**Python Coding Challenge**

**Name: S.Yazhini**

**Date: 15-11-2024**

1. **Printing rows of the Data**

* df = pd.read\_csv(‘ ’) is used to load and use the data in our code
* Loads data from a CSV file into a DataFrame using pd.read\_csv() and displays it using print()

**Code:**

import pandas as pd

# Load the dataset

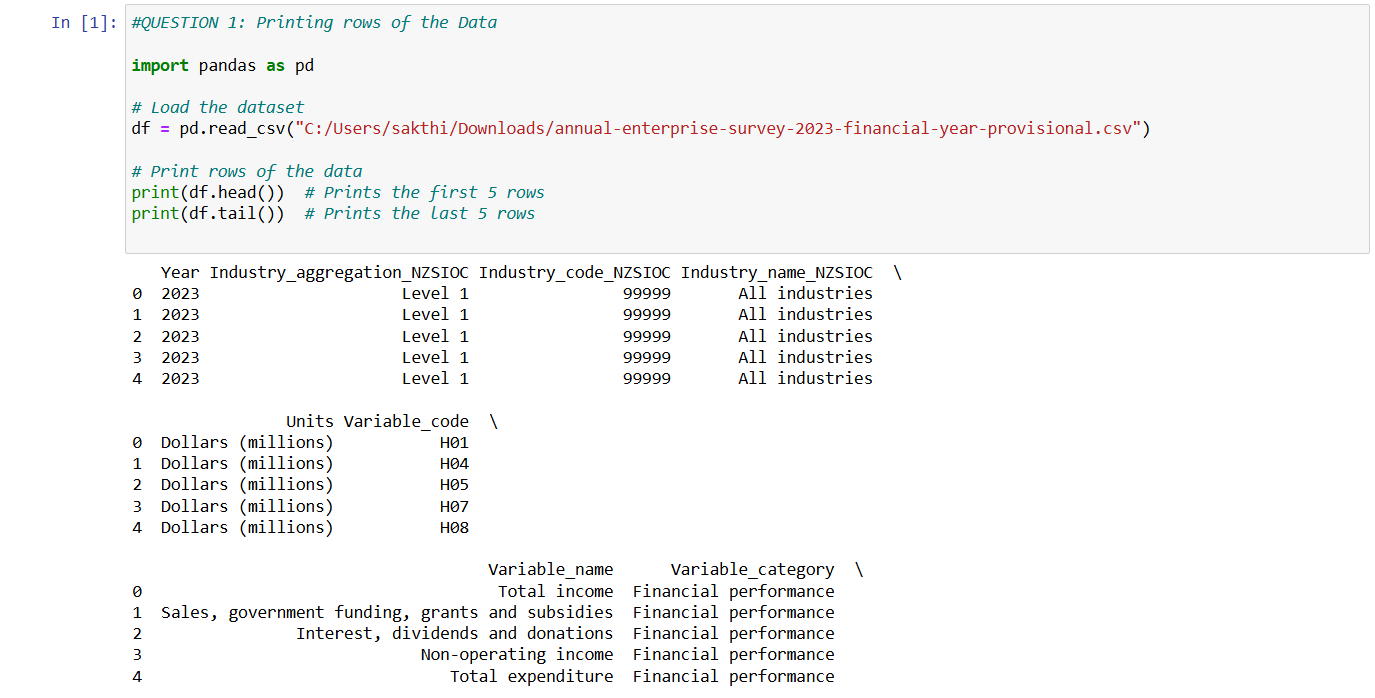
df = pd.read\_csv("C:/Users/sakthi/Downloads/annual-enterprise-survey-2023-financial-year-provisional.csv")

