Summary Of Task 1 (Data Cleaning And Preparation)

■ Software Used = Jupyter Notebook

Step 1: Load the Dataset

- Used pandas to load the CSV file into a DataFrame.
- Previewed the top rows to understand the structure.

Step 2: Understand the Dataset

- Checked the shape (9800 rows × 18 columns).
- Inspected column names, data types, and summary statistics.
- Identified the presence of missing values and data types (numeric, text, dates, etc.).

Step 3: Handle Missing Values

- Found that only postal_code had missing values.
- Filled missing values in postal_code with the most frequent value (mode).
- Verified there were no missing values left using .isnull().sum().

Step 4: Remove Duplicates

- Checked for duplicate rows using df.duplicated().sum().
- Removed all duplicates with drop_duplicates().

Step 5: Clean Text Columns

- Replaced spaces with underscores in string columns for consistency.
- Cleaned column headers: made lowercase and replaced spaces with underscores.

Step 6: Convert Data Types

- Converted order_date and ship_date to datetime format.
- Converted postal_code to integer after filling missing values.
- Converted categorical columns like segment, category, region, etc. to category dtype for better performance.

Step 7: Handle Outliers

- Detected outliers in the sales column using the IQR method.
- Removed rows with sales values outside the range of 1.5 × IQR.
- Created a new cleaned dataset df_cleaned.

Step 8: Encode Categorical Variables

- Identified text-based columns and applied One-Hot Encoding using pd.get_dummies().
- This converted categorical columns to numeric format suitable for ML models.

Step 9: Normalize Numerical Columns

- Selected numeric columns and applied Standard Scaling using StandardScaler.
- This gave all numeric columns a mean of 0 and standard deviation of 1.

Final Result:

- Cleaned and prepared dataset with:
- No missing or duplicate values
- All columns correctly typed
- No outliers in sales
- All categorical data encoded
- All numerical data scaled

CODE

~~~
## Step 1: Import Required Libraries
```python
import pandas as pd
```

## Step 2: Load the Dataset

df = pd.read_csv('path/to/train.csv')
df.head()  # Preview the data
```

---

## Step 3: Basic Exploration
# Shape of the dataset
print("Shape:", df.shape)

# Data types and column info
df.info()

# Summary statistics
df.describe(include='all')
```

---

## Step 4: Missing Values Analysis

missing = df.isnull().sum()
missing_percent = (missing / len(df)) * 100
missing_data = pd.DataFrame({'Missing Values': missing, 'Percent (%)': missing_percent})
missing_data = missing_data[missing_data["Missing Values"] > 0]
missing_data.sort_values(by='Percent (%)', ascending=False)
```
~~~

Step 5: Fill Missing Values in 'postal_code'
```
mode_postal = df['Postal Code'].mode()[0]
df['Postal Code'] = df['Postal Code'].fillna(mode_postal)
```

Step 6: Remove Duplicate Rows
```
df.drop_duplicates(inplace=True)
```

Step 7: Clean Text Columns and Column Headers

```
# Replace spaces in string values with underscores
text_cols = df.select_dtypes(include='object').columns
for col in text_cols:
    df[col] = df[col].str.replace(" ", "_", regex=False)

# Clean column names
df.columns = df.columns.str.lower().str.strip().str.replace(' ', '_')
```

Step 8: Convert Data Types
```
# Convert date columns
df['order_date'] = pd.to_datetime(df['order_date'], errors='coerce')
df['ship_date'] = pd.to_datetime(df['ship_date'], errors='coerce')

