MT1_148

June 1, 2023

1 MT1

2 1. Import Libraries and Load Dataset

[1]:
```
# Importing librares
import os  # to provide a way to use operating system-dependent functionalities.
 ↪ It allows you to interact with the underlying operating system, such as␣
 ↪reading or writing files, navigating directories, etc
import sys  # to provide access to some variables used or maintained by the␣
 ↪interpreter and functions that interact with the interpreter. It allows you␣
 ↪to manipulate the Python runtime environment
import warnings  # to provide a way to handle warnings issued by Python or␣
 ↪external libraries. It allows you to control the display and handling of␣
 ↪warning messages
import numpy as np  # to provide support for large, multi-dimensional arrays␣
 ↪and matrices, along with a wide range of mathematical functions to operate␣
 ↪on these arrays efficiently
import pandas as pd  # to provide high-performance data structures like␣
 ↪DataFrames and Series, along with a wide range of functions to manipulate,␣
 ↪clean, and analyze structured data
import matplotlib.pyplot as plt  # to provide a wide variety of functions to␣
 ↪create different types of plots and visualizations, allowing you to␣
 ↪customize the appearance of your plots extensively
import seaborn as sns  # to provide a high-level interface for creating␣
 ↪attractive and informative statistical graphics, making it easier to explore␣
 ↪and understand data visually
warnings.filterwarnings("ignore")  # to ensure that warning messages are not␣
 ↪displayed during the execution of your code
```

[2]:
```
# load in clean and tidy data and create workfile
df = pd.read_csv("df_milan-1.csv")
```

[3]:
```
df.head()
```

[3]:

	hotel_id	city	distance	stars	rating	country	city_actual	\
0	9797.0	Milan	14.0	NaN	4.1	Italy	Abbiategrasso	
1	9798.0	Milan	10.0	4.0	4.1	Italy	Agrate Brianza	

```
2    9799.0  Milan      12.0    3.0     4.3   Italy          Arcore
3    9800.0  Milan      13.0    3.0     3.5   Italy          Arcore
4    9801.0  Milan      13.0    4.0     4.6   Italy          Arcore

   rating_reviewcount center1label          center2label   neighbourhood  \
0                49.0  City centre  Milan Romolo Station   Abbiategrasso
1               113.0  City centre  Milan Romolo Station  Agrate Brianza
2                 8.0  City centre  Milan Romolo Station          Arcore
3                22.0  City centre  Milan Romolo Station          Arcore
4                74.0  City centre  Milan Romolo Station          Arcore

   ratingta  ratingta_count  distance_alter accommodation_type
0       3.5            90.0            12.0                Inn
1       4.0           910.0            12.0              Hotel
2       3.5           116.0            14.0              Hotel
3       3.5            80.0            15.0              Hotel
4       4.5           218.0            15.0              Hotel
```

[4]:
```
#Summary Statistics
df.describe()
```

[4]:
```
           hotel_id     distance       stars      rating  rating_reviewcount  \
count   1098.000000  1098.000000  541.000000  871.000000          871.000000
mean   10345.500000     4.002095    3.395564    3.849369          100.878301
std      317.109603     5.662733    0.897372    0.712642          164.442489
min     9797.000000     0.000000    1.000000    1.000000            1.000000
25%    10071.250000     1.100000    3.000000    3.500000            8.000000
50%    10345.500000     1.700000    4.000000    4.000000           44.000000
75%    10619.750000     3.800000    4.000000    4.300000          119.500000
max    10894.000000    30.000000    5.000000    5.000000         1445.000000

         ratingta  ratingta_count  distance_alter
count  722.000000      722.000000     1098.000000
mean     3.772853      358.005540        5.024772
std      0.685948      537.190405        5.520723
min      1.000000        1.000000        0.200000
25%      3.500000       60.000000        2.000000
50%      4.000000      170.000000        3.100000
75%      4.000000      456.000000        4.775000
max      5.000000     6697.000000       31.000000
```

2.1 Question 1: Get df as table and summary statistics for df and compare the variables in both results. If there is difference between column numbers between summary statistics and df tables explain why there is difference.

[5]:
```
# See columns
print(df.columns)
print(df.describe())
```

```
Index(['hotel_id', 'city', 'distance', 'stars', 'rating', 'country',
       'city_actual', 'rating_reviewcount', 'center1label', 'center2label',
       'neighbourhood', 'ratingta', 'ratingta_count', 'distance_alter',
       'accommodation_type'],
      dtype='object')
          hotel_id     distance       stars      rating  rating_reviewcount  \
count   1098.000000  1098.000000  541.000000  871.000000          871.000000
mean   10345.500000     4.002095    3.395564    3.849369          100.878301
std      317.109603     5.662733    0.897372    0.712642          164.442489
min     9797.000000     0.000000    1.000000    1.000000            1.000000
25%    10071.250000     1.100000    3.000000    3.500000            8.000000
50%    10345.500000     1.700000    4.000000    4.000000           44.000000
75%    10619.750000     3.800000    4.000000    4.300000          119.500000
max    10894.000000    30.000000    5.000000    5.000000         1445.000000

         ratingta  ratingta_count  distance_alter
count  722.000000      722.000000     1098.000000
mean     3.772853      358.005540        5.024772
std      0.685948      537.190405        5.520723
min      1.000000        1.000000        0.200000
25%      3.500000       60.000000        2.000000
50%      4.000000      170.000000        3.100000
75%      4.000000      456.000000        4.775000
max      5.000000     6697.000000       31.000000
```

[6]:
```
df.shape
```

[6]:
```
(1098, 15)
```

3 2. Finding numerical variables and feature engineering for numerical variables

3.1 Question 2: Report datatypes in data as df and how many numerical and catogorical variables there are in data only according to this information.