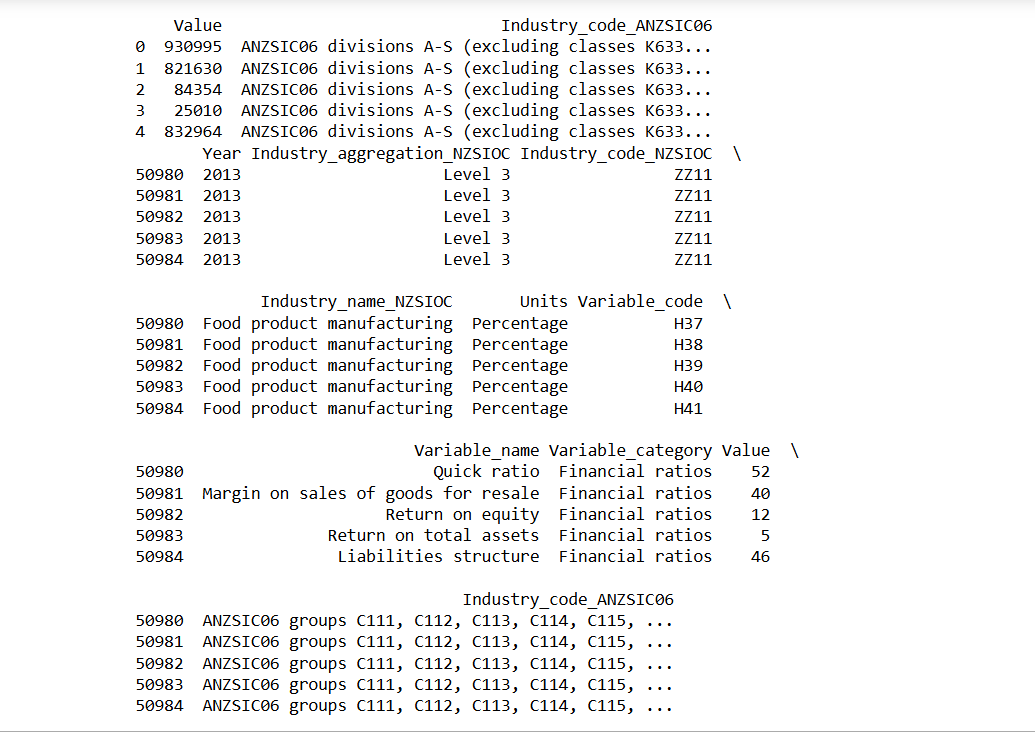
# Print rows of the data

print(df.head()) # Prints the first 5 rows

print(df.tail()) # Prints the last 5 rows

**Output:**

****

****

1. **Printing the column names of the DataFrame**

* df.columns returns a list of column names in the DataFrame.
* df.dtypes shows the data type (e.g., int64, float64, object) for each column. This is useful for understanding the structure of the data and determining if any columns need to be converted to different types for further processing.

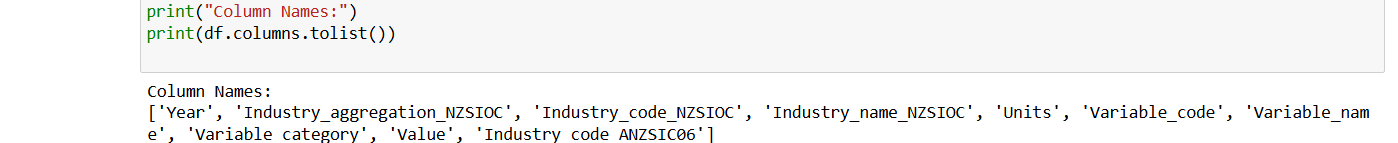
**Code:**

# Print column names

print("Column Names:")

print(df.columns.tolist())

