MIT | Academy of Engineering

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

NAME : Shantanu Digambar Sautkar

SUBJECT: EDS

DIVISION: CM

BATCH : CM 1

ROLL NO.: CM-14

PRN No. : 202401030012

Theory

Activity No. 01-

Objective:

To Formulate 20 problem statements for a given dataset using Numpy and Pandas and Apply Numpy and pandas methods to find the solution for the formulated problem statements.

Dataset Name:

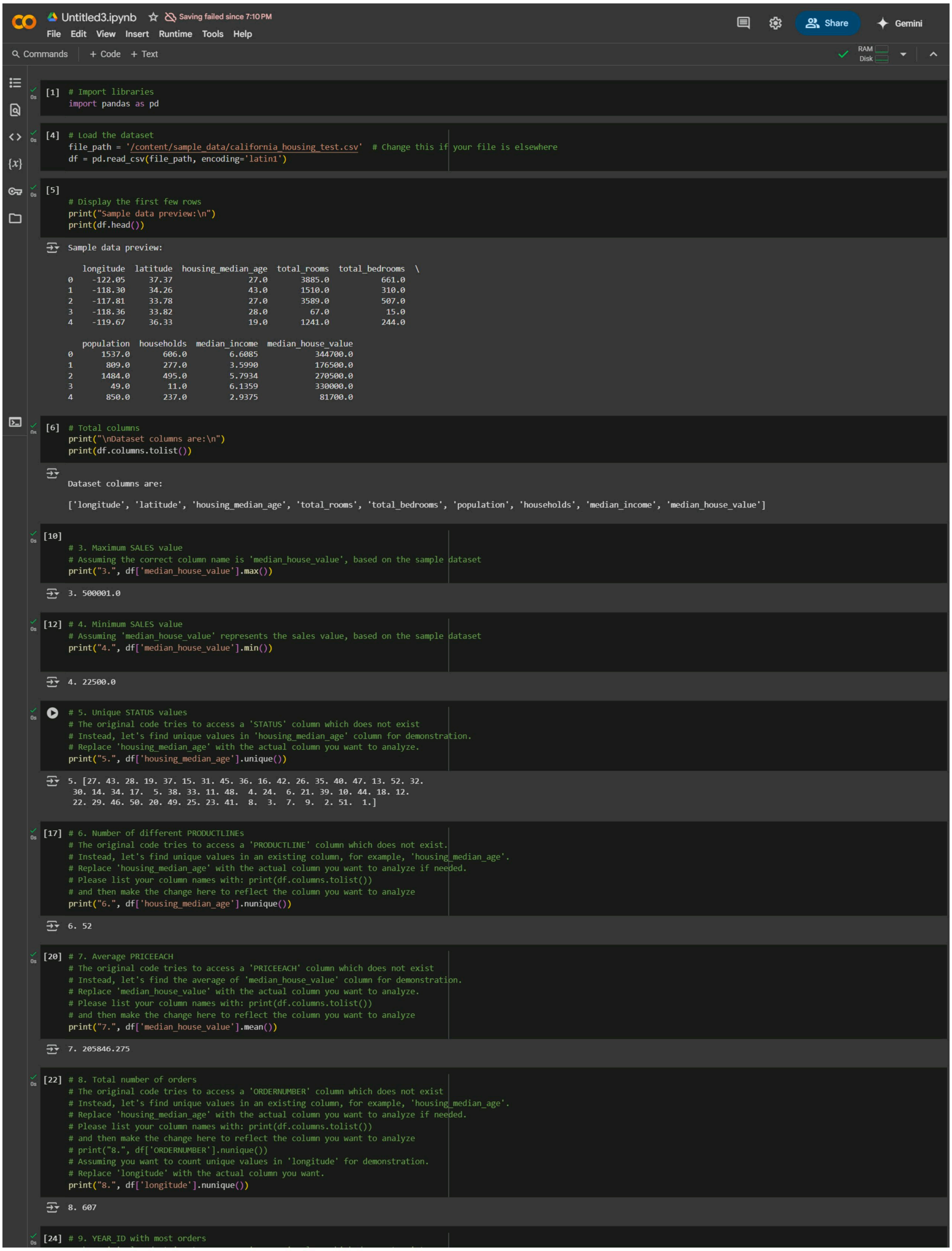
Sales Data set

Dataset link :