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1098 entries, 0 to 1097
```

```
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   hotel_id             1098 non-null   float64
 1   city                 1098 non-null   object
 2   distance             1098 non-null   float64
 3   stars                541 non-null    float64
 4   rating               871 non-null    float64
 5   country              1098 non-null   object
 6   city_actual          1098 non-null   object
 7   rating_reviewcount   871 non-null    float64
 8   center1label         1098 non-null   object
 9   center2label         1098 non-null   object
 10  neighbourhood        1098 non-null   object
 11  ratingta             722 non-null    float64
 12  ratingta_count       722 non-null    float64
 13  distance_alter       1098 non-null   float64
 14  accommodation_type   1096 non-null   object
dtypes: float64(8), object(7)
memory usage: 128.8+ KB
```

[8]:
```
# find numerical variables
numerical = df.select_dtypes(include=['int64', 'float64']).columns
print('There are {} numerical variables\n'.format(len(numerical)))
print('The numerical variables are :', numerical)
```

```
There are 8 numerical variables

The numerical variables are : Index(['hotel_id', 'distance', 'stars', 'rating',
'rating_reviewcount',
       'ratingta', 'ratingta_count', 'distance_alter'],
      dtype='object')
```

[9]:
```
# view the numerical variables
df[numerical].head()
```

[9]:

	hotel_id	distance	stars	rating	rating_reviewcount	ratingta	ratingta_count	distance_alter
0	9797.0	14.0	NaN	4.1	49.0	3.5	90.0	12.0
1	9798.0	10.0	4.0	4.1	113.0	4.0	910.0	12.0
2	9799.0	12.0	3.0	4.3	8.0	3.5	116.0	14.0
3	9800.0	13.0	3.0	3.5	22.0	3.5		
4	9801.0	13.0	4.0	4.6	74.0	4.5		

```
3            80.0            15.0
4           218.0            15.0
```

[10]:
```
df_num = df[numerical]
df_num
```

[10]:
```
      hotel_id  distance  stars  rating  rating_reviewcount  ratingta  \
0       9797.0      14.0    NaN     4.1                49.0       3.5
1       9798.0      10.0    4.0     4.1               113.0       4.0
2       9799.0      12.0    3.0     4.3                 8.0       3.5
3       9800.0      13.0    3.0     3.5                22.0       3.5
4       9801.0      13.0    4.0     4.6                74.0       4.5
…          …         …      …       …                   …         …
1093   10890.0      12.0    4.0     4.3                85.0       3.5
1094   10891.0      12.0    NaN     3.5                24.0       3.5
1095   10892.0      12.0    3.0     3.5                15.0       3.5
1096   10893.0       6.2    3.0     3.7                37.0       4.0
1097   10894.0      26.0    4.0     3.9               228.0       3.5

      ratingta_count  distance_alter
0               90.0            12.0
1              910.0            12.0
2              116.0            14.0
3               80.0            15.0
4              218.0            15.0
…                …               …
1093           305.0            14.0
1094            31.0            14.0
1095            82.0            14.0
1096           182.0             8.0
1097           347.0            26.0

[1098 rows x 8 columns]
```

[11]:
```
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1098 entries, 0 to 1097
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   hotel_id            1098 non-null   float64
 1   distance            1098 non-null   float64
 2   stars               541 non-null    float64
 3   rating              871 non-null    float64
 4   rating_reviewcount  871 non-null    float64
 5   ratingta            722 non-null    float64
```

```
 6   ratingta_count      722 non-null    float64
 7   distance_alter      1098 non-null   float64
dtypes: float64(8)
memory usage: 68.8 KB
```

3.2 Question 3: Report missing values in numarical variables and explain how to handle missing values in numerical variables.

[12]:
```
# See missing values using isnull
print(df_num.isnull().sum())
```

```
hotel_id               0
distance               0
stars                557
rating               227
rating_reviewcount   227
ratingta             376
ratingta_count       376
distance_alter         0
dtype: int64
```

[13]:
```
# Impute missing values using mean
df_num_imputed = df_num.fillna(df_num.mean())
df_num_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1098 entries, 0 to 1097
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   hotel_id            1098 non-null   float64
 1   distance            1098 non-null   float64
 2   stars               1098 non-null   float64
 3   rating              1098 non-null   float64
 4   rating_reviewcount  1098 non-null   float64
 5   ratingta            1098 non-null   float64
 6   ratingta_count      1098 non-null   float64
 7   distance_alter      1098 non-null   float64
dtypes: float64(8)
memory usage: 68.8 KB
```

[14]:
```
# Alternatively drop missing values in df_num
df_num_dropped = df_num.dropna()
df_num_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536 entries, 1 to 1097
Data columns (total 8 columns):
```

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   hotel_id            536 non-null    float64
 1   distance            536 non-null    float64
 2   stars               536 non-null    float64
 3   rating              536 non-null    float64
 4   rating_reviewcount  536 non-null    float64
 5   ratingta            536 non-null    float64
 6   ratingta_count      536 non-null    float64
 7   distance_alter      536 non-null    float64
dtypes: float64(8)
memory usage: 37.7 KB
```

[15]:
```
df_num_imputed.describe()
```

[15]:

	hotel_id	distance	stars	rating
count	1098.000000	1098.000000	1098.000000	1098.000000
mean	10345.500000	4.002095	3.395564	3.849369
std	317.109603	5.662733	0.629602	0.634641
min	9797.000000	0.000000	1.000000	1.000000
25%	10071.250000	1.100000	3.395564	3.700000
50%	10345.500000	1.700000	3.395564	3.849369
75%	10619.750000	3.800000	4.000000	4.100000
max	10894.000000	30.000000	5.000000	5.000000

	rating_reviewcount	ratingta	ratingta_count	distance_alter
count	1098.000000	1098.000000	1098.000000	1098.000000
mean	100.878301	3.772853	358.005540	5.024772
std	146.443586	0.556103	435.504418	5.520723
min	1.000000	1.000000	1.000000	0.200000
25%	14.250000	3.500000	106.000000	2.000000
50%	78.000000	3.772853	358.005540	3.100000
75%	100.878301	4.000000	358.005540	4.775000
max	1445.000000	5.000000	6697.000000	31.000000

