9  1.0  1.0  1.5   0.4
6  721      34225      F  50.0   2.0  50    4.0   3.6  1.3  0.9  2.1   0.6
7  421      34227      M  48.0   4.7  47    4.0   2.9  0.8  0.9  1.6   0.4
8  670      34229      M  43.0   2.6  67    4.0   3.8  0.9  2.4  3.7   1.0
9  759      34230      F  32.0   3.6  28    4.0   3.8  2.0  2.4  3.8   1.0

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N
5  21.0     N
6  24.0     N
7  24.0     N
8  21.0     N
9  24.0     N
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	200	454317	M	71.0	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	Y
996	671	876534	M	31.0	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	Y
997	669	87654	M	30.0	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	Y
998	99	24004	M	38.0	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	Y
999	248	24054	M	54.0	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	Y

1000 rows × 14 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
print(df.describe())
```

```
                ID     No_Pation         AGE         Urea           Cr  \
count  1000.000000  1.000000e+03  999.000000  1000.000000  1000.000000
mean    340.500000  2.705514e+05   53.536537     5.124743    68.943000
std     240.397673  3.380758e+06    8.799504     2.935165    59.984747
min       1.000000  1.230000e+02   20.000000     0.500000     6.000000
25%     125.750000  2.406375e+04   51.000000     3.700000    48.000000
50%     300.500000  3.439550e+04   55.000000     4.600000    60.000000
75%     550.250000  4.538425e+04   59.000000     5.700000    73.000000
max     800.000000  7.543566e+07   79.000000    38.900000   800.000000

             HbA1c         Chol           TG          HDL          LDL  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean      8.281160     4.862820     2.349610     1.204750     2.609790
std       2.534003     1.301738     1.401176     0.660414     1.115102
min       0.900000     0.000000     0.300000     0.200000     0.300000
25%       6.500000     4.000000     1.500000     0.900000     1.800000
50%       8.000000     4.800000     2.000000     1.100000     2.500000
75%      10.200000     5.600000     2.900000     1.300000     3.300000
max      16.000000    10.300000    13.800000     9.900000     9.900000

              VLDL         BMI
count  1000.000000  999.000000
mean      1.854700   29.584605
std       3.663599    4.960501
min       0.100000   19.000000
25%       0.700000   26.000000
50%       0.900000   30.000000
75%       1.500000   33.000000
max      35.000000   47.750000
```

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
AGE    1
BMI    1
dtype: int64
```

```
#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["AGE"]])
imputer2.fit(df_copy[["BMI"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["AGE"] = imputer1.transform(df[["AGE"]])
df_copy["BMI"] = imputer2.transform(df[["BMI"]])

# Verify that there are no missing values left
print(df_copy["AGE"].isnull().sum())
print(df_copy["BMI"].isnull().sum())
```

```
0
0
```

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

# Normalize the Gender column to be consistent (uppercase in this case)
df['Gender'] = df['Gender'].str.upper()  # Convert to uppercase
df['CLASS'] = df['CLASS'].str.upper()
# Initialize OrdinalEncoder for 'Gender' column
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])  # Encoding 'F' as 0, 'M' as 1

# Fit and transform the data in the 'Gender' column
df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])

# Initialize OneHotEncoder for the 'City' column (if the column exists)
# You should replace "City" with the actual column name in your dataset
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column (replace 'City' with the actual name of the column)
if 'CLASS' in df.columns:
    encoded_data = onehot_encoder.fit_transform(df[["CLASS"]])

    # Convert the sparse matrix to a dense array
    encoded_array = encoded_data.toarray()

    # Convert to DataFrame for better visualization
    encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))

    # Concatenate the one-hot encoded columns with the original DataFrame
    df_encoded = pd.concat([df, encoded_df], axis=1)