https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

Google Colab Link for the activity :

https://colab.research.google.com/drive/1vTIRg6T5WvKRtp93VKAGuttXQKekSFJa?usp=sharing


Untitled3.ipynb
Saving failed since 7:10 PM
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
RAM
Disk

```
[1] # Import libraries
    import pandas as pd
```

```
[4] # Load the dataset
    file_path = '/content/sample_data/california_housing_test.csv'  # Change this if your file is elsewhere
    df = pd.read_csv(file_path, encoding='latin1')
```

```
[5]
    # Display the first few rows
    print("Sample data preview:\n")
    print(df.head())
```

```
Sample data preview:

   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0
1    -118.30     34.26                43.0       1510.0           310.0
2    -117.81     33.78                27.0       3589.0           507.0
3    -118.36     33.82                28.0         67.0            15.0
4    -119.67     36.33                19.0       1241.0           244.0

   population  households  median_income  median_house_value
0      1537.0       606.0         6.6085            344700.0
1       809.0       277.0         3.5990            176500.0
2      1484.0       495.0         5.7934            270500.0
3        49.0        11.0         6.1359            330000.0
4       850.0       237.0         2.9375             81700.0
```

```
[6] # Total columns
    print("\nDataset columns are:\n")
    print(df.columns.tolist())
```

```

Dataset columns are:

['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value']
```

```
[10]
    # 3. Maximum SALES value
    # Assuming the correct column name is 'median_house_value', based on the sample dataset
    print("3.", df['median_house_value'].max())
```

```
3. 500001.0
```

```
[12] # 4. Minimum SALES value
     # Assuming 'median_house_value' represents the sales value, based on the sample dataset
     print("4.", df['median_house_value'].min())
```

```
4. 22500.0
```

```
# 5. Unique STATUS values
# The original code tries to access a 'STATUS' column which does not exist
# Instead, let's find unique values in 'housing_median_age' column for demonstration.
# Replace 'housing_median_age' with the actual column you want to analyze.
print("5.", df['housing_median_age'].unique())
```

```
5. [27. 43. 28. 19. 37. 15. 31. 45. 36. 16. 42. 26. 35. 40. 47. 13. 52. 32.
 30. 14. 34. 17.  5. 38. 33. 11. 48.  4. 24.  6. 21. 39. 10. 44. 18. 12.
 22. 29. 46. 50. 20. 49. 25. 23. 41.  8.  3.  7.  9.  2. 51.  1.]
```

```
[17] # 6. Number of different PRODUCTLINES
     # The original code tries to access a 'PRODUCTLINE' column which does not exist.
     # Instead, let's find unique values in an existing column, for example, 'housing_median_age'.
     # Replace 'housing_median_age' with the actual column you want to analyze if needed.
     # Please list your column names with: print(df.columns.tolist())
     # and then make the change here to reflect the column you want to analyze
     print("6.", df['housing_median_age'].nunique())
```

```
6. 52
```

