


Below are the dataset links and a question bank for practice. Focus on understanding each question, the code functions used to derive the answers, and their outputs in various scenarios.



```
1
2 # import pandas as pd
3
4 # listings = pd.read_csv("https://storage.googleapis.com/public-data-337819/listings%202%20reduced.csv", low_memory=False)
5 # reviews = pd.read_csv("https://storage.googleapis.com/public-data-337819/reviews%202%20reduced.csv", low_memory=False)
```

```
1 import pandas as pd
```

```
1 listings = pd.read_csv("listings 2 reduced.csv")
2 reviews = pd.read_csv("reviews 2 reduced.csv")
```


 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1565045993.py:1: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv("listings 2 reduced.csv")

```
1 reviews.head()
```

	listing_id	id	date	reviewer_id	reviewer_name	comments
0	13913	80770	2010-08-18	177109	Michael	My girlfriend and I hadn't known Alina before ...
1	13913	367568	2011-07-11	19835707	Mathias	Alina was a really good host. The flat is clea...
2	13913	529579	2011-09-13	1110304	Kristin	Alina is an amazing host. She made me feel rig...
3	13913	595481	2011-10-03	1216358	Camilla	Alina's place is so nice, the room is big and ...
4	13913	612947	2011-10-09	490840	Jorik	Nice location in Islington area, good for shor...

```
1 reviews.info()
```



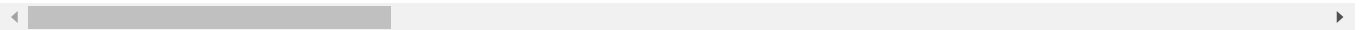
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1581033 entries, 0 to 1581032
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   listing_id      1581033 non-null  int64
1   id              1581033 non-null  int64
2   date            1581033 non-null  object
3   reviewer_id     1581033 non-null  int64
4   reviewer_name   1581033 non-null  object
5   comments        1580874 non-null  object
dtypes: int64(3), object(3)
memory usage: 72.4+ MB
```

```
1 listings.head()
```



	id	listing_url	scrape_id	last_scraped	source	name	description	neighborhood_overview	
0	92644	https://www.airbnb.com/rooms/92644	20230906022807	2023-09-06	city scrape	Rental unit in Earlsfield · ★4.57 · 1 bedroom ...	The space Hi everyone! I have 2 ro...	NaN	http
1	93015	https://www.airbnb.com/rooms/93015	20230906022807	2023-09-06	city scrape	Rental unit in Hammersmith · ★4.82 · 2 bedroom...	Gorgeous 2 bed ground floor apartment with per...	A bit of history about the W14 area: Com...	http
2	13913	https://www.airbnb.com/rooms/13913	20230906022807	2023-09-06	city scrape	Rental unit in Islington · ★4.80 · 1 bedroom · ...	My bright double bedroom with a large window h...	Finsbury Park is a friendly melting pot commun...	h
3	15400	https://www.airbnb.com/rooms/15400	20230906022807	2023-09-06	city scrape	Rental unit in London · ★4.80 · 1 bedroom · 1 ...	Lots of windows and light. St Luke's Gardens ...	It is Chelsea.	http
4	93734	https://www.airbnb.com/rooms/93734	20230906022807	2023-09-07	city scrape	Condo in London · ★4.62 · 1 bedroom · 1 bed · ...	During your stay I'm a professiona...	NaN	h

5 rows × 75 columns



```
1 listings.info()
```




```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 87946 entries, 0 to 87945
Data columns (total 75 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   id                                         87946 non-null  int64
1   listing_url                             87946 non-null  object
2   scrape_id                               87946 non-null  int64
3   last_scraped                            87946 non-null  object
4   source                                   87946 non-null  object
5   name                                      87946 non-null  object
6   description                              86679 non-null  object
7   neighborhood_overview                   47190 non-null  object
8   picture_url                             87943 non-null  object
9   host_id                                 87946 non-null  int64
10  host_url                                 87946 non-null  object
11  host_name                               87941 non-null  object
12  host_since                              87941 non-null  object
13  host_location                           69168 non-null  object
14  host_about                              45906 non-null  object
15  host_response_time                      59028 non-null  object
16  host_response_rate                      59028 non-null  object
17  host_acceptance_rate                    62758 non-null  object
18  host_is_superhost                       87014 non-null  object
19  host_thumbnail_url                      87941 non-null  object
20  host_picture_url                        87941 non-null  object
21  host_neighbourhood                      48079 non-null  object
22  host_listings_count                     87941 non-null  float64
23  host_total_listings_count               87941 non-null  float64
24  host_verifications                      87941 non-null  object
25  host_has_profile_pic                    87941 non-null  object
26  host_identity_verified                  87941 non-null  object
27  neighbourhood                           47191 non-null  object
28  neighbourhood_cleansed                  87946 non-null  object
29  neighbourhood_group_cleansed            0 non-null     float64
30  latitude                                87946 non-null  float64
31  longitude                                87946 non-null  float64
32  property_type                           87946 non-null  object
33  room_type                               87946 non-null  object
34  accommodates                            87946 non-null  int64
35  bathrooms                              0 non-null     float64
36  bathrooms_text                          87843 non-null  object
37  bedrooms                               55172 non-null  float64
38  beds                                   86812 non-null  float64
39  amenities                               87946 non-null  object
40  price                                   87946 non-null  object
41  minimum_nights                          87946 non-null  int64
42  maximum_nights                          87946 non-null  int64
43  minimum_minimum_nights                  87945 non-null  float64
44  maximum_minimum_nights                  87945 non-null  float64
45  minimum_maximum_nights                  87945 non-null  float64
46  maximum_maximum_nights                  87945 non-null  float64
```

47	minimum_nights_avg_ntm	87945	non-null	float64
48	maximum_nights_avg_ntm	87945	non-null	float64
49	calendar_updated	0	non-null	float64
50	has_availability	87946	non-null	object
51	availability_30	87946	non-null	int64
52	availability_60	87946	non-null	int64

1. What is the neighborhood in which superhosts have the biggest median price difference with respect to non-superhosts?

Use the `host_is_superhost`, `neighbourhood_cleansed`, and `price` columns.

```
1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace('\$', '', regex=True).astype(float)
8
9 # Convert the 'host_is_superhost' column to boolean
10 # The 'apply' method is used to convert 't' to True and 'f' to False
11 listings['host_is_superhost'] = listings['host_is_superhost'].apply(lambda x: True if x == 't' else False)
12
13 # Calculate the median price for superhosts and non-superhosts in each neighborhood
14 # Group the listings by 'neighbourhood_cleansed' and 'host_is_superhost'
15 # Calculate the median of the 'price' column for each group
16 # 'unstack' reshapes the DataFrame so that 'host_is_superhost' values become columns
17 median_prices = listings.groupby(['neighbourhood_cleansed', 'host_is_superhost'])['price'].median().unstack()
18
19 # Calculate the median price difference between superhosts and non-superhosts
20 # Create a new column 'difference' by subtracting the median price of non-superhosts from superhosts
21 median_prices['difference'] = median_prices[True] - median_prices[False]
22
23 # Find the neighborhood with the biggest median price difference
24 # 'idxmax' finds the index (neighborhood) with the maximum value in the 'difference' column
25 # 'max' returns the maximum value of the 'difference' column
26 biggest_difference_neighborhood = median_prices['difference'].idxmax()
27 biggest_difference_value = median_prices['difference'].max()
28
29 # Print the result
30 # Formatted string to display the neighborhood and the median price difference
31 print(f'The neighborhood with the biggest median price difference is {biggest_difference_neighborhood} with a difference of ${biggest_difference_value}')
32
```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\2787462854.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype before reading. (listings = pd.read_csv('listings 2 reduced.csv'))

The neighborhood with the biggest median price difference is Westminster with a difference of \$48.50.

2. What is the average price difference between a professional host and a non-professional one?


A professional host is defined as having listings in more than 5 different locations (`neighbourhood_cleansed`).

```
1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace('\$', '', regex=True).astype(float)
8
9 # Identify professional hosts
10 # Count the number of unique neighborhoods each host has listings in
11 # 'groupby' groups the data by 'host_id', and 'nunique' counts unique 'neighbourhood_cleansed'
12 host_neighbourhood_count = listings.groupby('host_id')['neighbourhood_cleansed'].nunique()
13
14 # Create a new column in the listings DataFrame to indicate if the host is professional
15 # 'apply' applies a function to each element in 'host_id' to check if the count is greater than 5
16 listings['is_professional_host'] = listings['host_id'].apply(lambda x: host_neighbourhood_count[x] > 5)
17
18 # Calculate the average price for professional and non-professional hosts
19 # 'groupby' groups the data by 'is_professional_host', and 'mean' calculates the average price
20 average_prices = listings.groupby('is_professional_host')['price'].mean()
21
22 # Calculate the price difference
23 # Subtract the average price of non-professional hosts from professional hosts
24 price_difference = average_prices[True] - average_prices[False]
25
```

```

26 # Print the result
27 # Formatted string to display the average price difference with two decimal places
28 print(f'The average price difference between professional hosts and non-professional hosts is ${price_difference:.2f}.')
29

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1386037801.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average price difference between professional hosts and non-professional hosts is \$285.71.


3. What is the median price premium given to entire homes/apartments with respect to other listings in the same neighborhood?

Use the `room_type` column to distinguish listing types and compute the average across all neighborhoods.

```

1 import pandas as pd
2
3
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'price' column to a numeric type after removing the currency symbol
7 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
8 listings['price'] = listings['price'].replace(['$', ''], regex=True).astype(float)
9
10 # Separate the listings into entire homes/apartments and other listings
11 # Filter the DataFrame to create two new DataFrames based on the 'room_type' column
12 entire_homes = listings[listings['room_type'] == 'Entire home/apt']
13 other_listings = listings[listings['room_type'] != 'Entire home/apt']
14
15 # Calculate the median price for entire homes/apartments within each neighborhood
16 # 'groupby' groups the data by 'neighbourhood_cleansed', and 'median' calculates the median price
17 median_price_entire_homes = entire_homes.groupby('neighbourhood_cleansed')['price'].median()
18
19 # Calculate the median price for other listings within each neighborhood
20 median_price_other_listings = other_listings.groupby('neighbourhood_cleansed')['price'].median()
21
22 # Calculate the median price premium for entire homes/apartments in each neighborhood
23 # Align the indices of both series and calculate the difference
24 price_premiums = median_price_entire_homes.align(median_price_other_listings, join='inner', fill_value=0)
25 price_premiums = price_premiums[0] - price_premiums[1]
26
27 # Compute the overall average price premium across all neighborhoods
28 average_price_premium = price_premiums.mean()
29
30 # Print the result
31 # Formatted string to display the average price premium with two decimal places
32 print(f'The average price premium for entire homes/apartments compared to other listings is ${average_price_premium:.2f}.')
33

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\955772620.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average price premium for entire homes/apartments compared to other listings is \$86.79.

4. Which neighborhood has the highest density of listings per square kilometer?

Use the `latitude`, `longitude`, and `neighbourhood_cleansed` columns.