**Output:**

****

1. **Summary of Data Frame**

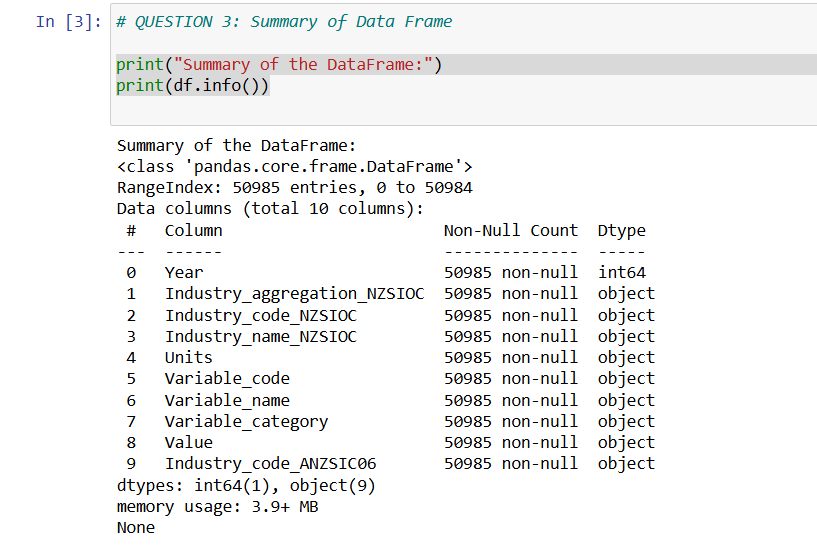
* df.info() Prints all the summary information about the data

**Code:**

print("Summary of the DataFrame:")

print(df.info())

**Output:**

****

1. **Descriptive Statistical Measures of a DataFrame**

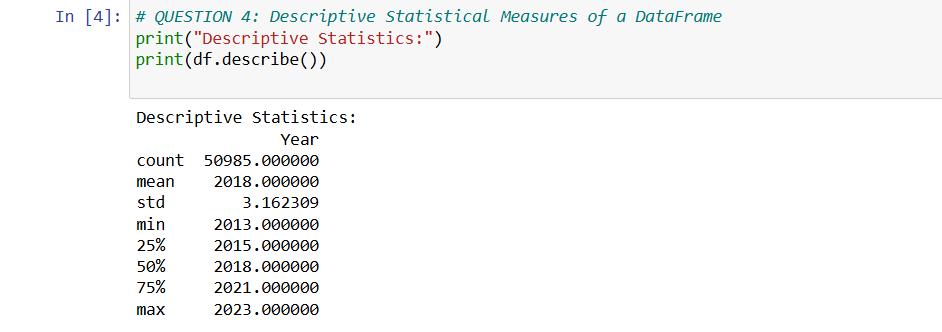
* df.describe() helps us to get the Descriptive Statistical Measures of a DataFrame.

**Code:**

# QUESTION 4: Descriptive Statistical Measures of a DataFrame

print("Descriptive Statistics:")

print(df.describe())

**Output:**

1. **Missing Data Handing**

* df.dropna() removes rows with any missing values, resulting in a new DataFrame df\_dropped without null values.
* df.fillna(df.mean()) replaces missing values in numeric columns with the column's mean, producing df\_filled, which retains all rows but fills nulls with calculated averages.

**Code:**

#QUESTION 5: Missing Data Handing

# Check for missing data

print("Missing Data:")

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

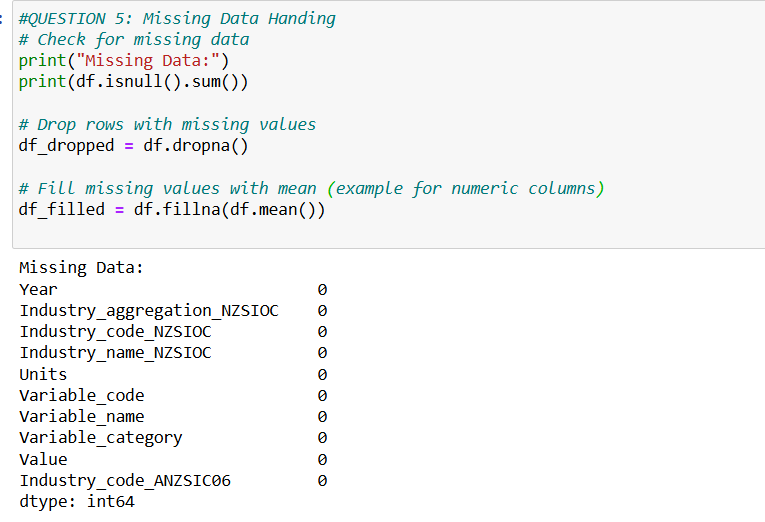
# Drop rows with missing values

df\_dropped = df.dropna()

# Fill missing values with mean (example for numeric columns)

df\_filled = df.fillna(df.mean())

**Output:**

****

1. **Sorting DataFrame values**

* This code sorts the rows of the DataFrame df in ascending order based on the values in the column 'Industry\_aggregation\_NZSIOC'. The sorted DataFrame is stored in sorted\_df and then printed.