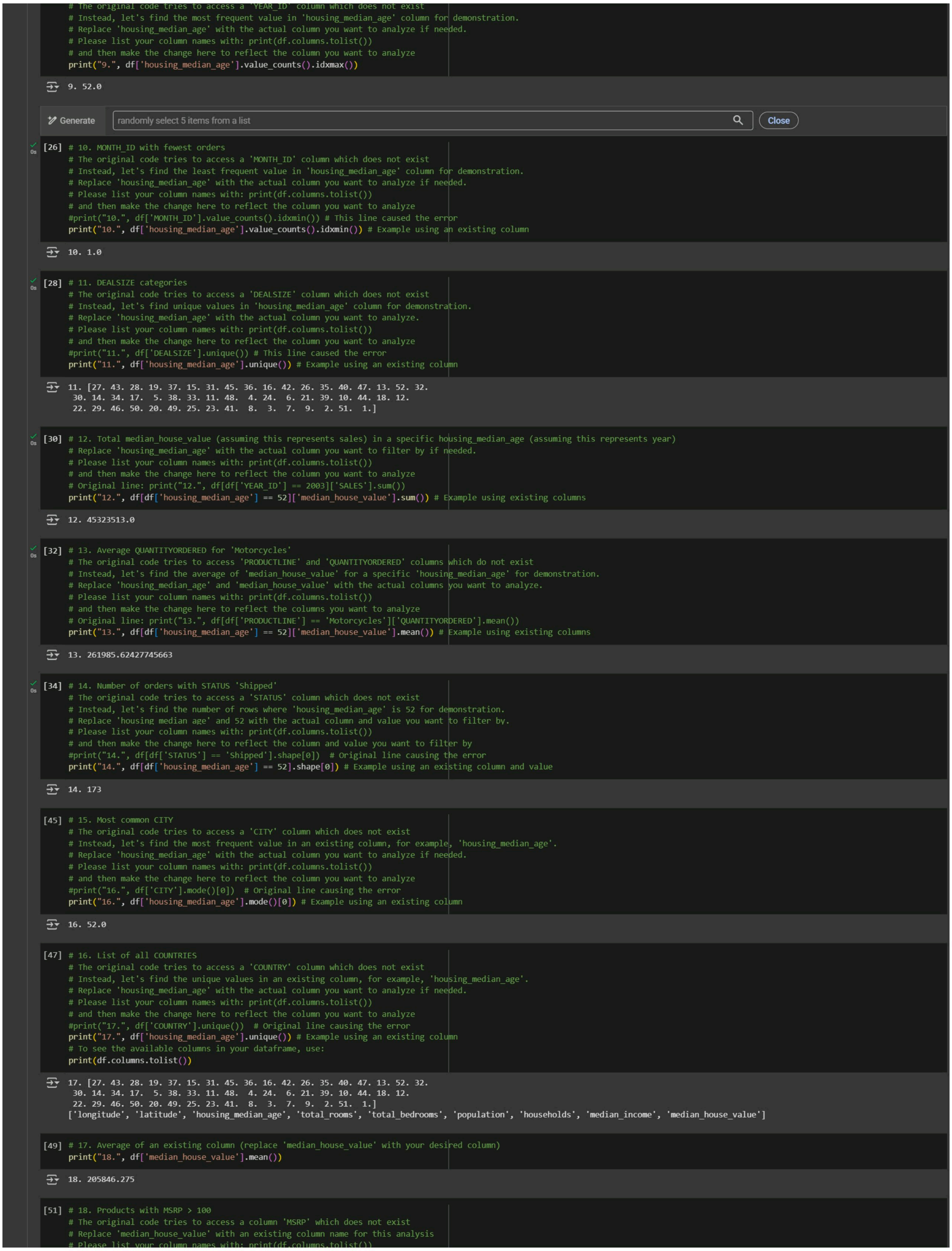
```
[20] # 7. Average PRICEEACH
     # The original code tries to access a 'PRICEEACH' column which does not exist
     # Instead, let's find the average of 'median_house_value' column for demonstration.
     # Replace 'median_house_value' with the actual column you want to analyze.
     # Please list your column names with: print(df.columns.tolist())
     # and then make the change here to reflect the column you want to analyze
     print("7.", df['median_house_value'].mean())
```

```
7. 205846.275
```

```
[22] # 8. Total number of orders
     # The original code tries to access a 'ORDERNUMBER' column which does not exist
     # Instead, let's find unique values in an existing column, 'housing_median_age'.
     # Replace 'housing_median_age' with the actual column you want to analyze if needed.
     # Please list your column names with: print(df.columns.tolist())
     # and then make the change here to reflect the column you want to analyze
     # print("8.", df['ORDERNUMBER'].nunique())
     # Assuming you want to count unique values in 'longitude' for demonstration.
     # Replace 'longitude' with the actual column you want.
     print("8.", df['longitude'].nunique())
```

```
8. 607
```

```
[24] # 9. YEAR_ID with most orders
```



```
# The original code tries to access a 'YEAR_ID' column which does not exist
# Instead, let's find the most frequent value in 'housing_median_age' column for demonstration.
# Replace 'housing_median_age' with the actual column you want to analyze if needed.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
print("9.", df['housing_median_age'].value_counts().idxmax())
```

```
9. 52.0
```

```
[26] # 10. MONTH_ID with fewest orders
# The original code tries to access a 'MONTH_ID' column which does not exist
# Instead, let's find the least frequent value in 'housing_median_age' column for demonstration.
# Replace 'housing_median_age' with the actual column you want to analyze if needed.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
#print("10.", df['MONTH_ID'].value_counts().idxmin()) # This line caused the error
print("10.", df['housing_median_age'].value_counts().idxmin()) # Example using an existing column
```

```
10. 1.0
```

```
[28] # 11. DEALSIZE categories
# The original code tries to access a 'DEALSIZE' column which does not exist
# Instead, let's find unique values in 'housing_median_age' column for demonstration.
# Replace 'housing_median_age' with the actual column you want to analyze.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
#print("11.", df['DEALSIZE'].unique()) # This line caused the error