    # Drop the original 'City' column as it is now encoded
    df_encoded.drop("CLASS", axis=1, inplace=True)

# If there is no 'City' column, proceed with just encoding 'Gender'
else:
    df_encoded = df.copy()

# Drop the original 'Gender' column
df_encoded.drop("Gender", axis=1, inplace=True)

# Display the first few rows of the encoded dataframe
print(df_encoded.head())
df_encoded
```

```
    ID  No_Pation   AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221  26.0   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223  33.0   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y
0             0.0      1.0       0.0      0.0      0.0       0.0
1             1.0      1.0       0.0      0.0      0.0       0.0
2             0.0      1.0       0.0      0.0      0.0       0.0
3             0.0      1.0       0.0      0.0      0.0       0.0
4             1.0      1.0       0.0      0.0      0.0       0.0
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLASS_Y
0	502	17975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
1	735	34221	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	1.0	0.0	0.0	0.0	0.0
2	420	47975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
3	680	87656	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
4	504	34223	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	200	454317	71.0	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	1.0	0.0	0.0	0.0	1.0	0.0
996	671	876534	31.0	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	1.0	0.0	0.0	0.0	0.0	1.0
997	669	87654	30.0	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	1.0	0.0	0.0	0.0	0.0	1.0
998	99	24004	38.0	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	1.0	0.0	0.0	0.0	0.0	1.0
999	248	24054	54.0	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	1.0	0.0	0.0	0.0	0.0	1.0

1000 rows × 18 columns

Next steps: Generate code with df_encoded View recommended plots New interactive sheet

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
normalizer = MinMaxScaler()

# Apply the MinMaxScaler to the 'AGE' column only
df_encoded[['AGE']] = normalizer.fit_transform(df_encoded[['AGE']])

# Display the first few rows to verify the transformation
print(df_encoded.head())
df_encoded
```

```
    ID  No_Pation       AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975  0.508475   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0   
1  735      34221  0.101695   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0   
2  420      47975  0.508475   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0   
3  680      87656  0.508475   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0   
4  504      34223  0.220339   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0   

   Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y   
0             0.0      1.0      0.0      0.0      0.0       0.0  
1             1.0      1.0      0.0      0.0      0.0       0.0  
2             0.0      1.0      0.0      0.0      0.0       0.0  
3             0.0      1.0      0.0      0.0      0.0       0.0  
4             1.0      1.0      0.0      0.0      0.0       0.0  
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLASS_
0	502	17975	0.508475	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.
1	735	34221	0.101695	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	1.0	0.0	0.0	0.0	0.
2	420	47975	0.508475	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.
3	680	87656	0.508475	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.
4	504	34223	0.220339	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	1.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
995	200	454317	0.864407	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	1.0	0.0	0.0	0.0	1.0	0.
996	671	876534	0.186441	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	1.0	0.0	0.0	0.0	0.0	1.
997	669	87654	0.169492	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	1.0	0.0	0.0	0.0	0.0	1.
998	99	24004	0.305085	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	1.0	0.0	0.0	0.0	0.0	1.
999	248	24054	0.576271	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	1.0	0.0	0.0	0.0	0.0	1.

1000 rows × 18 columns

Next steps: Generate code with df_encoded View recommended plots New interactive sheet

```
#Data Transformation
# Min-Max Scaler/Normalization (range 0-1)
#Pros: Keeps all data between 0 and 1; ideal for distance-based models.
#Cons: Can distort data distribution, especially with extreme outliers.
normalizer = MinMaxScaler()
df_encoded[['TG']] = normalizer.fit_transform(df_encoded[['TG']])
df_encoded.head()
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLASS_Y
0	502	17975	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
1	735	34221	26.0	4.5	62	4.9	3.7	0.081481	1.1	2.1	0.6	23.0	1.0	1.0	0.0	0.0	0.0	0.0
2	420	47975	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
3	680	87656	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	0.0
4	504	34223	33.0	7.1	46	4.9	4.9	0.051852	0.8	2.0	0.4	21.0	1.0	1.0	0.0	0.0	0.0	0.0

Next steps: Generate code with df_encoded View recommended plots New interactive sheet

```
# Standardization (mean=0, variance=1)
#Pros: Works well for normally distributed data; suitable for many models.
#Cons: Sensitive to outliers.
scaler = StandardScaler()
df_encoded[['AGE']] = scaler.fit_transform(df_encoded[['AGE']])
df_encoded.head()
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLA
0	502	17975	-0.402465	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	
1	735	34221	-3.132585	4.5	62	4.9	3.7	0.081481	1.1	2.1	0.6	23.0	1.0	1.0	0.0	0.0	0.0	
2	420	47975	-0.402465	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	
3	680	87656	-0.402465	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	0.0	1.0	0.0	0.0	0.0	
4	504	34223	-2.336300	7.1	46	4.9	4.9	0.051852	0.8	2.0	0.4	21.0	1.0	1.0	0.0	0.0	0.0	