[16]:
```
df_num_dropped.describe()
```

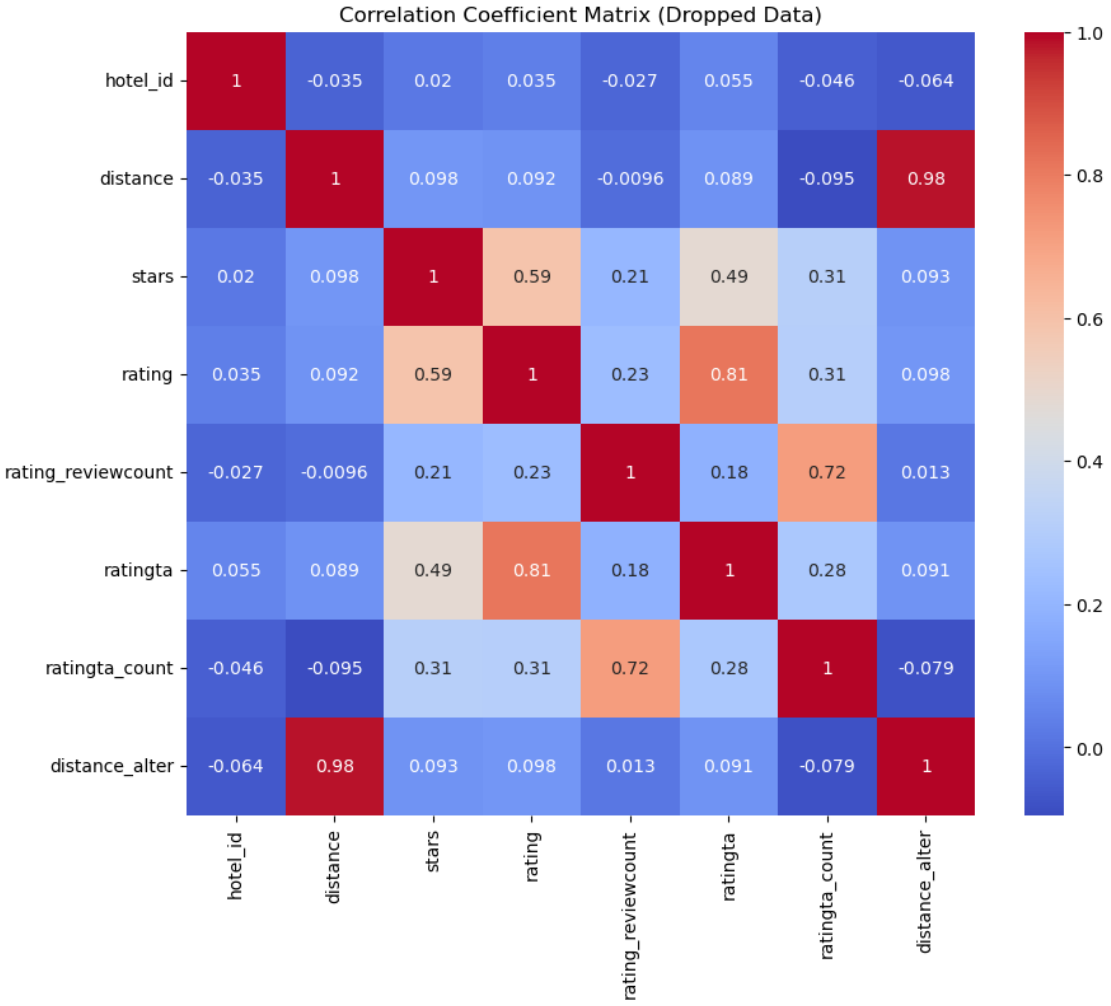
[16]:

	hotel_id	distance	stars	rating	rating_reviewcount
count	536.000000	536.000000	536.000000	536.000000	536.000000
mean	10317.904851	5.286007	3.393657	3.838433	151.126866
std	340.080178	6.540571	0.900092	0.532018	191.383011
min	9798.000000	0.100000	1.000000	1.500000	1.000000
25%	10018.250000	1.300000	3.000000	3.500000	39.000000
50%	10289.000000	2.100000	4.000000	4.000000	94.000000
75%	10649.250000	6.500000	4.000000	4.100000	187.750000
max	10894.000000	30.000000	5.000000	5.000000	1445.000000