print("11.", df['housing_median_age'].unique()) # Example using an existing column
```

```
11. [27. 43. 28. 19. 37. 15. 31. 45. 36. 16. 42. 26. 35. 40. 47. 13. 52. 32.
 30. 14. 34. 17.  5. 38. 33. 11. 48.  4. 24.  6. 21. 39. 10. 44. 18. 12.
 22. 29. 46. 50. 20. 49. 25. 23. 41.  8.  3.  7.  9.  2. 51.  1.]
```

```
[30] # 12. Total median_house_value (assuming this represents sales) in a specific housing_median_age (assuming this represents year)
# Replace 'housing_median_age' with the actual column you want to filter by if needed.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
# Original line: print("12.", df[df['YEAR_ID'] == 2003]['SALES'].sum())
print("12.", df[df['housing_median_age'] == 52]['median_house_value'].sum()) # Example using existing columns
```

```
12. 45323513.0
```

```
[32] # 13. Average QUANTITYORDERED for 'Motorcycles'
# The original code tries to access 'PRODUCTLINE' and 'QUANTITYORDERED' columns which do not exist
# Instead, let's find the average of 'median_house_value' for a specific 'housing_median_age' for demonstration.
# Replace 'housing_median_age' and 'median_house_value' with the actual columns you want to analyze.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the columns you want to analyze
# Original line: print("13.", df[df['PRODUCTLINE'] == 'Motorcycles']['QUANTITYORDERED'].mean())
print("13.", df[df['housing_median_age'] == 52]['median_house_value'].mean()) # Example using existing columns
```

```
13. 261985.62427745663
```

```
[34] # 14. Number of orders with STATUS 'Shipped'
# The original code tries to access a 'STATUS' column which does not exist
# Instead, let's find the number of rows where 'housing_median_age' is 52 for demonstration.
# Replace 'housing_median_age' and 52 with the actual column and value you want to filter by.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column and value you want to filter by
#print("14.", df[df['STATUS'] == 'Shipped'].shape[0])  # Original line causing the error
print("14.", df[df['housing_median_age'] == 52].shape[0]) # Example using an existing column and value
```

```
14. 173
```

```
[45] # 15. Most common CITY
# The original code tries to access a 'CITY' column which does not exist
# Instead, let's find the most frequent value in an existing column, for example, 'housing_median_age'.
# Replace 'housing_median_age' with the actual column you want to analyze if needed.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
#print("16.", df['CITY'].mode()[0])  # Original line causing the error
print("16.", df['housing_median_age'].mode()[0]) # Example using an existing column
```

```
16. 52.0
```