# Convert postal_code to integer
df['postal_code'] = df['postal_code'].astype(int)

# Convert selected columns to category dtype
cat_cols = ['ship_mode', 'segment', 'country', 'city', 'state', 'region', 'category', 'sub-category']
for col in cat_cols:
    df[col] = df[col].astype('category')
```

Step 9: Handle Outliers in 'sales'
```
Q1 = df['sales'].quantile(0.25)
Q3 = df['sales'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df_cleaned = df[(df['sales'] >= lower_bound) & (df['sales'] <= upper_bound)]
```

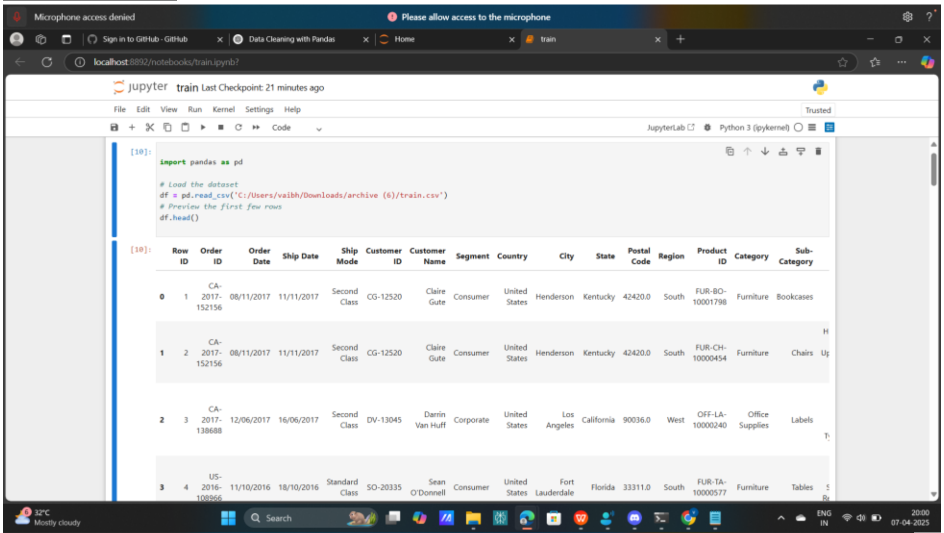
```
## Step 10: Encode Categorical Variables (One-Hot Encoding)
categorical_cols = df_cleaned.select_dtypes(include='object').columns
df_encoded = pd.get_dummies(df_cleaned, columns=categorical_cols, drop_first=True)
```

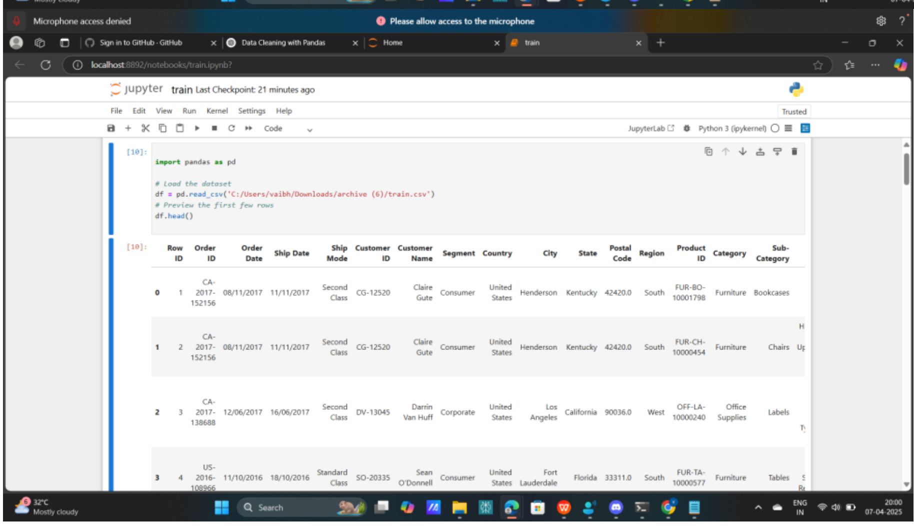
```
## Step 11: Feature Scaling
from sklearn.preprocessing import StandardScaler

numeric_cols = df_encoded.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
df_encoded[numeric_cols] = scaler.fit_transform(df_encoded[numeric_cols])

```

IMAGES





jupyter train Last Checkpoint: 21 minutes ago

```
16  Product Name   9800 non-null   object
17  Sales          9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

[12]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
count	9800.000000	9800	9800	9800	9800	9800	9800	9800	9800	9800	9800	9789.000000	9800	9800	9800
unique	NaN	4922	1230	1326	4	793	793	3	1	529	49	NaN	4	1861	3
top	NaN	CA-2018-100111	05/09/2017	26/09/2018	Standard Class	WB-21850	William Brown	Consumer	United States	New York City	California	NaN	West	OFF-PA-10001970	Office Supplies
freq	NaN	14	38	34	5859	35	35	5101	9800	891	1946	NaN	3140	19	5909
mean	4900.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	55273.322403	NaN	NaN	NaN
std	2829.160653	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	32041.223413	NaN	NaN	NaN
min	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1040.000000	NaN	NaN	NaN
25%	2450.750000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23223.000000	NaN	NaN	NaN
50%	4900.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	58103.000000	NaN	NaN	NaN
75%	7350.250000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	90008.000000	NaN	NaN	NaN
max	9800.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	99301.000000	NaN	NaN	NaN