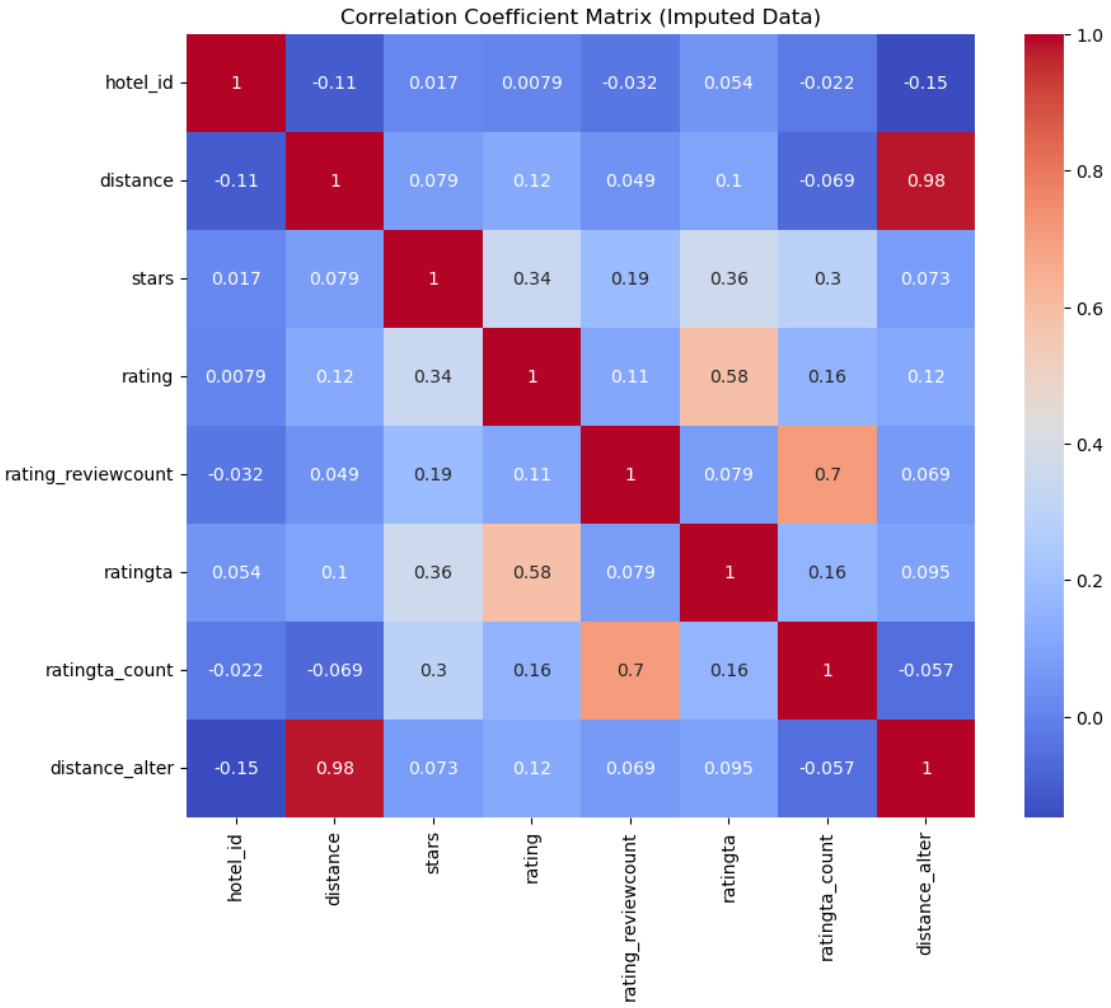
	ratingta	ratingta_count	distance_alter
count	536.000000	536.000000	536.000000
mean	3.720149	455.486940	6.342164
std	0.593615	589.862956	6.311776
min	1.500000	5.000000	0.300000
25%	3.500000	115.750000	2.600000
50%	4.000000	255.000000	3.600000
75%	4.000000	576.000000	7.600000
max	5.000000	6697.000000	31.000000

3.3 Question 4: Using either imputed or dropped data, report correlation matrix and and explain why you should drop some variable(s) from data.

[18]:
```
# Correlation in numerial data
correlation_matrix_dropped = df_num_dropped.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_dropped, annot=True, cmap="coolwarm")

# Set plot title
plt.title('Correlation Coefficient Matrix (Dropped Data)')

# Display the plot
plt.show()
```



[19]:
```
# Correlation in numerial data for imputed data
correlation_matrix_imputed = df_num_imputed.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_imputed, annot=True, cmap="coolwarm")

# Set plot title
plt.title('Correlation Coefficient Matrix (Imputed Data)')

# Display the plot
plt.show()
```



3.4 Question 5: Report subcategories in numerical variables and explain the number of subcategories of variable ratingta_count in terms of variation in this variable.

[20]:
```
# Checking subcategories for numerical variables to see if some integer valued␣
↪variables are actually categorical
for column in df_num:
    num_categories = df_num[column].nunique()
    print(f"Number of categories in {column}: {num_categories}")
```

```
Number of categories in hotel_id: 1098
Number of categories in distance: 112
Number of categories in stars: 6
Number of categories in rating: 25
Number of categories in rating_reviewcount: 275
Number of categories in ratingta: 9
```

```
Number of categories in ratingta_count: 448
Number of categories in distance_alter: 117
```

[21]:
```
df_num_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536 entries, 1 to 1097
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   hotel_id            536 non-null    float64
 1   distance            536 non-null    float64
 2   stars               536 non-null    float64
 3   rating              536 non-null    float64
 4   rating_reviewcount  536 non-null    float64
 5   ratingta            536 non-null    float64
 6   ratingta_count      536 non-null    float64
 7   distance_alter      536 non-null    float64
dtypes: float64(8)
memory usage: 37.7 KB
```

4 3. Outliers in numerical variables

4.1 Question 6: Create subplots for histograms and box plots and report the variable with the worst extreme outliers and explain why.