**Code:**

#QUESTION 6: Sorting DataFrame values

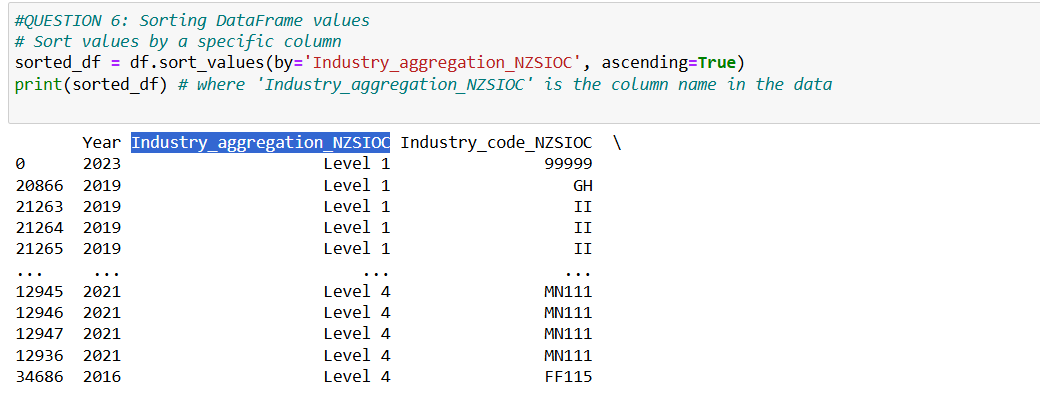
# Sort values by a specific column

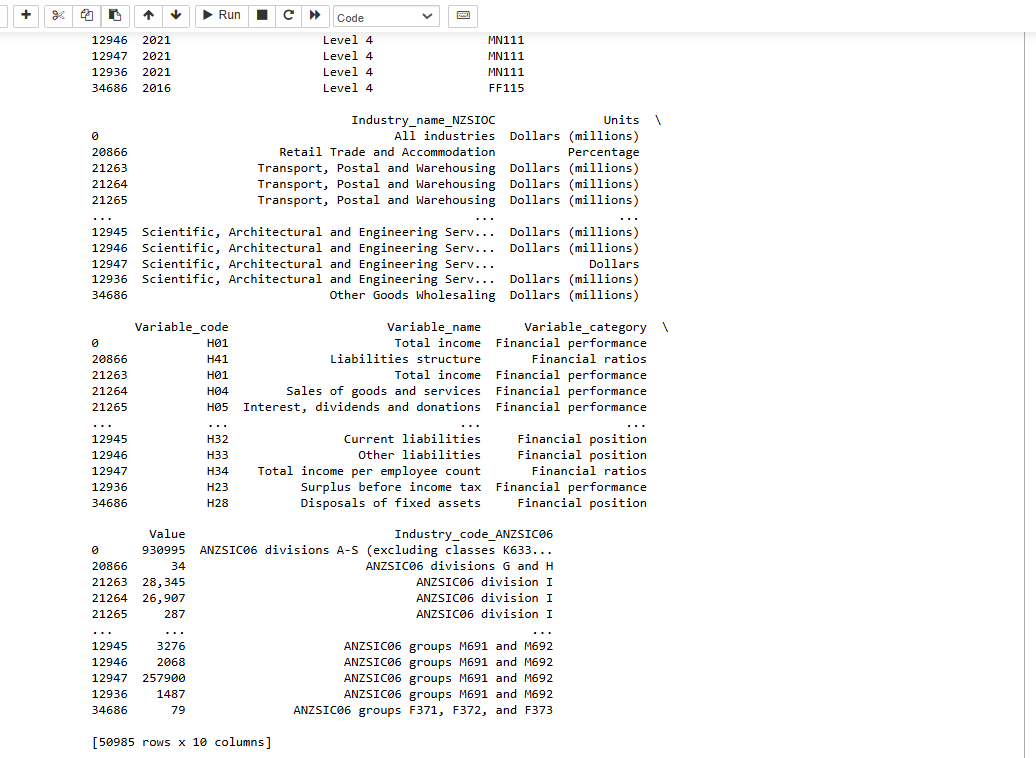
sorted\_df = df.sort\_values(by='Industry\_aggregation\_NZSIOC', ascending=True)