[14]:

```python
# Checks missing values per column
missing = df.isnull().sum()
missing_percent = (missing / len(df)) * 100
```

jupyter train Last Checkpoint: 21 minutes ago

[12]:

```python
# Checks the shape of the dataset
print("Shape:", df.shape)

# Checks data types and missing values info
df.info()

# Summary statistics for both numerical and categorical columns
df.describe(include='all')
```

```
Shape: (9800, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   int64
 1   Order ID       9800 non-null   object
 2   Order Date     9800 non-null   object
 3   Ship Date      9800 non-null   object
 4   Ship Mode      9800 non-null   object
 5   Customer ID    9800 non-null   object
 6   Customer Name  9800 non-null   object
 7   Segment        9800 non-null   object
 8   Country        9800 non-null   object
 9   City           9800 non-null   object
 10  State          9800 non-null   object
 11  Postal Code    9789 non-null   float64
 12  Region         9800 non-null   object
 13  Product ID     9800 non-null   object
 14  Category       9800 non-null   object
 15  Sub-Category   9800 non-null   object
 16  Product Name   9800 non-null   object
```

jupyter train Last Checkpoint: 21 minutes ago

[14]:

```python
# Checks missing values per column
missing = df.isnull().sum()
missing_percent = (missing / len(df)) * 100

# Display columns with missing data
missing_data = pd.DataFrame({
    'Missing Values': missing,
    'Percent (%)': missing_percent
})
missing_data = missing_data[missing_data["Missing Values"] > 0]
print(missing_data)
```

```
             Missing Values  Percent (%)
Postal Code              11     0.112245
```

[16]:

```python
#  missing values in 'Postal Code' with the mode (most frequent value)
mode_postal = df['Postal Code'].mode()[0]
df['Postal Code'] = df['Postal Code'].fillna(mode_postal)
```

[18]:

```python
# Count duplicates
print("Duplicate rows:", df.duplicated().sum())

# Drop duplicates
df.drop_duplicates(inplace=True)

# Confirmed  duplicates are removed
print("Duplicates after cleaning:", df.duplicated().sum())
```

```
Duplicate rows: 0
Duplicates after cleaning: 0
```

```
Duplicate rows: 0
Duplicates after cleaning: 0
```

[20]:
```python
# Clean column headers: lowercase + replace spaces with underscores
df.columns = df.columns.str.lower().str.strip().str.replace(' ', '_')

# Replace internal spaces in string/text columns with underscores
text_cols = df.select_dtypes(include='object').columns
for col in text_cols:
    df[col] = df[col].str.replace(" ", "_")
```

[22]:
```python
# Convert date columns to datetime format
df['order_date'] = pd.to_datetime(df['order_date'], errors='coerce')
df['ship_date'] = pd.to_datetime(df['ship_date'], errors='coerce')

# Convert 'postal_code' to integer
df['postal_code'] = df['postal_code'].astype(int)

# Convert some columns to 'category' type to save memory
cat_cols = ['ship_mode', 'segment', 'country', 'city', 'state',
            'region', 'category', 'sub-category']
for col in cat_cols:
    df[col] = df[col].astype('category')
```

[24]:
```python
# Use IQR to remove outliers from 'sales'
Q1 = df['sales'].quantile(0.25)
Q3 = df['sales'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```


[24]:
```python
# Use IQR to remove outliers from 'sales'
Q1 = df['sales'].quantile(0.25)
Q3 = df['sales'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```


```python
# Filter out outliers
df_cleaned = df[(df['sales'] >= lower_bound) & (df['sales'] <= upper_bound)]

# Optional: Check how many rows were removed
print("Rows removed due to outliers:", df.shape[0] - df_cleaned.shape[0])
```

```
Rows removed due to outliers: 1145
```