```


1 import pandas as pd
2 from geopy.distance import geodesic
3
4 #This line imports the pandas library for data manipulation and
5 # the geodesic function from the geopy library for calculating distances based on latitude and longitude coordinates.
6
7 listings = pd.read_csv('listings 2 reduced.csv')
8
9 # Define a function to approximate the area of a neighborhood using latitude and longitude coordinates
10 def calculate_area(coordinates):
11     # If there are fewer than 3 coordinates, return an area of 0.0
12     if len(coordinates) < 3:
13         return 0.0
14
15     # Initialize the total area to 0.0
16     total_area = 0.0
17
18     # Loop through the coordinates to calculate the perimeter of the neighborhood
19     for i in range(len(coordinates) - 1):

```

```

20     total_area += geodesic(coordinates[i], coordinates[i + 1]).kilometers
21
22     # Close the polygon by adding the distance between the last and the first coordinate
23     total_area += geodesic(coordinates[-1], coordinates[0]).kilometers
24
25     return total_area
26
27 # Group the listings by neighborhood and calculate the bounding box for each neighborhood
28 # 'apply' method is used to create a list of tuples (latitude, longitude) for each neighborhood
29 neighborhood_coordinates = listings.groupby('neighbourhood_cleansed')[['latitude', 'longitude']].apply(lambda x: list(zip(x['latitude'], x['longitude'])))
30
31 # Calculate the approximate area for each neighborhood using the 'calculate_area' function
32 neighborhood_areas = neighborhood_coordinates.apply(calculate_area)
33
34 # Count the number of listings in each neighborhood using 'value_counts'
35 neighborhood_listing_counts = listings['neighbourhood_cleansed'].value_counts()
36
37 # Calculate the density of listings for each neighborhood (listings per square kilometer)
38 # Divide the number of listings by the area of the neighborhood
39 neighborhood_densities = neighborhood_listing_counts / neighborhood_areas
40
41 # Find the neighborhood with the highest density of listings
42 # 'idxmax' returns the index (neighborhood) of the maximum value in 'neighborhood_densities'
43 # 'max' returns the maximum value of the 'neighborhood_densities' series
44 highest_density_neighborhood = neighborhood_densities.idxmax()
45 highest_density_value = neighborhood_densities.max()
46
47 # Print the result
48 # Formatted string to display the neighborhood and the density with two decimal places
49 print(f'The neighborhood with the highest density of listings is {highest_density_neighborhood} with a density of {highest_density_value} listings per square kilometer.')
50

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1000850131.py:6: DtypeWarning: Columns (68) have mixed types. Specify dtype option on import or set low_resolution=False or high_resolution=True.
 listings = pd.read_csv('listings 2 reduced.csv')
 The neighborhood with the highest density of listings is City of London with a density of 1.16 listings per square kilometer.


5. What is the average price per guest for listings in each room type category?

Use the `price` and `accommodates` columns.

```

1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace(['$', ''], regex=True).astype(float)
8
9 # Calculate the price per guest for each listing
10 # This is done by dividing the 'price' column by the 'accommodates' column
11 listings['price_per_guest'] = listings['price'] / listings['accommodates']
12
13 # Calculate the average price per guest for each room type category
14 # 'groupby' groups the data by 'room_type', and 'mean' calculates the average price per guest
15 average_price_per_guest = listings.groupby('room_type')['price_per_guest'].mean()
16
17 # Print the result
18 # Formatted string to display the average price per guest for each room type with two decimal places
19 print("The average price per guest for each room type category:")
20 for room_type, avg_price in average_price_per_guest.items():
21     print(f'{room_type}: ${avg_price:.2f}')
22

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\686401307.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype option on import or set low_resolution=False or high_resolution=True.
 listings = pd.read_csv('listings 2 reduced.csv')
 The average price per guest for each room type category:
 Entire home/apt: \$65.27
 Hotel room: \$89.60
 Private room: \$61.69
 Shared room: \$68.18

6. What is the percentage of listings priced above the overall average price in each neighborhood?

Use the `price` and `neighbourhood_cleansed` columns.

```

1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace(['$', ' ', regex=True).astype(float)
8
9 # Calculate the overall average price of all listings
10 overall_average_price = listings['price'].mean()
11
12 # Identify listings that are priced above the overall average price
13 # Create a new column 'above_average' to indicate whether the listing price is above the overall average price
14 listings['above_average'] = listings['price'] > overall_average_price
15
16 # Calculate the percentage of listings above the overall average price for each neighborhood
17 # Group the listings by 'neighbourhood_cleansed' and calculate the percentage
18 percentage_above_average = listings.groupby('neighbourhood_cleansed')['above_average'].mean() * 100
19
20 # Print the result
21 # Formatted string to display the neighborhood and the percentage with two decimal places
22 print("The percentage of listings priced above the overall average price in each neighborhood:")
23 for neighborhood, percentage in percentage_above_average.items():
24     print(f'{neighborhood}: {percentage:.2f}%')
25

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\233635830.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype c
listings = pd.read_csv('listings 2 reduced.csv')

The percentage of listings priced above the overall average price in each neighborhood:

Barking and Dagenham: 14.72%
 Barnet: 19.79%
 Bexley: 7.27%
 Brent: 22.41%
 Bromley: 12.82%
 Camden: 35.02%
 City of London: 43.84%
 Croydon: 8.76%
 Ealing: 16.69%
 Enfield: 11.59%
 Greenwich: 17.90%
 Hackney: 18.07%
 Hammersmith and Fulham: 29.57%
 Haringey: 16.37%
 Harrow: 10.74%
 Havering: 13.45%
 Hillingdon: 9.62%
 Hounslow: 22.43%
 Islington: 22.50%
 Kensington and Chelsea: 53.82%
 Kingston upon Thames: 17.60%
 Lambeth: 19.88%
 Lewisham: 10.46%
 Merton: 24.02%
 Newham: 20.64%
 Redbridge: 14.38%
 Richmond upon Thames: 28.32%
 Southwark: 23.03%
 Sutton: 6.56%
 Tower Hamlets: 19.82%
 Waltham Forest: 12.89%
 Wandsworth: 26.77%
 Westminster: 54.12%

✓ 7. Which neighborhood has the highest average revenue per listing based on the last 12 months?

Use the `reviews_per_month`, `price`, and `minimum_nights` columns.

```


1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace(['$', ' ', regex=True).astype(float)
8
9 # Estimate the monthly revenue for each listing
10 # Monthly revenue is estimated as: reviews_per_month * price * minimum_nights
11 listings['monthly_revenue'] = listings['reviews_per_month'] * listings['price'] * listings['minimum_nights']
12
13 # Calculate the average monthly revenue per listing for each neighborhood
14 # 'groupby' groups the data by 'neighbourhood_cleansed', and 'mean' calculates the average monthly revenue
15 average_revenue_per_neighborhood = listings.groupby('neighbourhood_cleansed')['monthly_revenue'].mean()

```

```

16
17 # Find the neighborhood with the highest average monthly revenue per listing
18 # 'idxmax' returns the index (neighborhood) of the maximum value in 'average_revenue_per_neighborhood'
19 # 'max' returns the maximum value of the 'average_revenue_per_neighborhood' series
20 highest_revenue_neighborhood = average_revenue_per_neighborhood.idxmax()
21 highest_revenue_value = average_revenue_per_neighborhood.max()
22
23 # Print the result
24 # Formatted string to display the neighborhood and the average monthly revenue with two decimal places
25 print(f'The neighborhood with the highest average monthly revenue per listing is {highest_revenue_neighborhood} with an average revenue of {highest_revenue_value:.2f}')
26

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\172774232.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype option on import or as df.to_csv(..., dtype=...)
 listings = pd.read_csv('listings 2 reduced.csv')
 The neighborhood with the highest average monthly revenue per listing is Kensington and Chelsea with an average revenue of \$1190.05


8. What is the review score attribute that has the highest correlation with the price of listings?

Analyze `review_scores_rating`, `review_scores_cleanliness`, etc., against price.

```

1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'price' column to a numeric type after removing the currency symbol
6 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
7 listings['price'] = listings['price'].replace('$', '', regex=True).astype(float)
8
9 # Select the relevant columns for correlation analysis
10 # This includes the 'price' column and the review score columns
11 review_columns = [
12     'review_scores_rating', 'review_scores_accuracy', 'review_scores_cleanliness',
13     'review_scores_checkin', 'review_scores_communication', 'review_scores_location',
14     'review_scores_value'
15 ]
16 correlation_data = listings[['price'] + review_columns]
17
18 # Calculate the correlation matrix for the selected columns
19 correlation_matrix = correlation_data.corr()
20
21 # Extract the correlation values between 'price' and each review score attribute
22 # We use the 'price' row from the correlation matrix
23 price_correlations = correlation_matrix.loc['price', review_columns]
24
25 # Identify the review score attribute with the highest correlation with price
26 highest_correlation_attribute = price_correlations.idxmax()
27 highest_correlation_value = price_correlations.max()
28
29 # Print the result
30 # Formatted string to display the attribute and the correlation value with two decimal places
31 print(f'The review score attribute with the highest correlation with price is {highest_correlation_attribute} with a correlation of {highest_correlation_value:.2f}')
32

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\3182913975.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype option on import or as df.to_csv(..., dtype=...)
 listings = pd.read_csv('listings 2 reduced.csv')
 The review score attribute with the highest correlation with price is review_scores_location with a correlation of 0.03.

9. What is the average difference between review scores of superhosts vs. normal hosts?

Use the `review_scores_rating` column.

```


1 import pandas as pd
2
3
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'host_is_superhost' column to boolean
7 # The 'apply' method is used to convert 't' to True and 'f' to False
8 listings['host_is_superhost'] = listings['host_is_superhost'].apply(lambda x: True if x == 't' else False)
9
10 # Calculate the average review scores for superhosts and normal hosts
11 # Group the listings by 'host_is_superhost' and calculate the mean of 'review_scores_rating'
12 average_review_scores = listings.groupby('host_is_superhost')['review_scores_rating'].mean()

```

```

13
14 # Calculate the difference between the average review scores
15 difference = average_review_scores[True] - average_review_scores[False]
16
17 # Print the result
18 # Formatted string to display the average difference with two decimal places
19 print(f'The average difference between review scores of superhosts and normal hosts is {difference:.2f}.')
20

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\2694870490.py:4: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average difference between review scores of superhosts and normal hosts is 0.32.

10. Which neighborhood has the highest average review score for communication?

Use the `review_scores_communication` and `neighbourhood_cleansed` columns.

```

1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Calculate the average review score for communication for each neighborhood
6 # Group the listings by 'neighbourhood_cleansed' and calculate the mean of 'review_scores_communication'
7 average_review_scores_communication = listings.groupby('neighbourhood_cleansed')['review_scores_communication'].mean()
8
9 # Find the neighborhood with the highest average review score for communication
10 # 'idxmax' returns the index (neighborhood) of the maximum value in 'average_review_scores_communication'
11 # 'max' returns the maximum value of the 'average_review_scores_communication' series
12 highest_avg_communication_neighborhood = average_review_scores_communication.idxmax()
13 highest_avg_communication_score = average_review_scores_communication.max()
14
15 # Print the result
16 # Formatted string to display the neighborhood and the average review score with two decimal places
17 print(f'The neighborhood with the highest average review score for communication is {highest_avg_communication_neighborhood} with an
18

```

 The neighborhood with the highest average review score for communication is Richmond upon Thames with an average score of 4.91.
C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\913483821.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')


11. What percentage of listings have review scores above 4.5 for each neighborhood?

Use the `review_scores_rating` and `neighbourhood_cleansed` columns.