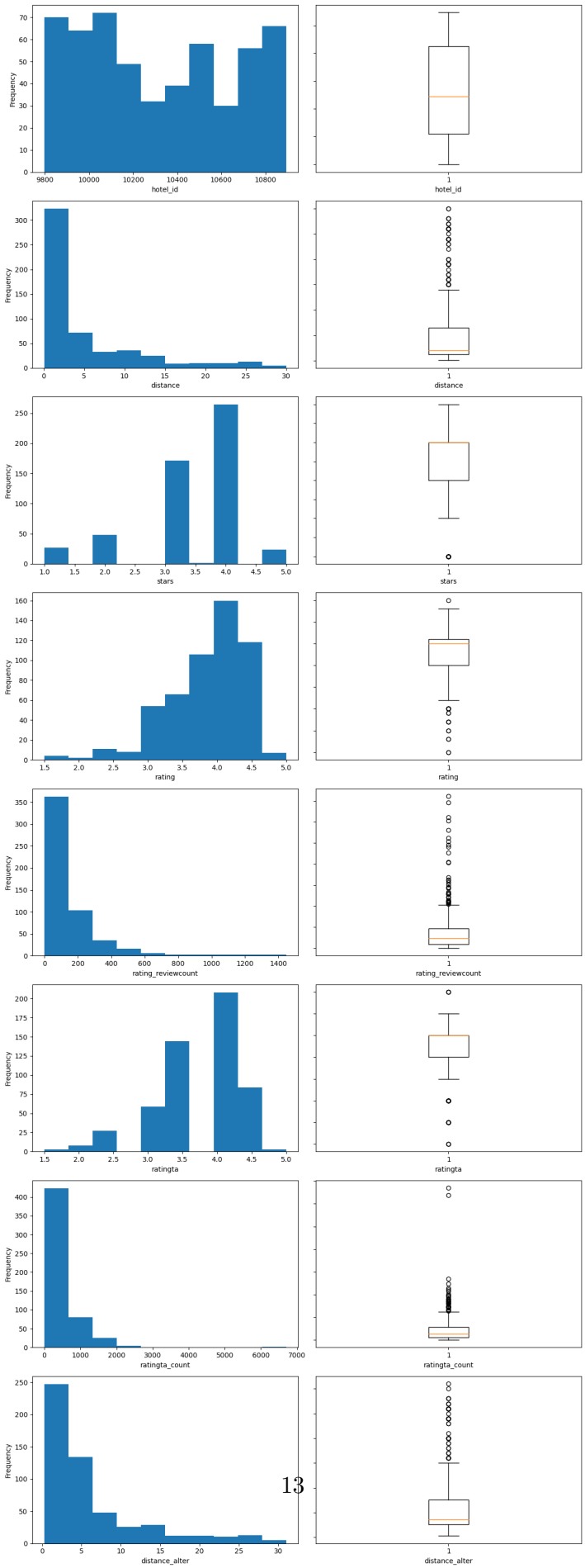
[23]:
```
df_num_dropped.columns
```

[23]:
```
Index(['hotel_id', 'distance', 'stars', 'rating', 'rating_reviewcount',
       'ratingta', 'ratingta_count', 'distance_alter'],
      dtype='object')
```

[24]:
```
# Define the numerical variables
numerical_vars = df_num_dropped.columns
```

[25]:
```
# Create subplots for histograms and box plots
fig, axs = plt.subplots(len(numerical_vars), 2, figsize=(12, 4 *␣
 ↪len(numerical_vars)))

# Plot histograms and box plots for each numerical variable
for i, var in enumerate(numerical_vars):
    # Plot histogram
    axs[i, 0].hist(df_num_dropped[var])
    axs[i, 0].set_xlabel(var)
    axs[i, 0].set_ylabel('Frequency')

    # Plot box plot
```

```
    axs[i, 1].boxplot(df_num_dropped[var])
    axs[i, 1].set_xlabel(var)
    axs[i, 1].set_yticklabels([])

# Adjust spacing between subplots
plt.tight_layout()

# Display the plot
plt.show()
```


Frequency
hotel_id
distance
stars
rating
rating_reviewcount
ratingta
ratingta_count
distance_alter

4.2 Question 7: After dropping missing values and scaling data, report scaled_df and check indexing for the row numbers and total observations. If there is something wrong report it and explain how to handle it.

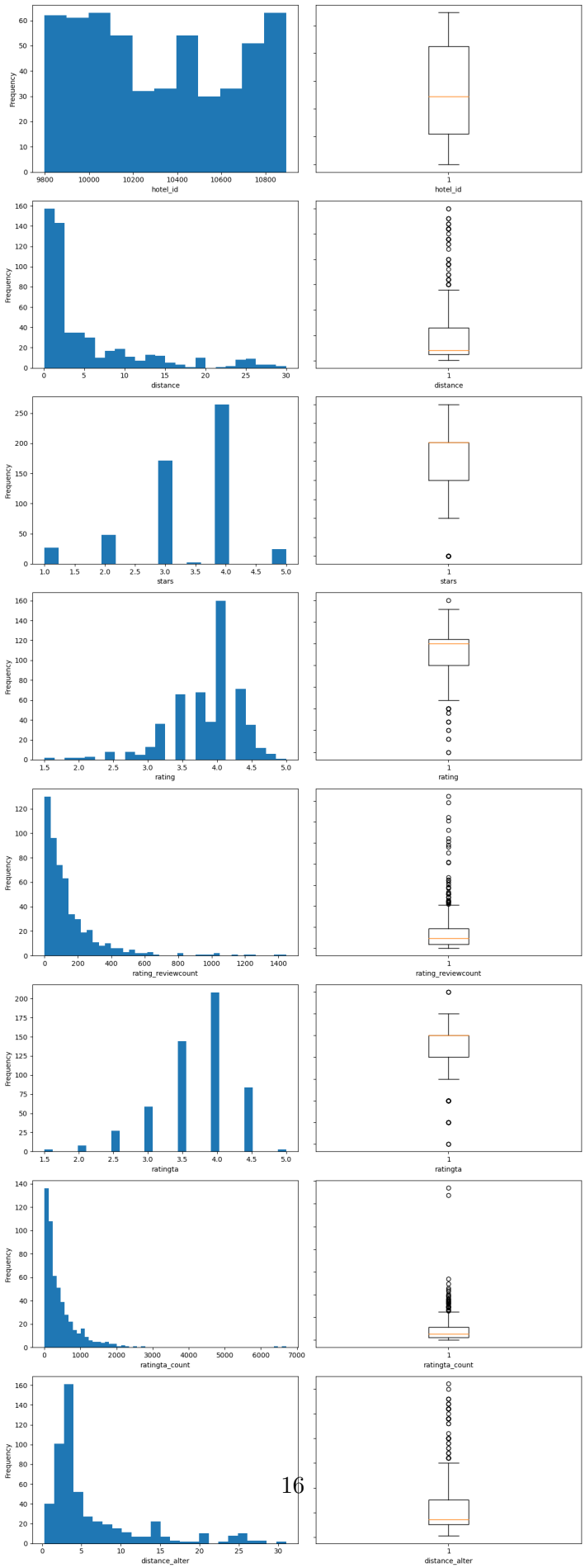
[26]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# Apply Winsorization to reduce outliers
winsorized_df = df_num_dropped.copy()
for var in numerical_vars:
    q_low = df_num_dropped[var].quantile(0.01)
    q_high = df_num_dropped[var].quantile(0.99)
    winsorized_df[var] = df_num_dropped[var].clip(q_low, q_high)