[26]:
```python
# Identify categorical columns (object or category types)
categorical_cols = df_cleaned.select_dtypes(include=['object', 'category']).columns

# Apply One-Hot Encoding
df_encoded = pd.get_dummies(df_cleaned, columns=categorical_cols, drop_first=True)

# Check new shape after encoding
print("Shape after encoding:", df_encoded.shape)
```

```
Shape after encoding: (8655, 10346)
```

[28]:
```python
from sklearn.preprocessing import StandardScaler

# Select numeric columns
numeric_cols = df_encoded.select_dtypes(include=['int64', 'float64']).columns
```


[28]:
```python
from sklearn.preprocessing import StandardScaler

# Select numeric columns
numeric_cols = df_encoded.select_dtypes(include=['int64', 'float64']).columns
```

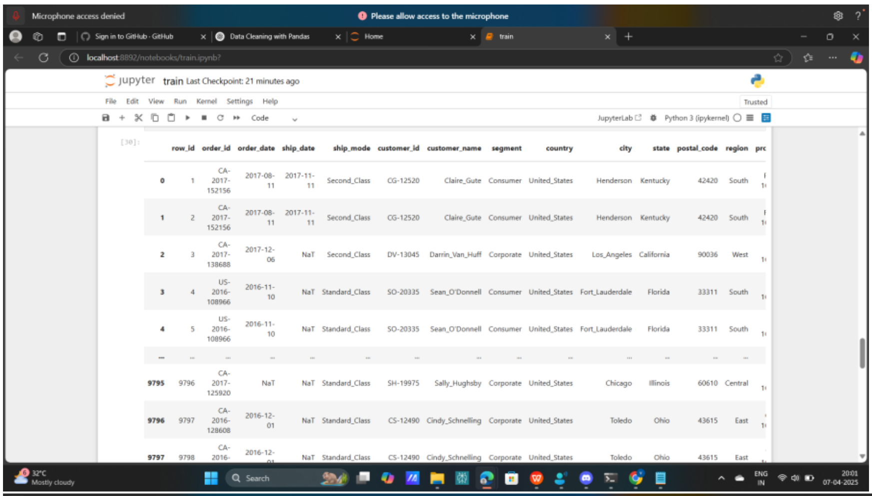

```python
# Apply Standard Scaling
scaler = StandardScaler()
df_encoded[numeric_cols] = scaler.fit_transform(df_encoded[numeric_cols])

# Check the result after scaling
df_encoded[numeric_cols].describe().round(2)
```

[28]:

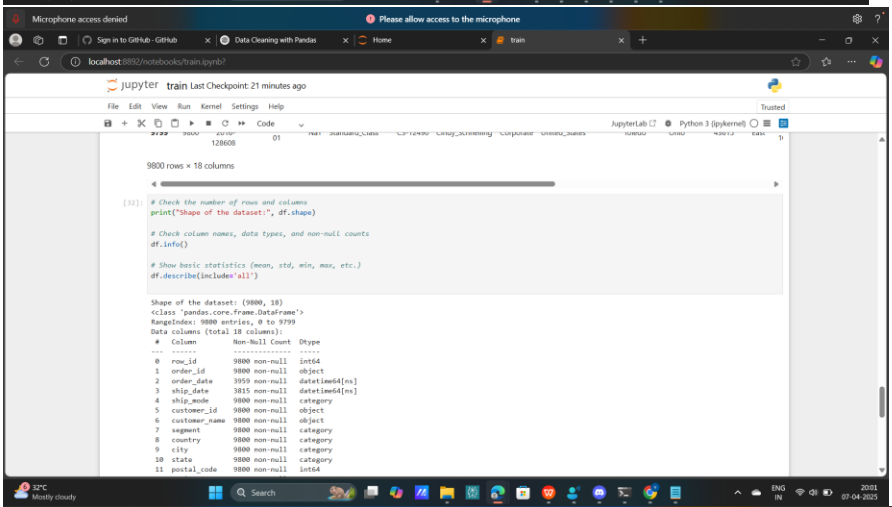
	row_id	postal_code	sales
count	8655.00	8655.00	8655.00
mean	0.00	0.00	0.00
std	1.00	1.00	1.00
min	-1.74	-1.70	-0.81
25%	-0.87	-1.00	-0.68
50%	0.00	0.14	-0.46
75%	0.86	1.08	0.27
max	1.73	1.37	3.55