```

1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Identify listings with review scores above 4.5
6 # Create a new column 'high_review_score' to indicate whether the review score is above 4.5
7 listings['high_review_score'] = listings['review_scores_rating'] > 4.5
8
9 # Calculate the percentage of listings with review scores above 4.5 for each neighborhood
10 # Group the listings by 'neighbourhood_cleansed' and calculate the percentage
11 percentage_high_reviews = listings.groupby('neighbourhood_cleansed')['high_review_score'].mean() * 100
12
13 # Print the result
14 # Formatted string to display the neighborhood and the percentage with two decimal places
15 print("The percentage of listings with review scores above 4.5 for each neighborhood:")
16 for neighborhood, percentage in percentage_high_reviews.items():
17     print(f'{neighborhood}: {percentage:.2f}%')
18

```

 The percentage of listings with review scores above 4.5 for each neighborhood:
Barking and Dagenham: 51.24%
Barnet: 51.93%
Bexley: 52.93%
Brent: 52.70%
Bromley: 62.95%
Camden: 53.97%
City of London: 52.61%
Croydon: 52.12%
Ealing: 55.69%


```

Enfield: 55.49%
Greenwich: 55.47%
Hackney: 63.33%
Hammersmith and Fulham: 56.55%
Haringey: 58.73%
Harrow: 53.08%
Havering: 56.85%
Hillingdon: 50.87%
Hounslow: 53.15%
Islington: 58.50%
Kensington and Chelsea: 51.01%
Kingston upon Thames: 64.66%
Lambeth: 62.84%
Lewisham: 60.52%
Merton: 58.64%
Newham: 50.71%
Redbridge: 51.90%
Richmond upon Thames: 65.95%
Southwark: 59.06%
Sutton: 58.27%
Tower Hamlets: 53.23%
Waltham Forest: 58.37%
Wandsworth: 61.76%
Westminster: 49.00%
C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\3357583205.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')

```

12. What is the trend in the number of reviews over time for the top 3 neighborhoods with the most listings?

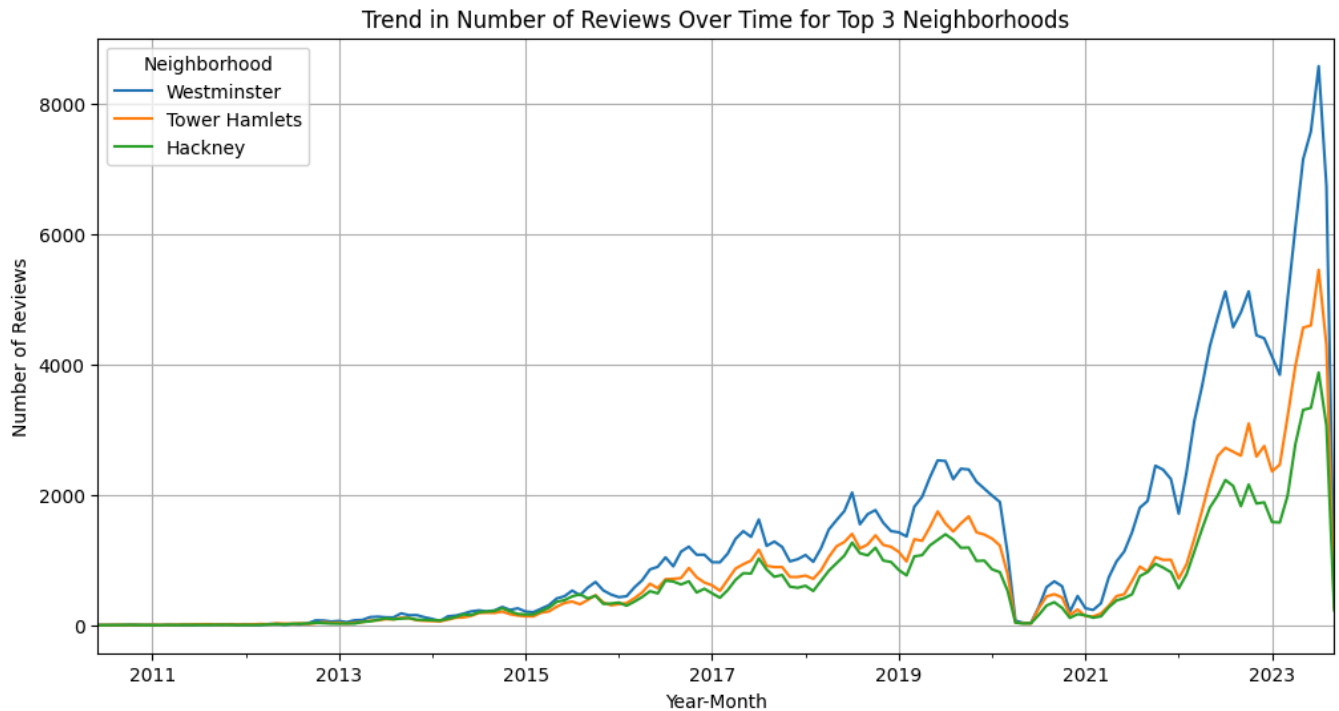
Use the `date` column from the `reviews` dataset and `neighbourhood_cleansed` from the `listings` dataset.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 reviews = pd.read_csv('reviews 2 reduced.csv')
7
8 # Identify the top 3 neighborhoods with the most listings
9 top_neighborhoods = listings['neighbourhood_cleansed'].value_counts().head(3).index.tolist()
10
11 # Filter listings to include only those in the top 3 neighborhoods
12 top_listings = listings[listings['neighbourhood_cleansed'].isin(top_neighborhoods)]
13
14 # Merge the reviews with the top listings to include the 'neighbourhood_cleansed' column
15 top_reviews = reviews.merge(top_listings[['id', 'neighbourhood_cleansed']], left_on='listing_id', right_on='id', how='inner')
16
17 # Convert the 'date' column in the reviews DataFrame to datetime format
18 top_reviews['date'] = pd.to_datetime(top_reviews['date'])
19
20 # Extract the year and month from the 'date' column
21 top_reviews['year_month'] = top_reviews['date'].dt.to_period('M')
22
23 # Aggregate the number of reviews by year and month for each neighborhood
24 reviews_trend = top_reviews.groupby(['year_month', 'neighbourhood_cleansed']).size().unstack(fill_value=0)
25
26 # Plot the trend in the number of reviews over time
27 plt.figure(figsize=(12, 6))
28 for neighborhood in top_neighborhoods:
29     reviews_trend[neighborhood].plot(label=neighborhood)
30
31 plt.title('Trend in Number of Reviews Over Time for Top 3 Neighborhoods')
32 plt.xlabel('Year-Month')
33 plt.ylabel('Number of Reviews')
34 plt.legend(title='Neighborhood')
35 plt.grid(True)
36 plt.show()
37

```

C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\391424437.py:6: DtypeWarning: Columns (68) have mixed types. Specify dtype c
listings = pd.read_csv('listings 2 reduced.csv')



13. What is the average number of reviews for listings hosted by superhosts vs. non-superhosts?

Use the `'number_of_reviews'` and `'host_is_superhost'` columns.

```
1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
4
5 # Convert the 'host_is_superhost' column to boolean
6 # The 'apply' method is used to convert 't' to True and 'f' to False
7 listings['host_is_superhost'] = listings['host_is_superhost'].apply(lambda x: True if x == 't' else False)
8
9 # Calculate the average number of reviews for superhosts and non-superhosts
10 # Group the listings by 'host_is_superhost' and calculate the mean of 'number_of_reviews'
11 average_number_of_reviews = listings.groupby('host_is_superhost')['number_of_reviews'].mean()
12
13 # Get the average number of reviews for superhosts
14 superhost_average = average_number_of_reviews[True]
15
16 # Get the average number of reviews for non-superhosts
17 non_superhost_average = average_number_of_reviews[False]
18
19 # Print the result
20 # Formatted string to display the average number of reviews for superhosts and non-superhosts with two decimal places
21 print(f'The average number of reviews for listings hosted by superhosts is {superhost_average:.2f}.')
22 print(f'The average number of reviews for listings hosted by non-superhosts is {non_superhost_average:.2f}.')
23
```

C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1137874011.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average number of reviews for listings hosted by superhosts is 42.46.
The average number of reviews for listings hosted by non-superhosts is 13.62.

14. What is the correlation between review scores for cleanliness and review scores for value?


Use the `'review_scores_cleanliness'` and `'review_scores_value'` columns.

```
1 import pandas as pd
2
3 listings = pd.read_csv('listings 2 reduced.csv')
```

```

4
5 # Select the relevant columns for correlation analysis
6 # This includes the 'review_scores_cleanliness' and 'review_scores_value' columns
7 correlation_data = listings[['review_scores_cleanliness', 'review_scores_value']]
8
9 # Calculate the correlation matrix for the selected columns
10 correlation_matrix = correlation_data.corr()
11
12 # Extract the correlation value between 'review_scores_cleanliness' and 'review_scores_value'
13 cleanliness_value_correlation = correlation_matrix.loc['review_scores_cleanliness', 'review_scores_value']
14
15 # Print the result
16 # Formatted string to display the correlation value with two decimal places
17 print(f'The correlation between review scores for cleanliness and review scores for value is {cleanliness_value_correlation:.2f}.')
18

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\733381323.py:3: DtypeWarning: Columns (68) have mixed types. Specify dtype c
listings = pd.read_csv('listings 2 reduced.csv')
The correlation between review scores for cleanliness and review scores for value is 0.72.


15. What is the average review score for listings with a minimum stay of more than 7 nights?

Use the `minimum_nights` and `review_scores_rating` columns.

```

1 import pandas as pd
2
3
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Filter listings to include only those with a minimum stay of more than 7 nights
7 # 'query' method is used to filter the DataFrame
8 listings_more_than_7_nights = listings.query('minimum_nights > 7')
9
10 # Calculate the average review score for the filtered listings
11 # 'mean' method is used to calculate the average of 'review_scores_rating'
12 average_review_score = listings_more_than_7_nights['review_scores_rating'].mean()
13
14 # Print the result
15 # Formatted string to display the average review score with two decimal places
16 print(f'The average review score for listings with a minimum stay of more than 7 nights is {average_review_score:.2f}.')
17

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\4183685909.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average review score for listings with a minimum stay of more than 7 nights is 4.60.


16. What is the average number of reviews for listings with at least one review within the last 30 days?

Use the `number_of_reviews` and `number_of_reviews_l30d` columns.

```

1 import pandas as pd
2
3
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Filter listings to include only those with at least one review within the last 30 days
7 # 'query' method is used to filter the DataFrame
8 listings_with_recent_reviews = listings.query('number_of_reviews_l30d > 0')
9
10 # Calculate the average number of reviews for the filtered listings
11 # 'mean' method is used to calculate the average of 'number_of_reviews'
12 average_number_of_reviews = listings_with_recent_reviews['number_of_reviews'].mean()
13
14 # Print the result
15 # Formatted string to display the average number of reviews with two decimal places
16 print(f'The average number of reviews for listings with at least one review within the last 30 days is {average_number_of_reviews:.2f}.')
17


```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1733139817.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The average number of reviews for listings with at least one review within the last 30 days is 40.61.