print(sorted\_df)

# where 'Industry\_aggregation\_NZSIOC' is the column name in the data

**Output:**

****

****

1. **Merge Data Frames**

* This code checks the data types of columns in two DataFrames (df and df2).
* It ensures that the 'Industry\_code\_NZSIOC' column in both DataFrames is of type str before merging them on this column using a left join (how='left').
* The result is stored in merged\_df and then printed.

**Code:**

# Check data types of both DataFrames

print(df.dtypes)

print(df2.dtypes)

# Ensuring 'Industry\_code\_NZSIOC' is of the same type in both DataFrames

df['Industry\_code\_NZSIOC'] = df['Industry\_code\_NZSIOC'].astype(str)

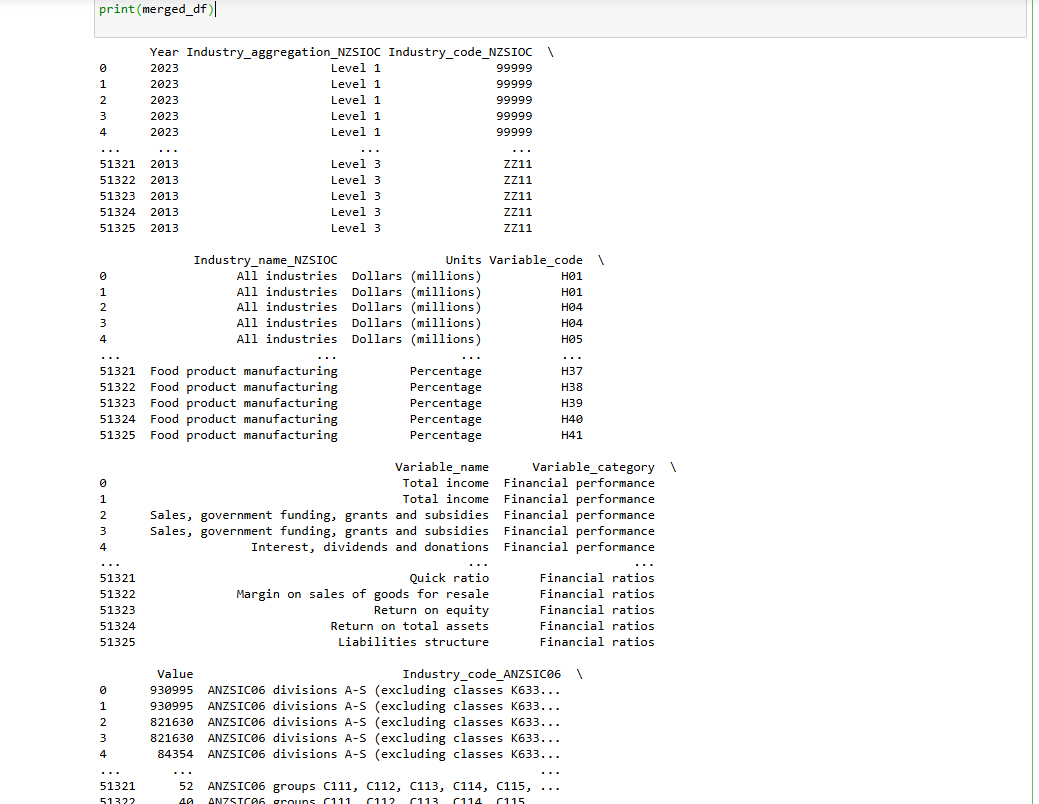
df2['Industry\_code\_NZSIOC'] = df2['Industry\_code\_NZSIOC'].astype(str)