Next steps: Generate code with df_encoded | View recommended plots | New interactive sheet

```
#Removing Outliers
# Outlier Detection and Treatment using IQR
#Pros: Simple and effective for mild outliers.
#Cons: May overly reduce variation if there are many extreme outliers.
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['Chol'].quantile(0.25)
Q3 = df_encoded_copy1['Chol'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['Chol'] = np.where(df_encoded_copy1['Chol'] > upper_bound, upper_bound,
                        np.where(df_encoded_copy1['Chol'] < lower_bound, lower_bound, df_encoded_copy1['Chol']))

print(df_encoded_copy1.head())
```

```
    ID  No_Pation       AGE  Urea  Cr  HbA1c  Chol        TG  HDL  LDL  VLDL  \
0  502      17975 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
1  735      34221 -3.132585   4.5  62    4.9   3.7  0.081481  1.1  2.1   0.6
2  420      47975 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
3  680      87656 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
4  504      34223 -2.336300   7.1  46    4.9   4.9  0.051852  0.8  2.0   0.4

    BMI  Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y
0  24.0             0.0      1.0       0.0      0.0      0.0       0.0
1  23.0             1.0      1.0       0.0      0.0      0.0       0.0
2  24.0             0.0      1.0       0.0      0.0      0.0       0.0
3  24.0             0.0      1.0       0.0      0.0      0.0       0.0
4  21.0             1.0      1.0       0.0      0.0      0.0       0.0
```

```
#Removing Outliers
# Z-score method
#Pros: Good for normally distributed data.
#Cons: Not suitable for non-normal data; may miss outliers in skewed distributions.

df_encoded_copy2['Chol_zscore'] = stats.zscore(df_encoded_copy2['Chol'])
df_encoded_copy2['Chol'] = np.where(df_encoded_copy2['Chol_zscore'].abs() > 3, np.nan, df_encoded_copy2['Chol'])  # Replace outliers with NaN
print(df_encoded_copy2.head())
```

```
    ID  No_Pation       AGE  Urea  Cr  HbA1c  Chol        TG  HDL  LDL  VLDL  \
0  502      17975 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
1  735      34221 -3.132585   4.5  62    4.9   3.7  0.081481  1.1  2.1   0.6
2  420      47975 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
3  680      87656 -0.402465   4.7  46    4.9   4.2  0.044444  2.4  1.4   0.5
4  504      34223 -2.336300   7.1  46    4.9   4.9  0.051852  0.8  2.0   0.4

    BMI  Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y   \
0  24.0             0.0      1.0       0.0      0.0      0.0       0.0
1  23.0             1.0      1.0       0.0      0.0      0.0       0.0
2  24.0             0.0      1.0       0.0      0.0      0.0       0.0
3  24.0             0.0      1.0       0.0      0.0      0.0       0.0
4  21.0             1.0      1.0       0.0      0.0      0.0       0.0

   Chol_zscore
0    -0.532005
1    -0.945425
2    -0.532005
3    -0.532005
4     0.046783
```

```
#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
df_encoded_copy3['Chol_zscore'] = stats.zscore(df_encoded_copy3['Chol'])
median_salary = df_encoded_copy3['Chol'].median()
df_encoded_copy3['Chol'] = np.where(df_encoded_copy3['Chol'].abs() > 3, median_salary, df_encoded_copy3['Chol'])
print(df_encoded_copy3.head())
```

```
    ID  No_Pation       AGE  Urea  Cr  HbA1c  Chol        TG  HDL  LDL  VLDL  \
0  502      17975 -0.402465   4.7  46    4.9   4.8  0.044444  2.4  1.4   0.5
1  735      34221 -3.132585   4.5  62    4.9   4.8  0.081481  1.1  2.1   0.6
```

```
        2  420      47975 -0.402465   4.7  46    4.9   4.8  0.044444  2.4  1.4   0.5
```