✓ **17. What is the listing with the best expected revenue based on the last 12 months, considering 60% of guests leave reviews and every guest stays only the minimum number of nights?**

Use both the `listings` and `reviews` datasets and only listings with `minimum_nights` ≤ 7 .

```
1 import pandas as pd
2
3 # Specify the data type for each column to avoid mixed types warning
4 dtype_spec = {
5     'id': 'int64', 'price': 'str', 'minimum_nights': 'int64',
6     'host_is_superhost': 'str'
7 }
8 listings = pd.read_csv('listings 2 reduced.csv', dtype=dtype_spec)
9
10 # Load the reviews data from a CSV file into a pandas DataFrame
11 # Replace 'reviews.csv' with the correct file path if necessary
12 reviews = pd.read_csv('reviews 2 reduced.csv')
13
14 # Filter listings to include only those with minimum_nights <= 7
15 filtered_listings = listings[listings['minimum_nights'] <= 7]
16
17 # Calculate the total number of reviews for each listing from the reviews dataset
18 # 'groupby' groups the data by 'listing_id', and 'size' counts the number of reviews per listing
19 total_reviews_per_listing = reviews.groupby('listing_id').size().reset_index(name='total_reviews')
20
21 # Merge the filtered listings with the total reviews per listing
22 # 'merge' joins the two DataFrames on the 'id' and 'listing_id' columns
23 listings_with_reviews = filtered_listings.merge(total_reviews_per_listing, left_on='id', right_on='listing_id', how='left')
24
25 # Fill NaN values in 'total_reviews' column with 0
26 listings_with_reviews['total_reviews'] = listings_with_reviews['total_reviews'].fillna(0)
27
28 # Calculate the expected number of bookings for each listing
29 # Assuming 60% of guests leave reviews, expected_bookings = total_reviews / 0.6
30 listings_with_reviews['expected_bookings'] = listings_with_reviews['total_reviews'] / 0.6
31
32 # Convert the 'price' column to a numeric type after removing the currency symbol
33 # The 'replace' method is used to remove the dollar sign, and 'astype' converts the column to float
34 listings_with_reviews['price'] = listings_with_reviews['price'].replace('[\$', ' ', regex=True).astype(float)
35
36 # Calculate the expected revenue for each listing
37 # expected_revenue = expected_bookings * price * minimum_nights
38 listings_with_reviews['expected_revenue'] = listings_with_reviews['expected_bookings'] * listings_with_reviews['price'] * listings_w
39
40 # Identify the listing with the best expected revenue
41 # 'idxmax' returns the index of the maximum value in the 'expected_revenue' column
42 best_listing_index = listings_with_reviews['expected_revenue'].idxmax()
43 best_listing = listings_with_reviews.loc[best_listing_index]
44
45 # Print the result
46 # Formatted string to display the listing details and the expected revenue with two decimal places
47 print(f'The listing with the best expected revenue is:')
48 print(f"Listing ID: {best_listing['id']}")
49 print(f"Expected Revenue: ${best_listing['expected_revenue']:.2f}")
50 print(f"Price: ${best_listing['price']:.2f} per night")
51 print(f"Minimum Nights: {best_listing['minimum_nights']} nights")
52 print(f"Total Reviews: {best_listing['total_reviews']}")
53 print(f"Expected Bookings: {best_listing['expected_bookings']:.2f}")
54
```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\683736179.py:10: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv', dtype=dtype_spec)
The listing with the best expected revenue is:
Listing ID: 13254774
Expected Revenue: \$26526060.00
Price: \$53588.00 per night
Minimum Nights: 3 nights
Total Reviews: 99.0
Expected Bookings: 165.00


✓ **18. Which host attribute has the second-highest correlation with the number of reviews for the listing?**

Use the following host attributes: `host_since`, `host_listings_count`, `host_identity_verified`, `calculated_host_listings_count`, and `host_is_s

```

1 import pandas as pd
2
3 # Load the listings data from a CSV file into a pandas DataFrame
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'host_is_superhost' column to boolean
7 # The 'apply' method is used to convert 't' to True and 'f' to False
8 listings['host_is_superhost'] = listings['host_is_superhost'].apply(lambda x: True if x == 't' else False)
9
10 # Convert the 'host_identity_verified' column to boolean
11 # The 'apply' method is used to convert 't' to True and 'f' to False
12 listings['host_identity_verified'] = listings['host_identity_verified'].apply(lambda x: True if x == 't' else False)
13
14 # Convert the 'host_since' column to datetime format
15 listings['host_since'] = pd.to_datetime(listings['host_since'])
16
17 # Calculate the number of days since the host joined
18 # This is done by subtracting the 'host_since' date from today's date
19 listings['days_since_host'] = (pd.to_datetime('today') - listings['host_since']).dt.days
20
21 # Select the relevant columns for correlation analysis
22 # This includes the 'number_of_reviews' column and the specified host attributes
23 correlation_data = listings[[
24     'number_of_reviews', 'days_since_host', 'host_listings_count',
25     'host_identity_verified', 'calculated_host_listings_count', 'host_is_superhost'
26 ]]
27
28 # Calculate the correlation matrix for the selected columns
29 correlation_matrix = correlation_data.corr()
30
31 # Extract the correlation values between 'number_of_reviews' and each host attribute
32 # We use the 'number_of_reviews' row from the correlation matrix
33 reviews_correlations = correlation_matrix.loc['number_of_reviews', [
34     'days_since_host', 'host_listings_count', 'host_identity_verified',
35     'calculated_host_listings_count', 'host_is_superhost'
36 ]]
37
38 # Identify the host attribute with the highest and second-highest correlation with 'number_of_reviews'
39 sorted_correlations = reviews_correlations.sort_values(ascending=False)
40 second_highest_correlation_attribute = sorted_correlations.index[1]
41 second_highest_correlation_value = sorted_correlations.iloc[1]
42
43 # Print the result
44 # Formatted string to display the attribute and the correlation value with two decimal places
45 print(f'The host attribute with the second-highest correlation with the number of reviews is {second_highest_correlation_attribute} with a correlation of {second_highest_correlation_value:.2f}.')
46

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1630735133.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The host attribute with the second-highest correlation with the number of reviews is days_since_host with a correlation of 0.14.

19. What is the percentage of listings belonging to hosts with identity verification enabled?


Use the 'host_identity_verified' column.

```

1 import pandas as pd
2
3 # Load the listings data from a CSV file into a pandas DataFrame
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'host_identity_verified' column to boolean
7 # The 'apply' method is used to convert 't' to True and 'f' to False
8 listings['host_identity_verified'] = listings['host_identity_verified'].apply(lambda x: True if x == 't' else False)
9
10 # Calculate the total number of listings
11 total_listings = len(listings)
12
13 # Calculate the number of listings with identity verification enabled
14 listings_with_verified_identity = listings['host_identity_verified'].sum()
15
16 # Calculate the percentage of listings with identity verification enabled
17 percentage_verified_identity = (listings_with_verified_identity / total_listings) * 100
18
19 # Print the result
20 # Formatted string to display the percentage with two decimal places

```

```
21 print(f'The percentage of listings belonging to hosts with identity verification enabled is {percentage_verified_identity:.2f}%.')
22
```


 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\3214231501.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype listings = pd.read_csv('listings 2 reduced.csv')

The percentage of listings belonging to hosts with identity verification enabled is 87.32%.

✓ 20. Which neighborhood has the highest proportion of professional hosts (more than 5 listings)?

Use the `host_listings_count` and `neighbourhood_cleansed` columns.

```
1 import pandas as pd
2
3 # Load the listings data from a CSV file into a pandas DataFrame
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Identify professional hosts (hosts with more than 5 listings)
7 # Create a new column 'is_professional_host' to indicate whether the host is professional
8 listings['is_professional_host'] = listings['host_listings_count'] > 5
9
10 # Calculate the total number of listings in each neighborhood
11 total_listings_per_neighborhood = listings['neighbourhood_cleansed'].value_counts()
12
13 # Calculate the number of professional host listings in each neighborhood
14 professional_listings_per_neighborhood = listings[listings['is_professional_host']][ 'neighbourhood_cleansed'].value_counts()
15
16 # Calculate the proportion of professional host listings in each neighborhood
17 proportion_professional_hosts = (professional_listings_per_neighborhood / total_listings_per_neighborhood) * 100
18
19 # Find the neighborhood with the highest proportion of professional hosts
20 highest_proportion_neighborhood = proportion_professional_hosts.idxmax()
21 highest_proportion_value = proportion_professional_hosts.max()
22
23 # Print the result
24 # Formatted string to display the neighborhood and the proportion with two decimal places
25 print(f'The neighborhood with the highest proportion of professional hosts is {highest_proportion_neighborhood} with a proportion of
26
```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\1205793555.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype listings = pd.read_csv('listings 2 reduced.csv')

The neighborhood with the highest proportion of professional hosts is City of London with a proportion of 61.75%.

✓ 21. What is the average tenure (in years) of superhosts on the platform compared to non-superhosts?


Use the `host_since` and `host_is_superhost` columns.

```
1 import pandas as pd
2
3 # Load the listings data from a CSV file into a pandas DataFrame
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'host_is_superhost' column to boolean
7 # The 'apply' method is used to convert 't' to True and 'f' to False
8 listings['host_is_superhost'] = listings['host_is_superhost'].apply(lambda x: True if x == 't' else False)
9
10 # Convert the 'host_since' column to datetime format
11 listings['host_since'] = pd.to_datetime(listings['host_since'])
12
13 # Calculate the number of days since the host joined
14 # This is done by subtracting the 'host_since' date from today's date
15 listings['days_since_host'] = (pd.to_datetime('today') - listings['host_since']).dt.days
16
17 # Calculate the tenure in years by dividing the number of days by 365
18 listings['tenure_years'] = listings['days_since_host'] / 365.25
19
20 # Calculate the average tenure for superhosts
21 average_tenure_superhosts = listings[listings['host_is_superhost']]['tenure_years'].mean()
22
23 # Calculate the average tenure for non-superhosts
24 average_tenure_non_superhosts = listings[~listings['host_is_superhost']]['tenure_years'].mean()
25
26 # Print the result
```

```

27 # Formatted string to display the average tenure for superhosts and non-superhosts with two decimal places
28 print(f'The average tenure of superhosts on the platform is {average_tenure_superhosts:.2f} years.')
29 print(f'The average tenure of non-superhosts on the platform is {average_tenure_non_superhosts:.2f} years.')
30

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\25816835.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype of listings = pd.read_csv('listings 2 reduced.csv')
 The average tenure of superhosts on the platform is 8.28 years.
 The average tenure of non-superhosts on the platform is 7.82 years.


22. What is the average response rate for hosts with more than 10 listings?

Use the `host_response_rate` and `host_listings_count` columns.

```

1 import pandas as pd
2
3 # Load the listings data from a CSV file into a pandas DataFrame
4 listings = pd.read_csv('listings 2 reduced.csv')
5
6 # Convert the 'host_response_rate' column to numeric type after removing the '%' symbol
7 # The 'replace' method is used to remove the percentage sign, and 'astype' converts the column to float
8 listings['host_response_rate'] = listings['host_response_rate'].replace('%', '', regex=True).astype(float)
9
10 # Filter listings to include only those where hosts have more than 10 listings
11 filtered_listings = listings[listings['host_listings_count'] > 10]
12
13 # Calculate the average response rate for hosts with more than 10 listings
14 average_response_rate = filtered_listings['host_response_rate'].mean()
15
16 # Print the result
17 # Formatted string to display the average response rate with two decimal places
18 print(f'The average response rate for hosts with more than 10 listings is {average_response_rate:.2f}%.')
19

```

 C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\2027346649.py:5: DtypeWarning: Columns (68) have mixed types. Specify dtype listings = pd.read_csv('listings 2 reduced.csv')
 The average response rate for hosts with more than 10 listings is 95.92%.