# Apply scaling to the winsorized DataFrame
scaler = MinMaxScaler()  # or StandardScaler()
scaled_df = pd.DataFrame(scaler.fit_transform(winsorized_df),␣
 ↪columns=numerical_vars)

# Display the scaled DataFrame
print(scaled_df)
```

```
     hotel_id  distance  stars   rating  rating_reviewcount  ratingta  \
0         0.0  0.363296   0.75  0.74359            0.107506       0.8
1         0.0  0.438202   0.50  0.81685            0.005811       0.6
2         0.0  0.475655   0.50  0.52381            0.019370       0.6
3         0.0  0.475655   0.75  0.92674            0.069734       1.0
4         0.0  0.183521   0.75  0.81685            0.173366       1.0
..        …       …    …       …                 …       …
531       1.0  0.513109   0.75  0.89011            0.085230       1.0
532       1.0  0.438202   0.75  0.81685            0.080387       0.6
533       1.0  0.438202   0.50  0.52381            0.012591       0.6
534       1.0  0.220974   0.50  0.59707            0.033898       0.8
535       1.0  0.962547   0.75  0.67033            0.218886       0.6

     ratingta_count  distance_alter
0          0.409850        0.433214
1          0.046779        0.508785
2          0.030317        0.546571
3          0.093420        0.546571
4          0.419452        0.093142
..              …               …
531        0.030317        0.584357
532        0.133202        0.508785
533        0.031231        0.508785
534        0.076958        0.282071
535        0.152408        0.962214
```

```
[536 rows x 8 columns]
```

[27]:
```
# To see if there is difference in distribution after droping outliers we will␣
 ↪drop outliers remained out of the first and third quantiles
Q1 = scaled_df.quantile(0.25)
Q3 = scaled_df.quantile(0.75)
IQR = Q3 - Q1
scaled_df = scaled_df[~((scaled_df < (Q1 - 1.5 * IQR)) | (scaled_df > (Q3 + 1.5␣
 ↪* IQR))).any(axis=1)]
```

[28]:
```
# Create subplots for histograms and box plots
fig, axs = plt.subplots(len(numerical_vars), 2, figsize=(12, 4 *␣
 ↪len(numerical_vars)))

# Plot histograms and box plots for each numerical variable
for i, var in enumerate(numerical_vars):
    # Plot histogram
    axs[i, 0].hist(df_num_dropped[var], bins='auto')
    axs[i, 0].set_xlabel(var)
    axs[i, 0].set_ylabel('Frequency')

    # Plot box plot
    axs[i, 1].boxplot(df_num_dropped[var])
    axs[i, 1].set_xlabel(var)
    axs[i, 1].set_yticklabels([])