[30]:
```python
df
```



[30]:

	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment	country	city	state	postal_code	region
0	1	CA-2017-152156	2017-08-11	2017-11-11	Second_Class	CG-12520	Claire_Gute	Consumer	United_States	Henderson	Kentucky	42420	South
1	2	CA-2017-152156	2017-08-11	2017-11-11	Second_Class	CG-12520	Claire_Gute	Consumer	United_States	Henderson	Kentucky	42420	South
2	3	CA-2017-138688	2017-12-06	NaT	Second_Class	DV-13045	Darrin_Van_Huff	Corporate	United_States	Los_Angeles	California	90036	West
3	4	US-2016-108966	2016-11-10	NaT	Standard_Class	SO-20335	Sean_O'Donnell	Consumer	United_States	Fort_Lauderdale	Florida	33311	South
4	5	US-2016-108966	2016-11-10	NaT	Standard_Class	SO-20335	Sean_O'Donnell	Consumer	United_States	Fort_Lauderdale	Florida	33311	South
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9795	9796	CA-2017-125920	NaT	NaT	Standard_Class	SH-19975	Sally_Hughsby	Corporate	United_States	Chicago	Illinois	60610	Central
9796	9797	CA-2016-128608	2016-12-01	NaT	Standard_Class	CS-12490	Cindy_Schnelling	Corporate	United_States	Toledo	Ohio	43615	East
9797	9798	CA-2016-	2016-12-	NaT	Standard_Class	CS-12490	Cindy_Schnelling	Corporate	United_States	Toledo	Ohio	43615	East



9800 rows × 18 columns

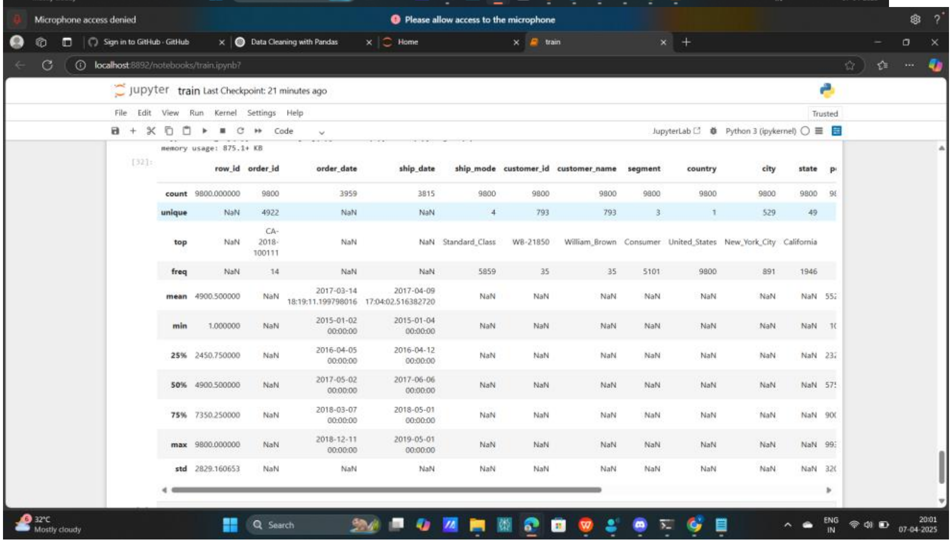
[32]:

```python
# Check the number of rows and columns
print("Shape of the dataset:", df.shape)

# Check column names, data types, and non-null counts
df.info()

# Show basic statistics (mean, std, min, max, etc.)
df.describe(include='all')
```