23. What is the relationship between the host's response time and the number of reviews their listings receive?

Use the `host_response_time` and `number_of_reviews` columns.

```

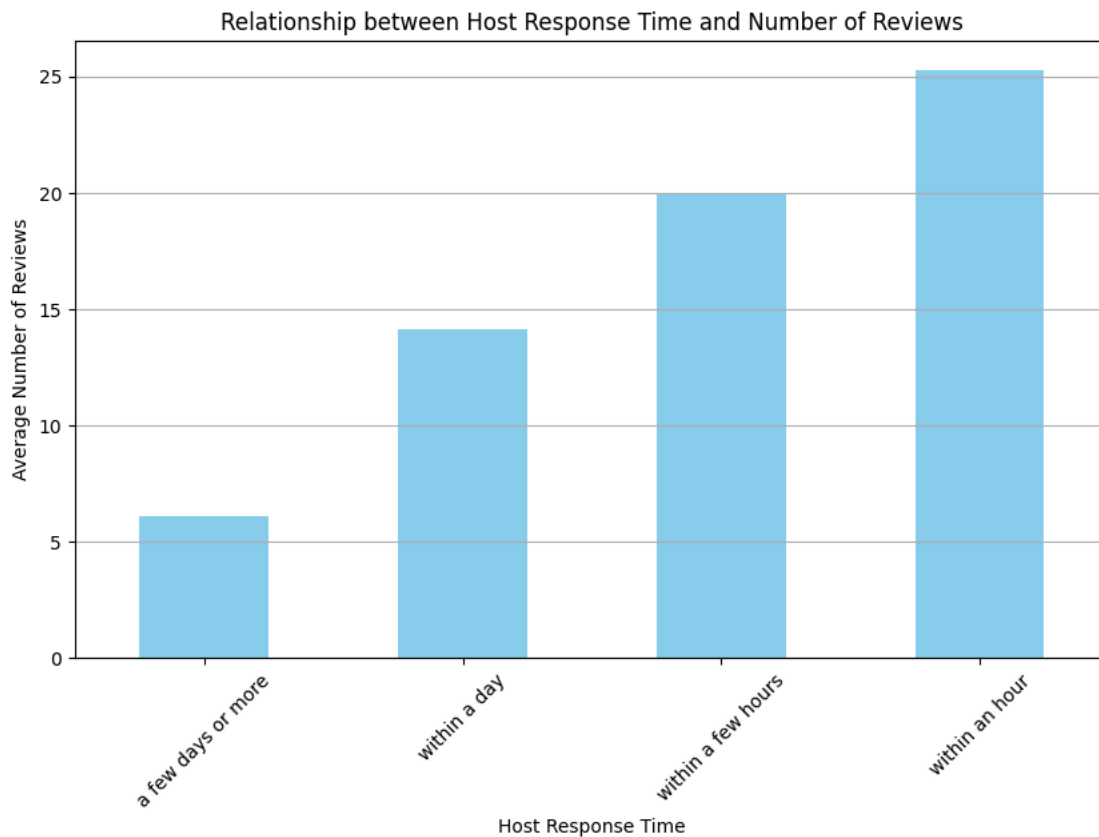
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the listings data from a CSV file into a pandas DataFrame
5 listings = pd.read_csv('listings 2 reduced.csv')
6
7 # Calculate the average number of reviews for each host response time category
8 # Group the listings by 'host_response_time' and calculate the mean of 'number_of_reviews'
9 average_reviews_by_response_time = listings.groupby('host_response_time')['number_of_reviews'].mean()
10
11 # Print the average number of reviews for each host response time category
12 # Formatted string to display the average number of reviews with two decimal places
13 print("Average number of reviews for each host response time category:")
14 for response_time, avg_reviews in average_reviews_by_response_time.items():
15     print(f'{response_time}: {avg_reviews:.2f}')
16
17 # Plot the relationship between host response time and number of reviews
18 plt.figure(figsize=(10, 6))
19 average_reviews_by_response_time.plot(kind='bar', color='skyblue')
20 plt.title('Relationship between Host Response Time and Number of Reviews')
21 plt.xlabel('Host Response Time')
22 plt.ylabel('Average Number of Reviews')
23 plt.xticks(rotation=45)
24 plt.grid(axis='y')
25 plt.show()
26

```

```

C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\2855346598.py:6: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
Average number of reviews for each host response time category:
a few days or more: 6.07
within a day: 14.10
within a few hours: 20.00
within an hour: 25.26

```



24. What are the top 3 most common amenities provided by listings in the dataset?

Parse the `amenities` column.

```

1 import pandas as pd
2 from collections import Counter
3
4 # Load the listings data from a CSV file into a pandas DataFrame
5 listings = pd.read_csv('listings 2 reduced.csv')
6
7 # Parse the 'amenities' column
8 # Remove curly braces and split the string by commas to get individual amenities
9 listings['amenities'] = listings['amenities'].str.replace('{}', '', regex=True).str.split(',')
10
11 # Flatten the list of amenities and count the occurrences
12 amenities_list = listings['amenities'].explode().str.strip()
13 amenities_counter = Counter(amenities_list)
14
15 # Identify the top 3 most common amenities
16 top_3_amenities = amenities_counter.most_common(3)
17
18 # Print the result
19 # Formatted string to display the top 3 amenities
20 print("The top 3 most common amenities provided by listings are:")
21 for amenity, count in top_3_amenities:
22     print(f'{amenity}: {count} listings')
23

```

```

C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\2207626809.py:6: DtypeWarning: Columns (68) have mixed types. Specify dtype
listings = pd.read_csv('listings 2 reduced.csv')
The top 3 most common amenities provided by listings are:
"Wifi": 78388 listings
"Smoke alarm": 78368 listings
"Kitchen": 77956 listings

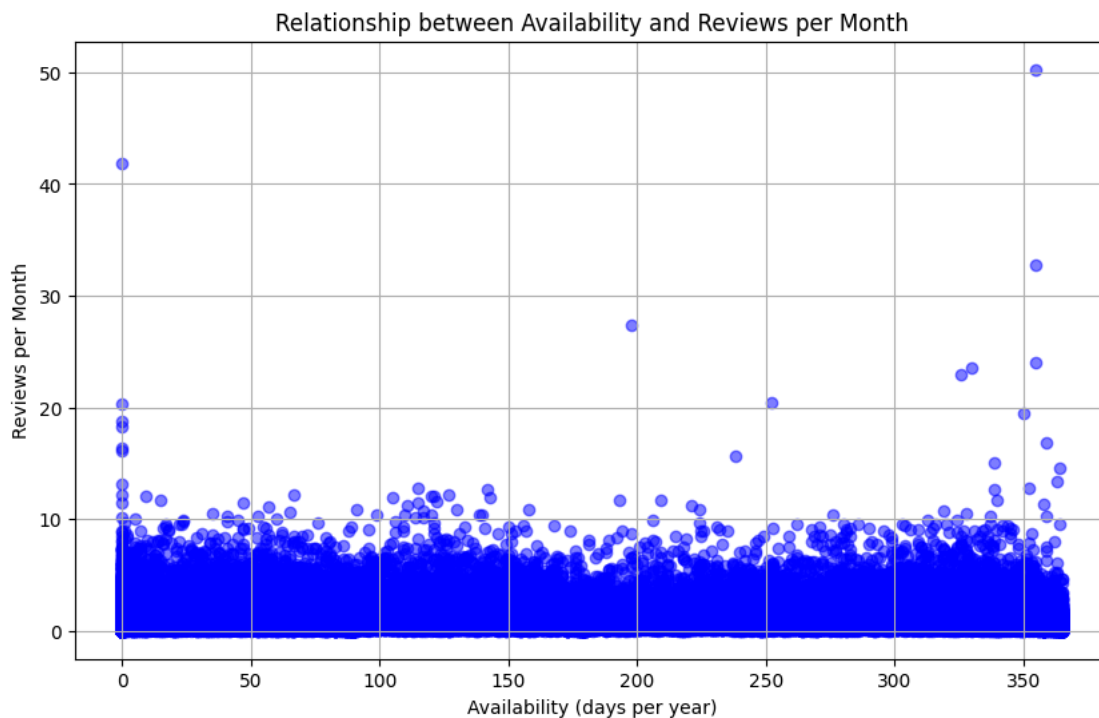
```


✓ 25. What is the relationship between the availability of a listing and the average number of reviews it receives per month?

Use the `availability_365` and `reviews_per_month` columns.