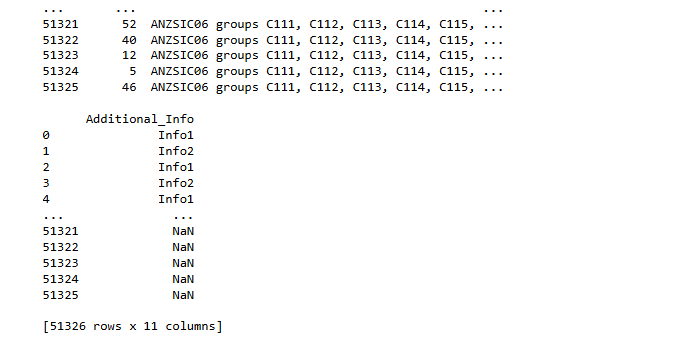
# merging data

merged\_df = pd.merge(df, df2, on='Industry\_code\_NZSIOC', how='left')

print(merged\_df)

**Output:**

****

****

1. **Apply Function**

* This code ensures the `Value` column is numeric by coercing invalid entries to `NaN`.
* It then uses the `apply()` method to transform values in the `Value` column to thousands using the `to\_thousands` function, creating a new column `Value\_in\_thousands`, and prints both columns.

**Code:**

#QUES 8: Apply Function

# Ensuring 'Value' column is numeric

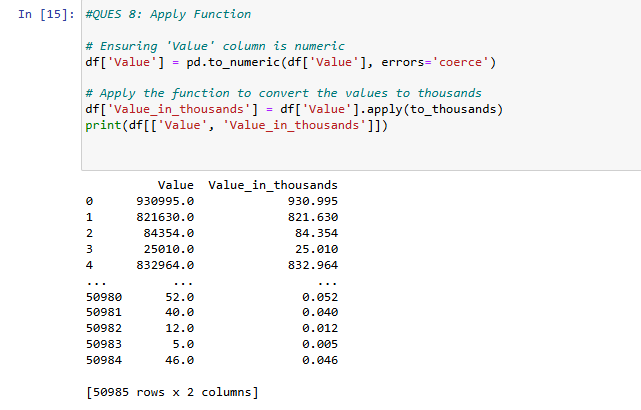
df['Value'] = pd.to\_numeric(df['Value'], errors='coerce')

# Apply the function to convert the values to thousands

df['Value\_in\_thousands'] = df['Value'].apply(to\_thousands)

print(df[['Value', 'Value\_in\_thousands']])

**Output:**

****

1. **By using the lambda operator**

* This code converts the `Value` column to numeric (handling invalid entries as `NaN`) and uses a `lambda` function to add 100 to each value.
* The transformed values are stored in a new column `Value\_plus\_100`, and both columns are printed.

**Code:**

# QUESTION 9: By using the lambda operator

#Ensuring 'Value' column is numeric

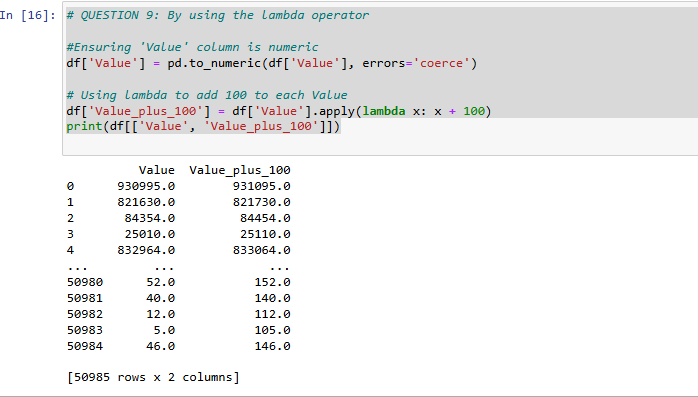
df['Value'] = pd.to\_numeric(df['Value'], errors='coerce')

# Using lambda to add 100 to each Value

df['Value\_plus\_100'] = df['Value'].apply(lambda x: x + 100)

print(df[['Value', 'Value\_plus\_100']])

**Output:**

****

1. **Visualizing DataFrame**

**matplotlib.pyplot:**

* A core library for creating static, interactive, and animated visualizations in Python.
* Used here for setting up the figure (plt.figure) and overall plot aesthetics (e.g., plt.title, plt.xticks).