# Adjust spacing between subplots
plt.tight_layout()

# Display the plot
plt.show()
```


Frequency
hotel_id
distance
stars
rating
rating_reviewcount
ratingta
ratingta_count
distance_alter

[29]:
```
scaled_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 378 entries, 0 to 534
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   hotel_id            378 non-null    float64
 1   distance            378 non-null    float64
 2   stars               378 non-null    float64
 3   rating              378 non-null    float64
 4   rating_reviewcount  378 non-null    float64
 5   ratingta            378 non-null    float64
 6   ratingta_count      378 non-null    float64
 7   distance_alter      378 non-null    float64
dtypes: float64(8)
memory usage: 26.6 KB
```

We have better distribution and less outliers but with serious cost!

[30]:
```
scaled_df
```

[30]:

	hotel_id	distance	stars	rating	rating_reviewcount	ratingta	\
0	0.000000	0.363296	0.75	0.74359	0.107506	0.8	
1	0.000000	0.438202	0.50	0.81685	0.005811	0.6	
2	0.000000	0.475655	0.50	0.52381	0.019370	0.6	
3	0.000000	0.475655	0.75	0.92674	0.069734	1.0	
4	0.000000	0.183521	0.75	0.81685	0.173366	1.0	
..	…	…	…	…	…	…	
528	0.996072	0.089888	0.75	0.70696	0.134625	0.8	
529	0.996996	0.086142	0.75	0.70696	0.113317	0.6	
532	1.000000	0.438202	0.75	0.81685	0.080387	0.6	
533	1.000000	0.438202	0.50	0.52381	0.012591	0.6	
534	1.000000	0.220974	0.50	0.59707	0.033898	0.8	

	ratingta_count	distance_alter
0	0.409850	0.433214
1	0.046779	0.508785
2	0.030317	0.546571
3	0.093420	0.546571
4	0.419452	0.093142
..	…	…
528	0.348576	0.164935
529	0.187160	0.161156
532	0.133202	0.508785
533	0.031231	0.508785

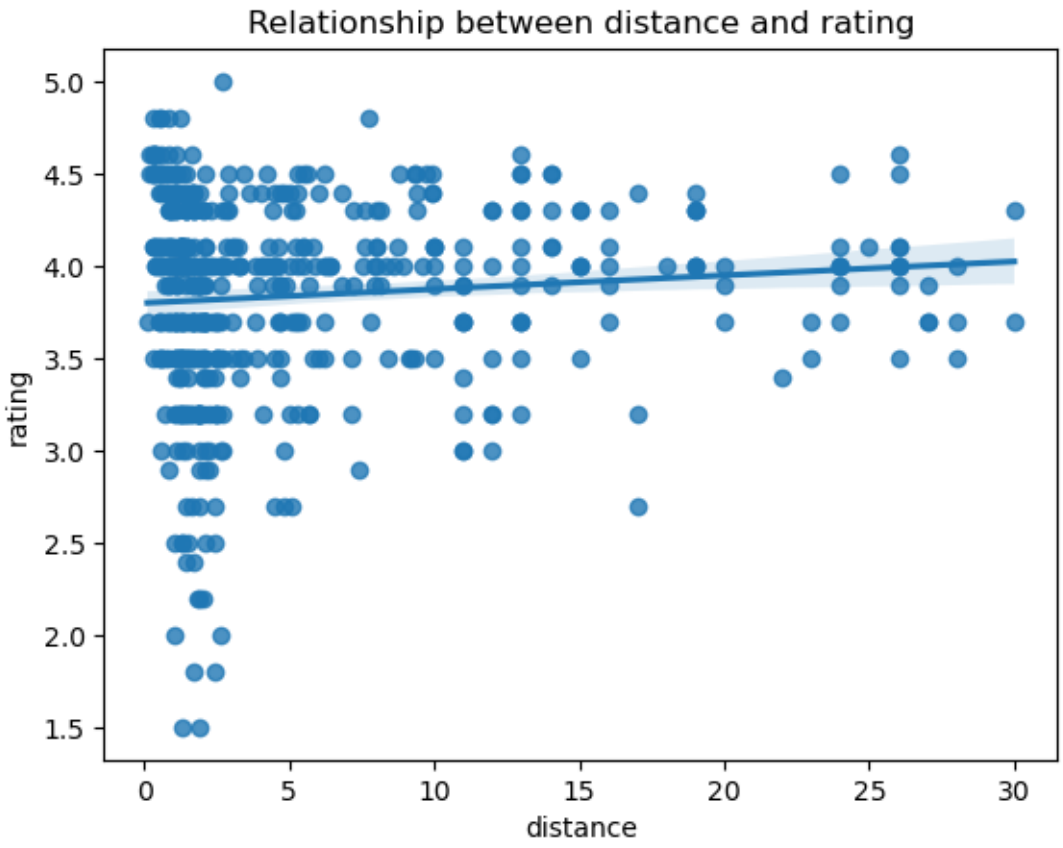
```
534        0.076958        0.282071

[378 rows x 8 columns]
```

[31]:
```
# To drop index and create new one
scaled_df = scaled_df.reset_index(drop=True)
scaled_df
```

[31]:
```
     hotel_id  distance  stars   rating  rating_reviewcount  ratingta  \
0    0.000000  0.363296   0.75  0.74359            0.107506       0.8
1    0.000000  0.438202   0.50  0.81685            0.005811       0.6
2    0.000000  0.475655   0.50  0.52381            0.019370       0.6
3    0.000000  0.475655   0.75  0.92674            0.069734       1.0
4    0.000000  0.183521   0.75  0.81685            0.173366       1.0
..        …       …     …       …                   …       …
373  0.996072  0.089888   0.75  0.70696            0.134625       0.8
374  0.996996  0.086142   0.75  0.70696            0.113317       0.6
375  1.000000  0.438202   0.75  0.81685            0.080387       0.6
376  1.000000  0.438202   0.50  0.52381            0.012591       0.6
377  1.000000  0.220974   0.50  0.59707            0.033898       0.8

     ratingta_count  distance_alter
0          0.409850        0.433214
1          0.046779        0.508785
2          0.030317        0.546571
3          0.093420        0.546571
4          0.419452        0.093142
..              …               …
373        0.348576        0.164935
374        0.187160        0.161156
375        0.133202        0.508785
376        0.031231        0.508785
377        0.076958        0.282071