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the listings data from a CSV file into a pandas DataFrame
5 listings = pd.read_csv('listings 2 reduced.csv')
6
7 # Calculate the correlation between availability and reviews per month
8 correlation = listings[['availability_365', 'reviews_per_month']].corr().loc['availability_365', 'reviews_per_month']
9
10 # Print the correlation value
11 # Formatted string to display the correlation value with two decimal places
12 print(f'The correlation between availability and reviews per month is {correlation:.2f}.')
13
14 # Plot the relationship between availability and reviews per month
15 plt.figure(figsize=(10, 6))
16 plt.scatter(listings['availability_365'], listings['reviews_per_month'], alpha=0.5, color='blue')
17 plt.title('Relationship between Availability and Reviews per Month')
18 plt.xlabel('Availability (days per year)')
19 plt.ylabel('Reviews per Month')
20 plt.grid(True)
21 plt.show()
22
```

↗ C:\Users\Manish Gupta\AppData\Local\Temp\ipykernel_6440\211332429.py:6: DtypeWarning: Columns (68) have mixed types. Specify dtype c
listings = pd.read_csv('listings 2 reduced.csv')
The correlation between availability and reviews per month is 0.18.



Questions :

✓ Price and Neighborhood Analysis

1. What is the neighborhood in which superhosts have the biggest median price difference with respect to non-superhosts?

Use the `host_is_superhost`, `neighbourhood_cleansed`, and `price` columns.

2. What is the average price difference between a professional host and a non-professional one?

A professional host is defined as having listings in more than 5 different locations (`neighbourhood_cleansed`).

3. **What is the median price premium given to entire homes/apartments with respect to other listings in the same neighborhood?**
Use the `room_type` column to distinguish listing types and compute the average across all neighborhoods.
 4. **Which neighborhood has the highest density of listings per square kilometer?**
Use the `latitude`, `longitude`, and `neighbourhood_cleansed` columns.
 5. **What is the average price per guest for listings in each room type category?**
Use the `price` and `accommodates` columns.
 6. **What is the percentage of listings priced above the overall average price in each neighborhood?**
Use the `price` and `neighbourhood_cleansed` columns.
 7. **Which neighborhood has the highest average revenue per listing based on the last 12 months?**
Use the `reviews_per_month`, `price`, and `minimum_nights` columns.
-

Review Scores and Trends

8. **What is the review score attribute that has the highest correlation with the price of listings?**
Analyze `review_scores_rating`, `review_scores_cleanliness`, etc., against `price`.
 9. **What is the average difference between review scores of superhosts vs. normal hosts?**
Use the `review_scores_rating` column.
 10. **Which neighborhood has the highest average review score for communication?**
Use the `review_scores_communication` and `neighbourhood_cleansed` columns.
 11. **What percentage of listings have review scores above 4.5 for each neighborhood?**
Use the `review_scores_rating` and `neighbourhood_cleansed` columns.
 12. **What is the trend in the number of reviews over time for the top 3 neighborhoods with the most listings?**
Use the `date` column from the `reviews` dataset and `neighbourhood_cleansed` from the `listings` dataset.
 13. **What is the average number of reviews for listings hosted by superhosts vs. non-superhosts?**
Use the `number_of_reviews` and `host_is_superhost` columns.
 14. **What is the correlation between review scores for cleanliness and review scores for value?**
Use the `review_scores_cleanliness` and `review_scores_value` columns.
 15. **What is the average review score for listings with a minimum stay of more than 7 nights?**
Use the `minimum_nights` and `review_scores_rating` columns.
 16. **What is the average number of reviews for listings with at least one review within the last 30 days?**
Use the `number_of_reviews` and `number_of_reviews_130d` columns.
-

Host and Listing Attributes

17. **What is the listing with the best expected revenue based on the last 12 months, considering 60% of guests leave reviews and every guest stays only the minimum number of nights?**
Use both the `listings` and `reviews` datasets and only listings with `minimum_nights <= 7`.
 18. **Which host attribute has the second-highest correlation with the number of reviews for the listing?**
Use the following host attributes: `host_since`, `host_listings_count`, `host_identity_verified`, `calculated_host_listings_count`, and `host_is_superhost`.
 19. **What is the percentage of listings belonging to hosts with identity verification enabled?**
Use the `host_identity_verified` column.
 20. **Which neighborhood has the highest proportion of professional hosts (more than 5 listings)?**
Use the `host_listings_count` and `neighbourhood_cleansed` columns.
 21. **What is the average tenure (in years) of superhosts on the platform compared to non-superhosts?**
Use the `host_since` and `host_is_superhost` columns.
 22. **What is the average response rate for hosts with more than 10 listings?**
Use the `host_response_rate` and `host_listings_count` columns.
 23. **What is the relationship between the host's response time and the number of reviews their listings receive?**
Use the `host_response_time` and `number_of_reviews` columns.
-

Amenities and Availability

24. **What are the top 3 most common amenities provided by listings in the dataset?**
Parse the `amenities` column.

25. What is the relationship between the availability of a listing and the average number of reviews it receives per month?

Use the `availability_365` and `reviews_per_month` columns.

Python Question : 1

You are given an array `prices` where `prices[i]` represents the price of a given stock on the i -th day.

Your goal is to maximize profit by selecting **one day** to buy the stock and **another day in the future** to sell it. Return the maximum profit you can achieve from this transaction. If no profit can be achieved, return `0`.

Examples:

Example 1:

- **Input:** `prices = [7,1,5,3,6,4]`
- **Output:** `5`
- **Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), resulting in profit = $6 - 1 = 5$.
Note that buying on day 2 and selling on day 1 is not allowed as you must buy before selling.

Example 2:

- **Input:** `prices = [7,6,4,3,1]`
- **Output:** `0`
- **Explanation:** No profitable transactions are possible, so the maximum profit is `0`.

Constraints:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

Let me know if you'd like help solving this problem!

```
1 def max_profit(prices):
2     # Initialize minimum price to a large value
3     min_price = float('inf')
4     # Initialize maximum profit to 0
5     max_profit = 0
6
7     # Iterate through each price in the array
8     for price in prices:
9         # Update minimum price seen so far
10        min_price = min(min_price, price)
11        # Calculate potential profit for the current price
12        profit = price - min_price
13        # Update maximum profit if the calculated profit is greater
14        max_profit = max(max_profit, profit)
15
16    # Return the maximum profit found
17    return max_profit
18
19 # Test cases
20 prices1 = [7, 1, 5, 3, 6, 4]
21 prices2 = [7, 6, 4, 3, 1]
22
23 # Expected outputs are 5 and 0 respectively
24 print(f'Example 1: {max_profit(prices1)}') # Output: 5
25 print(f'Example 2: {max_profit(prices2)}') # Output: 0
26
```

Example 1: 5
Example 2: 0

Question 2 :

You are given an array of positive integers `nums` and a positive integer `target`. Your task is to return the **minimal length** of a contiguous subarray whose sum is **greater than or equal to** `target`. If no such subarray exists, return `0`.

Examples:

Example 1:

- **Input:** target = 7, nums = [2,3,1,2,4,3]
- **Output:** 2
- **Explanation:** The subarray [4,3] has the minimal length under the given constraints.

Example 2:

- **Input:** target = 4, nums = [1,4,4]
- **Output:** 1
- **Explanation:** The subarray [4] satisfies the condition with minimal length.

Example 3:

- **Input:** target = 11, nums = [1,1,1,1,1,1,1,1]
- **Output:** 0
- **Explanation:** There is no subarray whose sum is greater than or equal to 11.

Constraints:

- $1 \leq \text{target} \leq 10^9$
- $1 \leq \text{nums.length} \leq 10^5$
- $1 \leq \text{nums}[i] \leq 10^4$

```
1 def min_subarray_len(target, nums):
2     # Initialize pointers and variables
3     left = 0
4     current_sum = 0
5     min_length = float('inf')
6
7     # Iterate through the array with the right pointer
8     for right in range(len(nums)):
9         # Add the current element to the current_sum
10        current_sum += nums[right]
11
12        # While the current_sum is greater than or equal to the target
13        while current_sum >= target:
14            # Update the minimal length of the subarray
15            min_length = min(min_length, right - left + 1)
16            # Shrink the window from the left
17            current_sum -= nums[left]
18            left += 1
19
20    # Return the minimal length or 0 if no such subarray exists
21    return min_length if min_length != float('inf') else 0
22
23 # Test cases
24 target1 = 7
25 nums1 = [2, 3, 1, 2, 4, 3]
26
27 target2 = 4
28 nums2 = [1, 4, 4]
29
30 target3 = 11
31 nums3 = [1, 1, 1, 1, 1, 1, 1, 1]
32
33 # Expected outputs are 2, 1, and 0 respectively
34 print(f'Example 1: {min_subarray_len(target1, nums1)}') # Output: 2
35 print(f'Example 2: {min_subarray_len(target2, nums2)}') # Output: 1
36 print(f'Example 3: {min_subarray_len(target3, nums3)}') # Output: 0
37
```

↗ Example 1: 2
Example 2: 1
Example 3: 0

✓ Question 3 :

Given a string *s* and an integer *k*, return the **maximum number of vowel letters** in any substring of *s* with a length of *k*. Vowel letters in English are 'a', 'e', 'i', 'o', and 'u'.

Examples:**Example 1:**

- **Input:** s = "abciidef", k = 3

- **Output:** 3
- **Explanation:** The substring "iii" contains 3 vowel letters.

Example 2:

- **Input:** s = "aeiou", k = 2
- **Output:** 2
- **Explanation:** Any substring of length 2 contains 2 vowels.

Example 3:

- **Input:** s = "leetcode", k = 3
- **Output:** 2
- **Explanation:** The substrings "lee", "eet", and "ode" each contain 2 vowels.


Constraints:

- $1 \leq s.length \leq 10^5$
- s consists of lowercase English letters.
- $1 \leq k \leq s.length$

```

1 def max_vowels(s, k):
2     # Define a set of vowels for quick lookup
3     vowels = {'a', 'e', 'i', 'o', 'u'}
4
5     # Initialize the current number of vowels in the window and the maximum number of vowels
6     current_vowels = 0
7     max_vowels = 0
8
9     # Iterate through the first k characters to initialize the first window
10    for i in range(k):
11        if s[i] in vowels:
12            current_vowels += 1
13
14    # Initialize the maximum number of vowels with the first window
15    max_vowels = current_vowels
16
17    # Slide the window through the string
18    for i in range(k, len(s)):
19        # Add the new character at the right end of the window
20        if s[i] in vowels:
21            current_vowels += 1
22        # Remove the character at the left end of the window
23        if s[i - k] in vowels:
24            current_vowels -= 1
25        # Update the maximum number of vowels
26        max_vowels = max(max_vowels, current_vowels)
27
28    # Return the maximum number of vowels found
29    return max_vowels
30
31 # Test cases
32 s1 = "abciideef"
33 k1 = 3
34
35 s2 = "aeiou"
36 k2 = 2
37
38 s3 = "leetcode"
39 k3 = 3
40
41 # Expected outputs are 3, 2, and 2 respectively
42 print(f'Example 1: {max_vowels(s1, k1)}') # Output: 3
43 print(f'Example 2: {max_vowels(s2, k2)}') # Output: 2
44 print(f'Example 3: {max_vowels(s3, k3)}') # Output: 2
45

```

 Example 1: 3
 Example 2: 2
 Example 3: 2

✓ Question 4 :

Given a string `S = 'abciiaaec'`, write a Python function to find all substrings consisting of **consecutive repeating characters**. Determine the **maximum length** of such substrings and return all substrings with this maximum length if there are multiple.

Example:

Input: `S = 'abciiaec'`


Output:

- Maximum Length: 3
- Substrings: ['iii']

Constraints:

- The input string `S` consists of lowercase English letters.
- The function should handle strings of varying lengths efficiently.

```
1 def find_max_repeating_substrings(S):
2     # Initialize variables
3     max_length = 0
4     current_length = 0
5     max_substrings = []
6     current_char = None
7     current_substring = ''
8
9     # Iterate through the string
10    for char in S:
11        if char == current_char:
12            current_length += 1
13            current_substring += char
14        else:
15            # Check if the current substring is the longest
16            if current_length > max_length:
17                max_length = current_length
18                max_substrings = [current_substring]
19            elif current_length == max_length:
20                max_substrings.append(current_substring)
21
22            # Reset for the new character
23            current_char = char
24            current_length = 1
25            current_substring = char
26
27    # Final check for the last substring
28    if current_length > max_length:
29        max_length = current_length
30        max_substrings = [current_substring]
31    elif current_length == max_length:
32        max_substrings.append(current_substring)
33
34    return max_length, max_substrings
35
36 # Test case
37 S = 'abciiaec'
38 max_length, max_substrings = find_max_repeating_substrings(S)
39
40 # Expected output: Maximum Length: 3, Substrings: ['iii']
41 print(f'Maximum Length: {max_length}')
42 print(f'Substrings: {max_substrings}')
43
```

 Maximum Length: 3
Substrings: ['iii']

✓ Question 5:

Question: 238. Product of Array Except Self

Given an array, find the product of all elements except the element itself for each position in the array.

Examples:

Example 1:

- **Input:** [1, 2, 3, 4]
- **Output:** [24, 12, 8, 6]

Example 2:

- **Input:** [-1, 3, 0, -3, 1]
- **Output:** [0, 0, 9, 0, 0]

Approach to Explain to the Interviewer:

1. Brute Force Approach:

- Iterate through the array for each element and calculate the product of all other elements.
- **Time Complexity:** $O(n^2)$

2. Optimized Approach Using Prefix and Suffix Products:

- **Prefix Products:** For each element, calculate the product of all elements to its left.
- **Suffix Products:** For each element, calculate the product of all elements to its right.
- Multiply the prefix and suffix products for each element to get the result.
- **Time Complexity:** $O(n)$
- **Space Complexity:** $O(n)$

3. Optimized Space Approach:

- Use the input array to store prefix and suffix products to reduce space usage.
- **Time Complexity:** $O(n)$
- **Space Complexity:** $O(1)$ (excluding the output array).