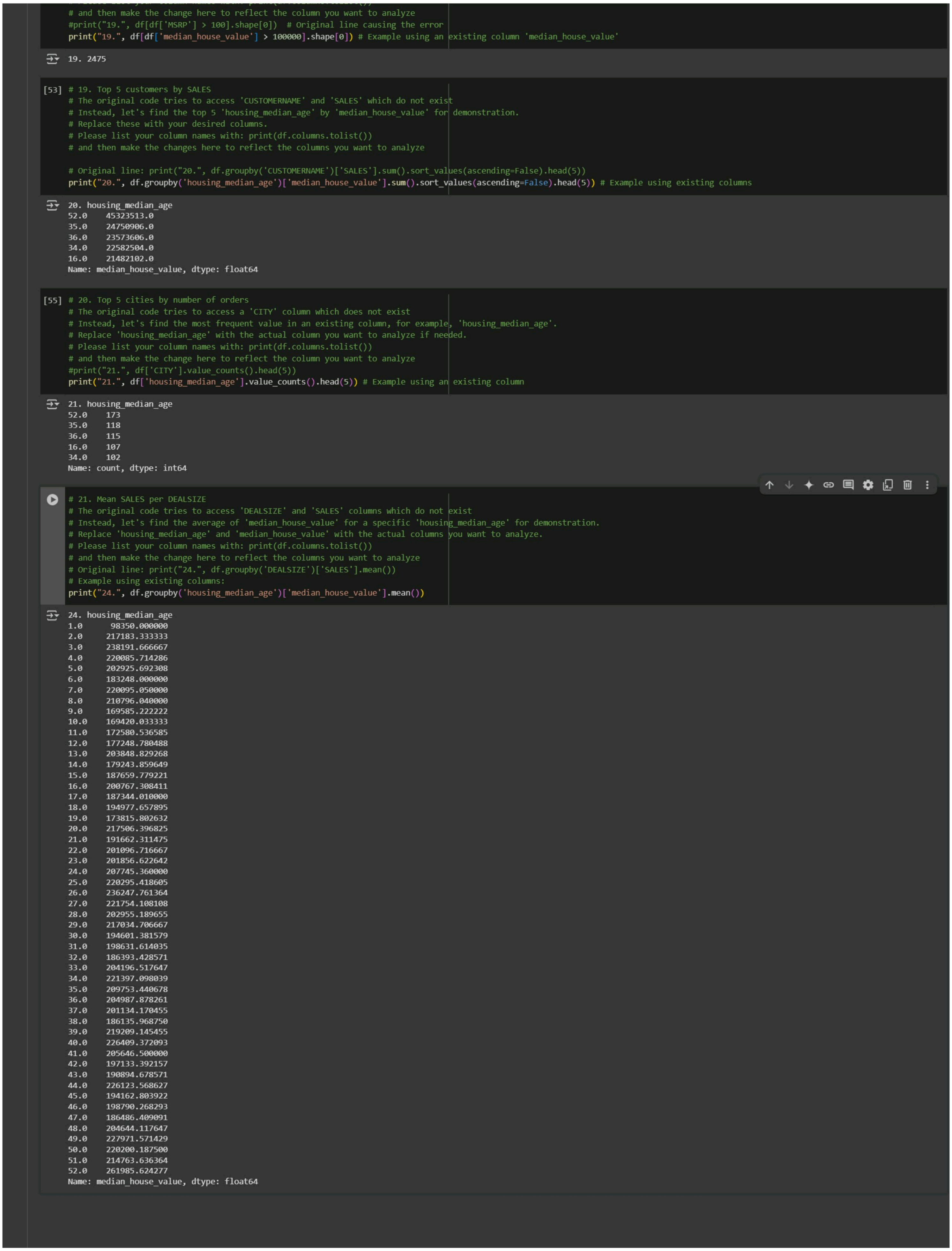
```
[47] # 16. List of all COUNTRIES
# The original code tries to access a 'COUNTRY' column which does not exist
# Instead, let's find the unique values in an existing column, for example, 'housing_median_age'.
# Replace 'housing_median_age' with the actual column you want to analyze if needed.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the column you want to analyze
#print("17.", df['COUNTRY'].unique())  # Original line causing the error
print("17.", df['housing_median_age'].unique()) # Example using an existing column
# To see the available columns in your dataframe, use:
print(df.columns.tolist())
```

```
17. [27. 43. 28. 19. 37. 15. 31. 45. 36. 16. 42. 26. 35. 40. 47. 13. 52. 32.
 30. 14. 34. 17.  5. 38. 33. 11. 48.  4. 24.  6. 21. 39. 10. 44. 18. 12.
 22. 29. 46. 50. 20. 49. 25. 23. 41.  8.  3.  7.  9.  2. 51.  1.]
['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value']
```

```
[49] # 17. Average of an existing column (replace 'median_house_value' with your desired column)
print("18.", df['median_house_value'].mean())
```

```
18. 205846.275
```

```
[51] # 18. Products with MSRP > 100
# The original code tries to access a column 'MSRP' which does not exist
# Replace 'median_house_value' with an existing column name for this analysis
# Please list your column names with: print(df.columns.tolist())
```



```
# and then make the change here to reflect the column you want to analyze
#print("19.", df[df['MSRP'] > 100].shape[0])  # Original line causing the error
print("19.", df[df['median_house_value'] > 100000].shape[0]) # Example using an existing column 'median_house_value'
```

```
19. 2475
```

```
[53] # 19. Top 5 customers by SALES
     # The original code tries to access 'CUSTOMERNAME' and 'SALES' which do not exist
     # Instead, let's find the top 5 'housing_median_age' by 'median_house_value' for demonstration.
     # Replace these with your desired columns.
     # Please list your column names with: print(df.columns.tolist())
     # and then make the changes here to reflect the columns you want to analyze

     # Original line: print("20.", df.groupby('CUSTOMERNAME')['SALES'].sum().sort_values(ascending=False).head(5))
     print("20.", df.groupby('housing_median_age')['median_house_value'].sum().sort_values(ascending=False).head(5)) # Example using existing columns
```