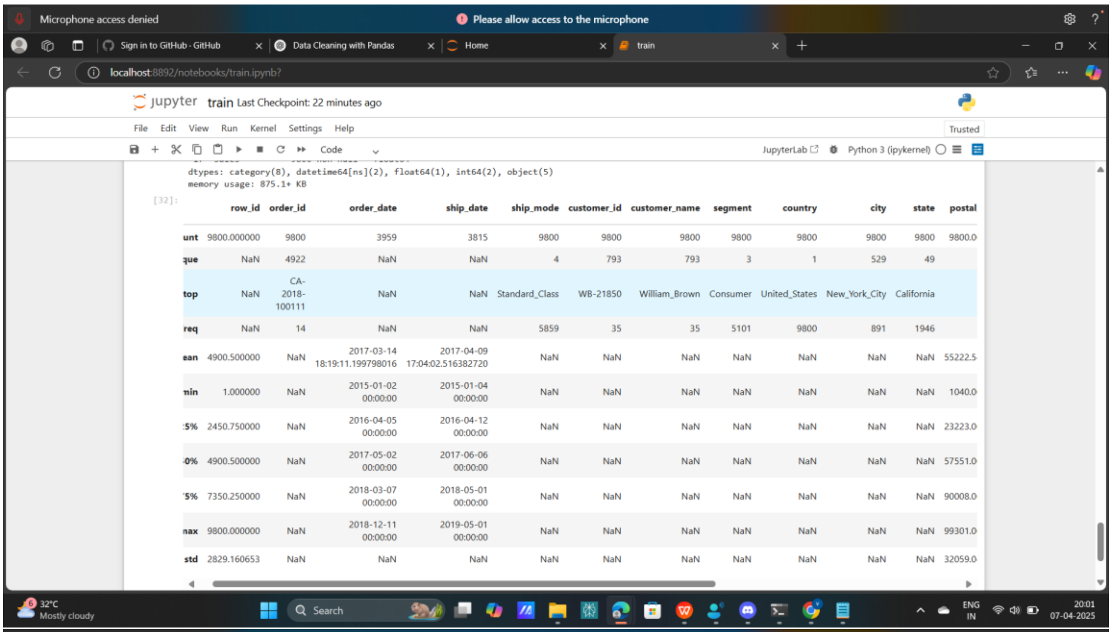
```
Shape of the dataset: (9800, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   row_id         9800 non-null   int64
 1   order_id       9800 non-null   object
 2   order_date     3959 non-null   datetime64[ns]
 3   ship_date      3815 non-null   datetime64[ns]
 4   ship_mode      9800 non-null   category
 5   customer_id    9800 non-null   object
 6   customer_name  9800 non-null   object
 7   segment        9800 non-null   category
 8   country        9800 non-null   category
 9   city           9800 non-null   category
 10  state          9800 non-null   category
 11  postal_code    9800 non-null   int64
```



```
memory usage: 875.1+ KB
```

[32]:

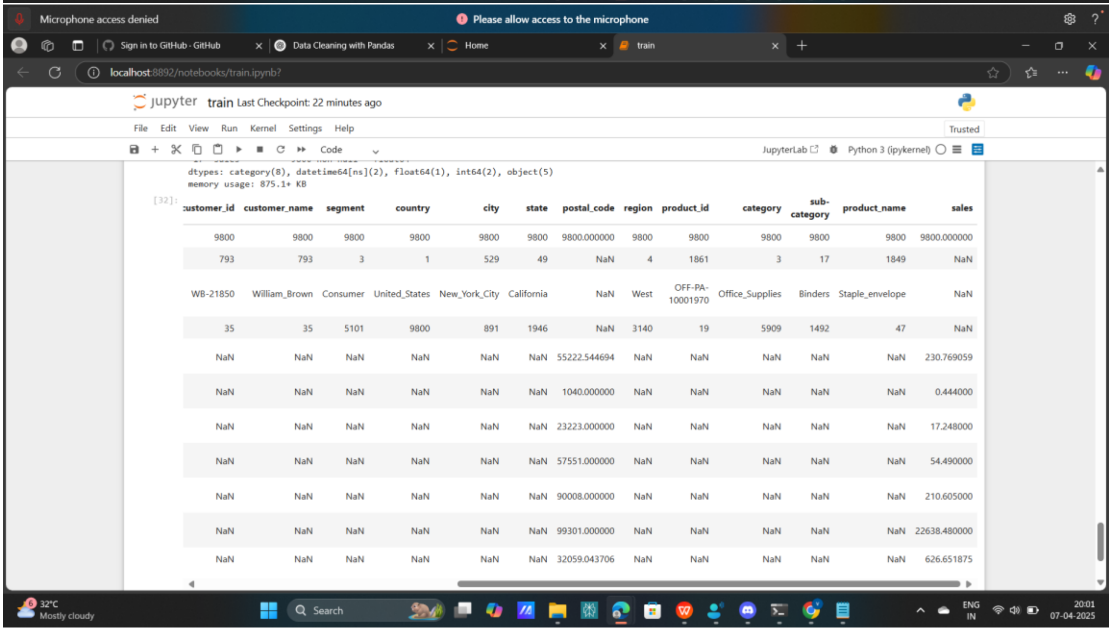
	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment	country	city	state
count	9800.000000	9800	3959	3815	9800	9800	9800	9800	9800	9800	9800
unique	NaN	4922	NaN	NaN	4	793	793	3	1	529	49
top	NaN	CA-2018-100111	NaN	NaN	Standard_Class	WB-21850	William_Brown	Consumer	United_States	New_York_City	California
freq	NaN	14	NaN	NaN	5859	35	35	5101	9800	891	1946
mean	4900.500000	NaN	2017-03-14 18:19:11.199798016	2017-04-09 17:04:02.516382720	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	1.000000	NaN	2015-01-02 00:00:00	2015-01-04 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	2450.750000	NaN	2016-04-05 00:00:00	2016-04-12 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	4900.500000	NaN	2017-05-02 00:00:00	2017-06-06 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	7350.250000	NaN	2018-03-07 00:00:00	2018-05-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	9800.000000	NaN	2018-12-11 00:00:00	2019-05-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	2829.160653	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



```
dtypes: category(8), datetime64[ns](2), float64(1), int64(2), object(5)
memory usage: 875.1+ KB
```