```
1 def find_max_repeating_substrings(S):
2     # Initialize variables
3     max_length = 0
4     current_length = 0
5     max_substrings = []
6     current_char = None
7     current_substring = ''
8
9     # Iterate through the string
10    for char in S:
11        if char == current_char:
12            current_length += 1
13            current_substring += char
14        else:
15            # Check if the current substring is the longest
16            if current_length > max_length:
17                max_length = current_length
18                max_substrings = [current_substring]
19            elif current_length == max_length:
20                max_substrings.append(current_substring)
21
22            # Reset for the new character
23            current_char = char
24            current_length = 1
25            current_substring = char
26
27    # Final check for the last substring
28    if current_length > max_length:
29        max_length = current_length
30        max_substrings = [current_substring]
31    elif current_length == max_length:
32        max_substrings.append(current_substring)
33
34    return max_length, max_substrings
35
36 # Test case
37 S = 'abciiaec'
38 max_length, max_substrings = find_max_repeating_substrings(S)
39
40 # Expected output: Maximum Length: 3, Substrings: ['iii']
41 print(f'Maximum Length: {max_length}')
42 print(f'Substrings: {max_substrings}')
43
```

```
➦ Maximum Length: 3
  Substrings: ['iii']
```

✓ Question 6 :

Problem Statement

You are tasked with finding **double pairs** in a given array. A **double pair** consists of two numbers, where one is exactly double the other. The array provided contains unique integers. Your goal is to return the total number of such pairs in the array.

Input Format

1. The first line contains an integer N , representing the number of elements in the array.
2. The next N lines contain the array elements.

Output Format

- Return a single integer, denoting the count of double pairs in the array.

Constraints

- $1 \leq N \leq 10^5$
- $1 \leq \text{arr}[i] \leq 10^4$

Example:

Sample Input:

```
6
7
12
14
28
```

Sample Output:

```
3
```

Explanation:

The following **3 double pairs** exist in the array:

- (6, 12)
- (7, 14)
- (14, 28)

Thus, the output is 3.

Execution Time Limit:

- 10 seconds

```
1 def count_double_pairs(arr):
2     # Create a set for quick lookup of elements
3     num_set = set(arr)
4     double_pairs_count = 0
5
6     # Iterate through each number in the array
7     for num in arr:
8         # Check if double of the number exists
9         if num * 2 in num_set:
10             double_pairs_count += 1
11         # Check if half of the number exists (and it should be even)
12         if num % 2 == 0 and num // 2 in num_set:
13             double_pairs_count += 1
14
15     return double_pairs_count // 2 # Each pair is counted twice
16
17 # Sample Input
18 arr = [6, 7, 12, 14, 28]
19
20 # Expected output: 3
21 print(count_double_pairs(arr))
22
```

 3

✓ Leetcode Questions:

✓ 1.

You are given an integer array `nums` and an integer `k`.

In one operation, you can pick two numbers from the array whose sum equals `k` and remove them from the array.

Return the maximum number of operations you can perform on the array.

Example 1:

Input: `nums = [1,2,3,4]`, `k = 5` Output: 2 Explanation: Starting with `nums = [1,2,3,4]`:

- Remove numbers 1 and 4, then `nums = [2,3]`
- Remove numbers 2 and 3, then `nums = []` There are no more pairs that sum up to 5, hence a total of 2 operations. Example 2:

Input: `nums = [3,1,3,4,3]`, `k = 6` Output: 1 Explanation: Starting with `nums = [3,1,3,4,3]`:

- Remove the first two 3's, then `nums = [1,4,3]` There are no more pairs that sum up to 6, hence a total of 1 operation.

Constraints:

$1 \leq \text{nums.length} \leq 105$ $1 \leq \text{nums}[i] \leq 109$ $1 \leq k \leq 109$

<https://leetcode.com/problems/max-number-of-k-sum-pairs/description/?envType=study-plan-v2&envId=leetcode-75>

```
1 # n = [3,1,3,4,3]
2 # k = 6
3 # lst = []
4 # for i in range(len(n)):
5 #     for j in range(i+1, len(n)):
6 #         if n[i]+n[j] == k:
7 #             lst.append((i,j))
8 #             n.pop(i)
9 #             n.pop
10 # len(lst)
11
```

```
1 def max_operations(nums, k):
2     # Initialize the hash map to store the frequency of numbers
3     num_count = {}
4     max_operations_count = 0
5
6     # Iterate through each number in the array
7     for num in nums:
8         complement = k - num
9         # Check if the complement exists in the hash map and has a positive count
10        if complement in num_count and num_count[complement] > 0:
11            # Perform the operation by decrementing the count
12            num_count[complement] -= 1
13            max_operations_count += 1
14        else:
15            # Increment the count of the current number in the hash map
16            if num in num_count:
17                num_count[num] += 1
18            else:
19                num_count[num] = 1
20
21    return max_operations_count
22
23 # Test cases
24 nums1 = [1, 2, 3, 4]
25 k1 = 5
26
27 nums2 = [3, 1, 3, 4, 3]
28 k2 = 6
29
30 # Expected outputs are 2 and 1 respectively
31 print(f'Example 1: {max_operations(nums1, k1)}') # Output: 2
32 print(f'Example 2: {max_operations(nums2, k2)}') # Output: 1
33
```

↔ Example 1: 2
Example 2: 1

✓ 2.

Given an integer array `nums`, return an array `answer` such that `answer[i]` is equal to the product of all the elements of `nums` except `nums[i]`. Given an integer array `nums`, return an array `answer` such that `answer[i]` is equal to the product of all the elements of `nums` except `nums[i]`.

The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in $O(n)$ time and without using the division operation.

Example 1:

Input: nums = [1,2,3,4] Output: [24,12,8,6] Example 2:

Input: nums = [-1,1,0,-3,3] Output: [0,0,9,0,0]

Constraints:

$2 \leq \text{nums.length} \leq 10^5$ $-30 \leq \text{nums}[i] \leq 30$ The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

Follow up: Can you solve the problem in $O(1)$ extra space complexity? (The output array does not count as extra space for space complexity analysis.)

<https://leetcode.com/problems/product-of-array-except-self/description/?envType=study-plan-v2&envId=top-interview-150>

```
1 nums = [1,2,3,4,5]
2 Output= [24,12,8,6]
3 product = 1
4 for i in nums:
5     if i == 0:
6         continue
7     else:
8         product*= i
9 res = []
10 for j in nums:
11     if j == 0:
12         res.append(product)
13     else:
14         res.append(product/j)
15 res
```

→ [120.0, 60.0, 40.0, 30.0, 24.0, 120]

```
1 def product_except_self(nums):
2     n = len(nums)
3
4     # Initialize the answer array
5     answer = [1] * n
6
7     # Calculate left products
8     left_product = 1
9     for i in range(n):
10         answer[i] = left_product
11         left_product *= nums[i]
12
13     # Calculate right products and update the answer array
14     right_product = 1
15     for i in range(n - 1, -1, -1):
16         answer[i] *= right_product
17         right_product *= nums[i]
18
19     return answer
20
21 # Test cases
22 nums1 = [1, 2, 3, 4]
23 nums2 = [-1, 1, 0, -3, 3]
24
25 # Expected outputs are [24, 12, 8, 6] and [0, 0, 9, 0, 0] respectively
26 print(f'Example 1: {product_except_self(nums1)}') # Output: [24, 12, 8, 6]
27 print(f'Example 2: {product_except_self(nums2)}') # Output: [0, 0, 9, 0, 0]
28
```

→ Example 1: [24, 12, 8, 6]
Example 2: [0, 0, 9, 0, 0]

✓ 3.

Given an integer array nums sorted in non-decreasing order, remove some duplicates in-place such that each unique element appears at most twice. The relative order of the elements should be kept the same.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Do not allocate extra space for another array. You must do this by modifying the input array in-place with $O(1)$ extra memory.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length
```

```
int k = removeDuplicates(nums); // Calls your implementation
```

assert k == expectedNums.length; for (int i = 0; i < k; i++) { assert nums[i] == expectedNums[i]; } If all assertions pass, then your solution will be accepted.

Example 1:

Input: nums = [1,1,1,2,2,3] Output: 5, nums = [1,1,2,2,3,_] Explanation: Your function should return k = 5, with the first five elements of nums being 1, 1, 2, 2 and 3 respectively. It does not matter what you leave beyond the returned k (hence they are underscores). Example 2:


Input: nums = [0,0,1,1,1,2,3,3] Output: 7, nums = [0,0,1,1,2,3,3,_,_] Explanation: Your function should return k = 7, with the first seven elements of nums being 0, 0, 1, 1, 2, 3 and 3 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

1 <= nums.length <= 3 * 10⁴ - 10⁴ <= nums[i] <= 10⁴ nums is sorted in non-decreasing order.

<https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii/description/?envType=study-plan-v2&envId=top-interview-150>

```
1 def remove_duplicates(nums):
2     # Initialize the second pointer and the count
3     j = 1
4     count = 1
5
6     # Iterate through the array starting from the second element
7     for i in range(1, len(nums)):
8         # If the current element is the same as the previous element
9         if nums[i] == nums[i - 1]:
10             count += 1
11         else:
12             count = 1
13
14         # If the count is less than or equal to 2, place the element at index j
15         if count <= 2:
16             nums[j] = nums[i]
17             j += 1
18
19     return j
20
21 # Test cases
22 nums1 = [1, 1, 1, 2, 2, 3]
23 nums2 = [0, 0, 1, 1, 1, 1, 2, 3, 3]
24
25 # Expected outputs: 5 and 7
26 print(f'Example 1: {remove_duplicates(nums1)}, nums = {nums1[:remove_duplicates(nums1)]}')
27 print(f'Example 2: {remove_duplicates(nums2)}, nums = {nums2[:remove_duplicates(nums2)]}')
28
```

 Example 1: 5, nums = [1, 1, 2, 2, 3, 3]
Example 2: 7, nums = [0, 0, 1, 1, 2, 3, 3]

▼ 4.

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "{}"

Output: false

Example 4:

Input: s = "([)]"


Output: true

Constraints:

1 <= s.length <= 104 s consists of parentheses only '{}[]()'

<https://leetcode.com/problems/valid-parentheses/description/?envType=study-plan-v2&envId=top-interview-150>

```
1 def is_valid(s):
2     # Dictionary to hold mappings of closing brackets to opening brackets
3     bracket_map = {')': '(', '}': '{', ']': '['}
4     # Stack to keep track of opening brackets
5     stack = []
6
7     # Iterate through each character in the string
8     for char in s:
9         if char in bracket_map:
10             # Get the top element from the stack, or a dummy value if the stack is empty
11             top_element = stack.pop() if stack else '#'
12             # Check if the top element matches the corresponding opening bracket
13             if bracket_map[char] != top_element:
14                 return False
15         else:
16             # Push the opening bracket onto the stack
17             stack.append(char)
18
19     # Check if the stack is empty
20     return not stack
21
22 # Test cases
23 s1 = "()"
24 s2 = "()[]{}"
25 s3 = "(]"
26 s4 = "([)]"
27
28 # Expected outputs: True, True, False, True
29 print(f'Example 1: {is_valid(s1)}') # Output: True
30 print(f'Example 2: {is_valid(s2)}') # Output: True
31 print(f'Example 3: {is_valid(s3)}') # Output: False
32 print(f'Example 4: {is_valid(s4)}') # Output: True
33
```

 Example 1: True
Example 2: True
Example 3: False
Example 4: True

5.