**seaborn:**

* A data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.
* Used here to create a bar plot (sns.barplot), which visualizes the relationship between 'Variable\_name' and 'Value'.

**pandas:**

* This library for Python, especially useful for working with tabular data.
* Used here to ensure the 'Value' column is numeric (pd.to\_numeric) and to clean the data by dropping rows with NaN values in 'Value' (df.dropna).

**Code:**

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Ensuring 'Value' column is numeric

df['Value'] = pd.to\_numeric(df['Value'], errors='coerce')

# Drop any rows where 'Value' is NaN after conversion

df = df.dropna(subset=['Value'])

# Bar Plot of Variable\_name vs Value

plt.figure(figsize=(10, 6))

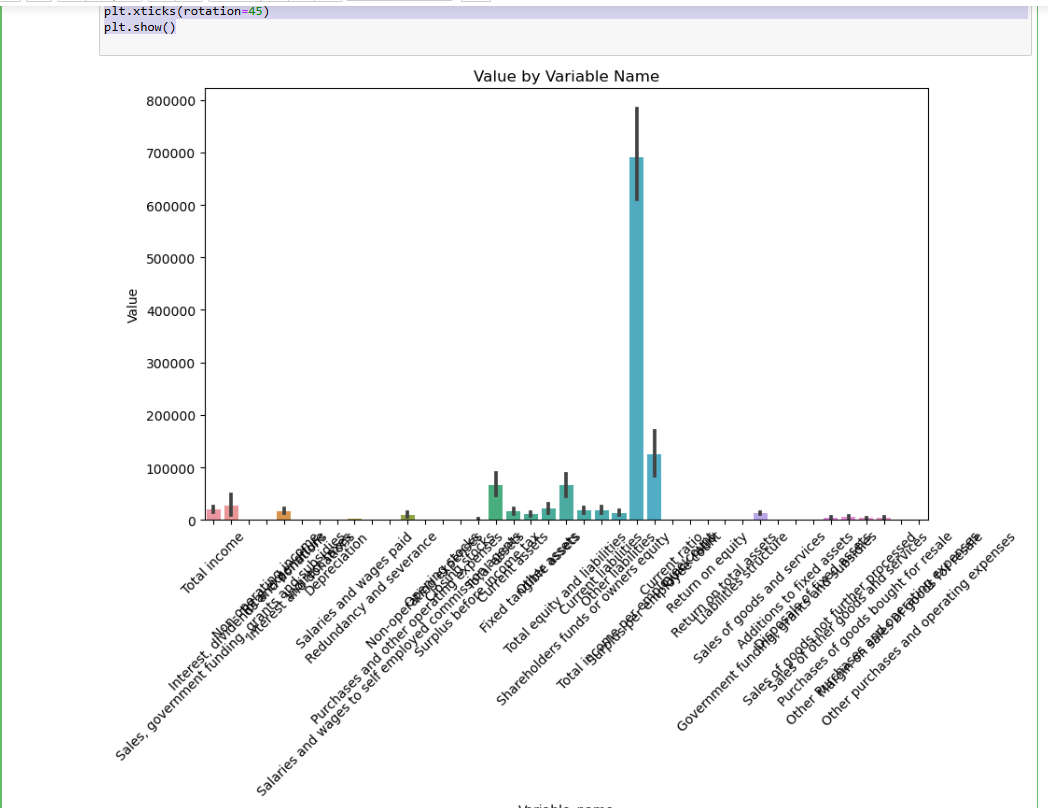
sns.barplot(data=df, x='Variable\_name', y='Value')

plt.title('Value by Variable Name')

plt.xticks(rotation=45)

plt.show()

**Output:**

****