[32]:

	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment	country	city	state	postal
unt	9800.000000	9800	3959	3815	9800	9800	9800	9800	9800	9800	9800	9800.0
que	NaN	4922	NaN	NaN	4	793	793	3	1	529	49	
top	NaN	CA-2018-100111	NaN	NaN	Standard_Class	WB-21850	William_Brown	Consumer	United_States	New_York_City	California	
req	NaN	14	NaN	NaN	5859	35	35	5101	9800	891	1946	
ean	4900.500000	NaN	2017-03-14 18:19:11.199798016	2017-04-09 17:04:02.516382720	NaN	NaN	NaN	NaN	NaN	NaN	NaN	55222.5
nin	1.000000	NaN	2015-01-02 00:00:00	2015-01-04 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1040.0
5%	2450.750000	NaN	2016-04-05 00:00:00	2016-04-12 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23223.0
0%	4900.500000	NaN	2017-05-02 00:00:00	2017-06-06 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	57551.0
5%	7350.250000	NaN	2018-03-07 00:00:00	2018-05-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	90008.0
nax	9800.000000	NaN	2018-12-11 00:00:00	2019-05-01 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	99301.0
std	2829.160653	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	32059.0



```
dtypes: category(8), datetime64[ns](2), float64(1), int64(2), object(5)
memory usage: 875.1+ KB
```

[32]:

ustomer_id	customer_name	segment	country	city	state	postal_code	region	product_id	category	sub-category	product_name	sales
9800	9800	9800	9800	9800	9800	9800.000000	9800	9800	9800	9800	9800	9800.000000
793	793	3	1	529	49	NaN	4	1861	3	17	1849	NaN
WB-21850	William_Brown	Consumer	United_States	New_York_City	California	NaN	West	OFF-PA-10001970	Office_Supplies	Binders	Staple_envelope	NaN
35	35	5101	9800	891	1946	NaN	3140	19	5909	1492	47	NaN
NaN	NaN	NaN	NaN	NaN	NaN	55222.544694	NaN	NaN	NaN	NaN	NaN	230.769059
NaN	NaN	NaN	NaN	NaN	NaN	1040.000000	NaN	NaN	NaN	NaN	NaN	0.444000
NaN	NaN	NaN	NaN	NaN	NaN	23223.000000	NaN	NaN	NaN	NaN	NaN	17.248000
NaN	NaN	NaN	NaN	NaN	NaN	57551.000000	NaN	NaN	NaN	NaN	NaN	54.490000
NaN	NaN	NaN	NaN	NaN	NaN	90008.000000	NaN	NaN	NaN	NaN	NaN	210.605000
NaN	NaN	NaN	NaN	NaN	NaN	99301.000000	NaN	NaN	NaN	NaN	NaN	22638.480000
NaN	NaN	NaN	NaN	NaN	NaN	32059.043706	NaN	NaN	NaN	NaN	NaN	626.651875

END