[378 rows x 8 columns]
```

4.3 Question 8: Explaing the relationship between distance and rating in data using scatter plot and sns function with trend line.

[32]:
```
# Plotting the relationship between rating and distance
sns.regplot(x='distance', y='rating', data=df_num_dropped)
plt.title('Relationship between distance and rating')
plt.show()
```



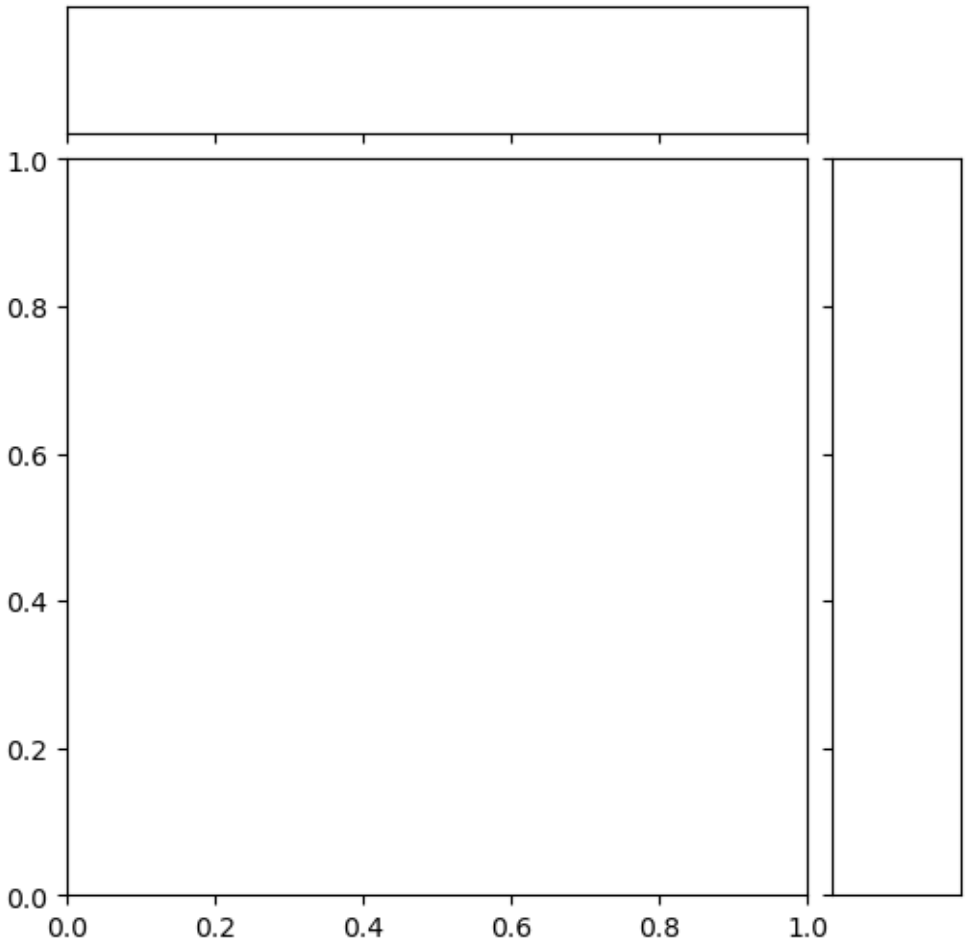
There is always better when it comes to fancy visualization

[33]:
```
# Create a joint plot for price and distance
sns.jointplot(x='distance', y='price', data=df_num_dropped)
plt.title('Relationship between rating and distance')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[33], line 2
      1 # Create a joint plot for price and distance
----> 2 sns.jointplot(x='distance', y='price', data=df_num_dropped)
      3 plt.title('Relationship between rating and distance')
      4 plt.show()

File /opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:46, in␣
 ↪_deprecate_positional_args.<locals>.inner_f(*args, **kwargs)
     36     warnings.warn(
     37         "Pass the following variable{} as {}keyword arg{}: {}. "
     38         "From version 0.12, the only valid positional argument "
```

```
   (…)
     43         FutureWarning
     44     )
     45 kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46 return f(**kwargs)

File /opt/conda/lib/python3.9/site-packages/seaborn/axisgrid.py:2230, in␣
 ↪jointplot(x, y, data, kind, color, height, ratio, space, dropna, xlim, ylim,␣
 ↪marginal_ticks, joint_kws, marginal_kws, hue, palette, hue_order, hue_norm,␣
 ↪**kwargs)
   2227     dropna = True
   2229 # Initialize the JointGrid object
-> 2230 grid = JointGrid(
   2231     data=data, x=x, y=y, hue=hue,
   2232     palette=palette, hue_order=hue_order, hue_norm=hue_norm,
   2233     dropna=dropna, height=height, ratio=ratio, space=space,
   2234     xlim=xlim, ylim=ylim, marginal_ticks=marginal_ticks,
   2235 )
   2237 if grid.hue is not None:
   2238     marginal_kws.setdefault("legend", False)

File /opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:46, in␣
 ↪_deprecate_positional_args.<locals>.inner_f(*args, **kwargs)
     36     warnings.warn(
     37         "Pass the following variable{} as {}keyword arg{}: {}. "
     38         "From version 0.12, the only valid positional argument "
   (…)
     43         FutureWarning
     44     )
     45 kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46 return f(**kwargs)

File /opt/conda/lib/python3.9/site-packages/seaborn/axisgrid.py:1702, in␣
 ↪JointGrid.__init__(self, x, y, data, height, ratio, space, dropna, xlim, ylim␣
 ↪size, marginal_ticks, hue, palette, hue_order, hue_norm)
   1699     ax_marg_y.xaxis.grid(False)
   1701 # Process the input variables
-> 1702 p = VectorPlotter(data=data, variables=dict(x=x, y=y, hue=hue))
   1703 plot_data = p.plot_data.loc[:, p.plot_data.notna().any()]
   1705 # Possibly drop NA

File /opt/conda/lib/python3.9/site-packages/seaborn/_core.py:605, in␣
 ↪VectorPlotter.__init__(self, data, variables)
    603 def __init__(self, data=None, variables={}):
--> 605     self.assign_variables(data, variables)
    607     for var, cls in self._semantic_mappings.items():
    608
    609         # Create the mapping function
```

```
    610         map_func = partial(cls.map, plotter=self)

File /opt/conda/lib/python3.9/site-packages/seaborn/_core.py:668, in␣
 ↪VectorPlotter.assign_variables(self, data, variables)
    666 else:
    667     self.input_format = "long"
--> 668     plot_data, variables = self._assign_variables_longform(
    669         data, **variables,
    670     )
    672 self.plot_data = plot_data
    673 self.variables = variables

File /opt/conda/lib/python3.9/site-packages/seaborn/_core.py:903, in␣
 ↪VectorPlotter._assign_variables_longform(self, data, **kwargs)
    898 elif isinstance(val, (str, bytes)):
    899
    900     # This looks like a column name but we don't know what it means!
    902     err = f"Could not interpret value `{val}` for parameter `{key}`"
--> 903     raise ValueError(err)
    905 else:
    906
    907     # Otherwise, assume the value is itself data
    908
    909     # Raise when data object is present and a vector can't matched
    910     if isinstance(data, pd.DataFrame) and not isinstance(val, pd.Series):

ValueError: Could not interpret value `price` for parameter `y`
```



[ ]:
```
sns.pairplot(scaled_df)
```

[ ]:
```
sns.set(style="ticks", color_codes=True)
g = sns.pairplot(scaled_df, kind="reg", plot_kws={'line_kws':{'color':
  ↪'purple'}})
plt.show()
```