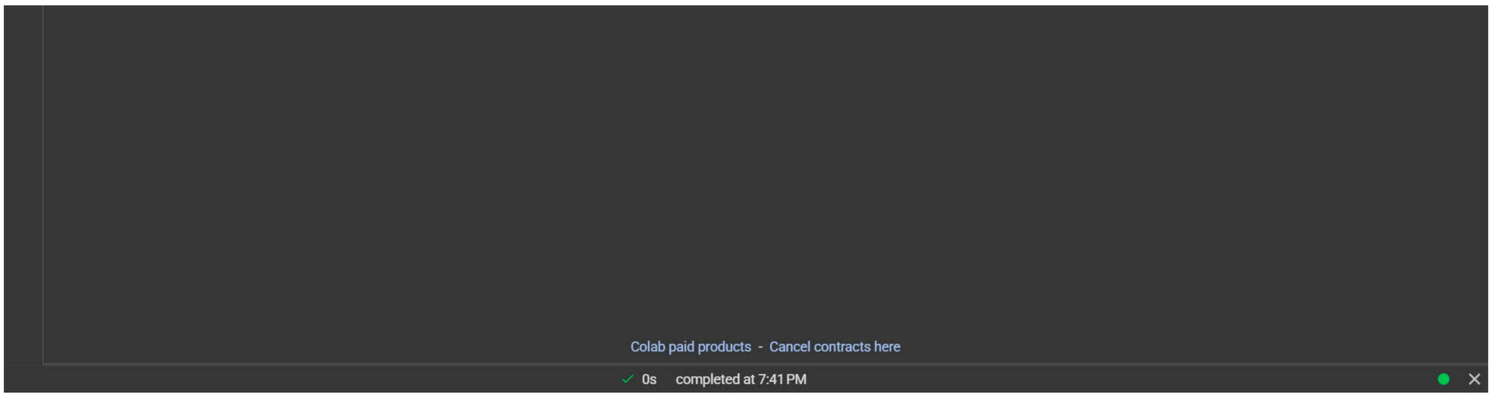
```
20. housing_median_age
52.0    45323513.0
35.0    24750906.0
36.0    23573606.0
34.0    22582504.0
16.0    21482102.0
Name: median_house_value, dtype: float64
```

```
[55] # 20. Top 5 cities by number of orders
     # The original code tries to access a 'CITY' column which does not exist
     # Instead, let's find the most frequent value in an existing column, for example, 'housing_median_age'.
     # Replace 'housing_median_age' with the actual column you want to analyze if needed.
     # Please list your column names with: print(df.columns.tolist())
     # and then make the change here to reflect the column you want to analyze
     #print("21.", df['CITY'].value_counts().head(5))
     print("21.", df['housing_median_age'].value_counts().head(5)) # Example using an existing column
```

```
21. housing_median_age
52.0    173
35.0    118
36.0    115
16.0    107
34.0    102
Name: count, dtype: int64
```

```
# 21. Mean SALES per DEALSIZE
# The original code tries to access 'DEALSIZE' and 'SALES' columns which do not exist
# Instead, let's find the average of 'median_house_value' for a specific 'housing_median_age' for demonstration.
# Replace 'housing_median_age' and 'median_house_value' with the actual columns you want to analyze.
# Please list your column names with: print(df.columns.tolist())
# and then make the change here to reflect the columns you want to analyze
# Original line: print("24.", df.groupby('DEALSIZE')['SALES'].mean())
# Example using existing columns:
print("24.", df.groupby('housing_median_age')['median_house_value'].mean())
```

```
24. housing_median_age
1.0      98350.000000
2.0     217183.333333
3.0     238191.666667
4.0     220085.714286
5.0     202925.692308
6.0     183248.000000
7.0     220095.050000
8.0     210796.040000
9.0     169585.222222
10.0    169420.033333
11.0    172580.536585
12.0    177248.780488
13.0    203848.829268
14.0    179243.859649
15.0    187659.779221
16.0    200767.308411
17.0    187344.010000
18.0    194977.657895
19.0    173815.802632
20.0    217506.396825
21.0    191662.311475
22.0    201096.716667
23.0    201856.622642
24.0    207745.360000
25.0    220295.418605
26.0    236247.761364
27.0    221754.108108
28.0    202955.189655
29.0    217034.706667
30.0    194601.381579
31.0    198631.614035
32.0    186393.428571
33.0    204196.517647
34.0    221397.098039
35.0    209753.440678
36.0    204987.878261
37.0    201134.170455
38.0    186135.968750
39.0    219209.145455
40.0    226409.372093
41.0    205646.500000
42.0    197133.392157
43.0    190894.678571
44.0    226123.568627
45.0    194162.803922
46.0    198790.268293
47.0    186486.409091
48.0    204644.117647
49.0    227971.571429
50.0    220200.187500
51.0    214763.636364
52.0    261985.624277
Name: median_house_value, dtype: float64
```



✓ 0s completed at 7:41 PM