Given an integer array nums and an integer k, return the kth largest element in the array.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

Input: nums = [3,2,1,5,6,4], k = 2 Output: 5 Example 2:

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4 Output: 4

Constraints:

1 <= k <= nums.length <= 105 -104 <= nums[i] <= 104


<https://leetcode.com/problems/kth-largest-element-in-an-array/description/?envType=study-plan-v2&envId=top-interview-150>

```
1 import heapq
2
3 def find_kth_largest(nums, k):
4     # Create a min-heap with the first k elements
```

```

5     heap = nums[:k]
6     heapq.heapify(heap)
7
8     # Iterate through the remaining elements
9     for num in nums[k:]:
10         if num > heap[0]:
11             heapq.heappushpop(heap, num)
12
13     # The root of the heap is the kth largest element
14     return heap[0]
15
16 # Test cases
17 nums1 = [3, 2, 1, 5, 6, 4]
18 k1 = 2
19
20 nums2 = [3, 2, 3, 1, 2, 4, 5, 5, 6]
21 k2 = 4
22
23 # Expected outputs: 5 and 4
24 print(f'Example 1: {find_kth_largest(nums1, k1)}') # Output: 5
25 print(f'Example 2: {find_kth_largest(nums2, k2)}') # Output: 4
26

```

 Example 1: 5
 Example 2: 4

✓ 6.

You are given two integer arrays `nums1` and `nums2` sorted in non-decreasing order and an integer `k`.

Define a pair (u, v) which consists of one element from the first array and one element from the second array.

Return the `k` pairs $(u_1, v_1), (u_2, v_2), \dots, (u_k, v_k)$ with the smallest sums.

Example 1:

Input: `nums1 = [1,7,11]`, `nums2 = [2,4,6]`, `k = 3` Output: `[[1,2],[1,4],[1,6]]` Explanation: The first 3 pairs are returned from the sequence: `[1,2],[1,4],[1,6],[7,2],[7,4],[7,6],[11,2],[11,4],[11,6]` Example 2:

Input: `nums1 = [1,1,2]`, `nums2 = [1,2,3]`, `k = 2` Output: `[[1,1],[1,1]]` Explanation: The first 2 pairs are returned from the sequence: `[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]`

Constraints:

`1 <= nums1.length, nums2.length <= 105` `109 <= nums1[i], nums2[i] <= 109` `nums1` and `nums2` both are sorted in non-decreasing order. `1 <= k <= 104` `k <= nums1.length * nums2.length`

<https://leetcode.com/problems/find-k-pairs-with-smallest-sums/description/?envType=study-plan-v2&envId=top-interview-150>

```


1 import heapq
2
3 def k_smallest_pairs(nums1, nums2, k):
4     # If either array is empty, return an empty list
5     if not nums1 or not nums2:
6         return []
7
8     # Initialize the min-heap
9     heap = []
10
11     # Initialize the result list
12     result = []
13
14     # Push the initial pairs (nums1[0], nums2[j]) for all j < min(k, len(nums2))
15     for j in range(min(k, len(nums2))):
16         heapq.heappush(heap, (nums1[0] + nums2[j], 0, j))
17
18     # Iterate until we find k pairs
19     while k > 0 and heap:
20         # Pop the smallest pair from the heap
21         current_sum, i, j = heapq.heappop(heap)
22         result.append([nums1[i], nums2[j]])
23
24         # Push the next pair (nums1[i + 1], nums2[j]) if i + 1 < len(nums1)
25         if i + 1 < len(nums1):
26             heapq.heappush(heap, (nums1[i + 1] + nums2[j], i + 1, j))
27
28         # Decrement k
29         k -= 1
30

```

```

31     return result
32
33 # Test cases
34 nums1_1 = [1, 7, 11]
35 nums2_1 = [2, 4, 6]
36 k1 = 3
37
38 nums1_2 = [1, 1, 2]
39 nums2_2 = [1, 2, 3]
40 k2 = 2
41
42 # Expected outputs: [[1, 2], [1, 4], [1, 6]] and [[1, 1], [1, 1]]
43 print(f'Example 1: {k_smallest_pairs(nums1_1, nums2_1, k1)}') # Output: [[1, 2], [1, 4], [1, 6]]
44 print(f'Example 2: {k_smallest_pairs(nums1_2, nums2_2, k2)}') # Output: [[1, 1], [1, 1]]
45

```

 Example 1: [[1, 2], [1, 4], [1, 6]]
 Example 2: [[1, 1], [1, 1]]

✓ 7.

Pandas :

Table: Customers

```

+-----+-----+
| Column Name | Type |
+-----+-----+
| id          | int  |
| name       | varchar |
+-----+-----+

```

id is the primary key (column with unique values) for this table.
 Each row of this table indicates the ID and name of a customer.

Table: Orders

```

+-----+-----+
| Column Name | Type |
+-----+-----+
| id          | int  |
| customerId  | int  |
+-----+-----+

```

id is the primary key (column with unique values) for this table.
 customerId is a foreign key (reference columns) of the ID from the Customers table.
 Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write a solution to find all customers who never order anything.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Customers table:

```

+----+-----+
| id | name |
+----+-----+
| 1  | Joe  |
| 2  | Henry |
| 3  | Sam  |
| 4  | Max  |
+----+-----+

```

Orders table:

```

+---+-----+
| id | customerId |
+---+-----+
| 1  | 3          |
| 2  | 1          |
+---+-----+
Output:
+-----+
| Customers |
+-----+
| Henry      |
| Max        |
+-----+

```

<https://leetcode.com/problems/customers-who-never-order/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>

```

1 import pandas as pd
2
3 # Example data for Customers table
4 customers_data = {'id': [1, 2, 3, 4], 'name': ['Joe', 'Henry', 'Sam', 'Max']}
5 customers_df = pd.DataFrame(customers_data)
6
7 # Example data for Orders table
8 orders_data = {'id': [1, 2], 'customerId': [3, 1]}
9 orders_df = pd.DataFrame(orders_data)
10
11 # Perform a left join between Customers and Orders
12 merged_df = pd.merge(customers_df, orders_df, left_on='id', right_on='customerId', how='left')
13
14 # Filter customers who never placed an order (customerId is NaN)
15 customers_never_ordered = merged_df[merged_df['customerId'].isna()][['name']]
16
17 # Display the result
18 print(customers_never_ordered)
19

```

```

1 import pandas as pd
2
3 # Sample data for Customers
4 customers_data = {
5     'id': [1, 2, 3, 4],
6     'name': ['Joe', 'Henry', 'Sam', 'Max']
7 }
8 customers = pd.DataFrame(customers_data)
9
10 # Sample data for Orders
11 orders_data = {
12     'id': [1, 2],
13     'customerId': [3, 1]
14 }
15 orders = pd.DataFrame(orders_data)
16
17 # Perform a left join on the 'id' column from Customers and 'customerId' column from Orders
18 merged_df = customers.merge(orders, left_on='id', right_on='customerId', how='left')
19
20 # Filter out customers who never ordered anything (where 'customerId' is NaN)
21 never_ordered_customers = merged_df[merged_df['customerId'].isna()][['name']]
22
23 # Convert the result to a DataFrame with the column name 'Customers'
24 result = never_ordered_customers.to_frame(name='Customers')
25
26 # Print the result
27 print(result)
28

```

```

↔ Customers
1    Henry
3      Max

```

Table: Employee

```
+-----+-----+
| Column Name | Type |
+-----+-----+
| id          | int  |
| salary      | int  |
+-----+-----+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the nth highest salary from the Employee table. If there is no nth highest salary, return null.

The result format is in the following example.

Example 1:

Input:

Employee table:

```
+-----+-----+
| id | salary |
+-----+-----+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+-----+-----+
```

n = 2

Output:

```
+-----+-----+
| getNthHighestSalary(2) |
+-----+-----+
| 200                     |
+-----+-----+
```

Example 2:

Input:

Employee table:

```
+-----+-----+
| id | salary |
+-----+-----+
| 1  | 100    |
+-----+-----+
```

n = 2

Output:

```
+-----+-----+
| getNthHighestSalary(2) |
+-----+-----+
| null                   |
+-----+-----+
```

<https://leetcode.com/problems/nth-highest-salary/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>

```
1 import pandas as pd
2
3 # Example data for Employee table
4 employee_data = {'id': [1, 2, 3], 'salary': [100, 200, 300]}
5 employee_df = pd.DataFrame(employee_data)
6
7 def getNthHighestSalary(employee_df, n):
8     # Sort salaries in descending order and remove duplicates
9     unique_salaries = employee_df['salary'].drop_duplicates().sort_values(ascending=False)
10
11     # Check if n is greater than the number of unique salaries
12     if len(unique_salaries) >= n:
13         return unique_salaries.iloc[n-1]
14     else:
15         return None
```



```

16
17 # Example usage
18 n = 2
19 result = getNthHighestSalary(employee_df, n)
20 print(f"The {n}th highest salary is: {result}")
21

```

🔄 The 2th highest salary is: 200

✓ 9.

```

+-----+-----+
| Column Name | Type   |
+-----+-----+
| id           | int    |
| name         | varchar|
| salary       | int    |
| departmentId | int    |
+-----+-----+

```

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference columns) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

```

+-----+-----+
| Column Name | Type   |
+-----+-----+
| id           | int    |
| name         | varchar|
+-----+-----+

```

id is the primary key (column with unique values) for this table. It is guaranteed that department name is not NULL.

Each row of this table indicates the ID of a department and its name.

Write a solution to find employees who have the highest salary in each of the departments.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Employee table:

```

+-----+-----+-----+-----+
| id | name | salary | departmentId |
+-----+-----+-----+-----+
| 1  | Joe  | 70000  | 1             |
| 2  | Jim  | 90000  | 1             |
| 3  | Henry| 80000  | 2             |
| 4  | Sam  | 60000  | 2             |
| 5  | Max  | 90000  | 1             |
+-----+-----+-----+-----+

```

Department table:

```

+-----+-----+
| id | name |
+-----+-----+
| 1  | IT   |
| 2  | Sales|
+-----+-----+

```

Output:

```

+-----+-----+-----+
| Department | Employee | Salary |
+-----+-----+-----+

```

```
+-----+-----+-----+
| IT      | Jim    | 90000 |
| Sales   | Henry  | 80000 |
| IT      | Max    | 90000 |
+-----+-----+-----+
```

Explanation: Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.

<https://leetcode.com/problems/department-highest-salary/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>

```
1 import pandas as pd
2
3 # Sample data for Employee
4 employee_data = {
5     'id': [1, 2, 3, 4, 5],
6     'name': ['Joe', 'Jim', 'Henry', 'Sam', 'Max'],
7     'salary': [70000, 90000, 80000, 60000, 90000],
8     'departmentId': [1, 1, 2, 2, 1]
9 }
10 employee = pd.DataFrame(employee_data)
11
12 # Sample data for Department
13 department_data = {
14     'id': [1, 2],
15     'name': ['IT', 'Sales']
16 }
17 department = pd.DataFrame(department_data)
18
19 # Merge the Employee and Department data
20 merged_df = employee.merge(department, left_on='departmentId', right_on='id', suffixes=('_employee', '_department'))
21
22 # Find the maximum salary in each department
23 max_salary_df = merged_df.groupby('departmentId')['salary'].max().reset_index()
24
25 # Merge the maximum salaries with the merged dataframe to get the corresponding employees
26 result_df = merged_df.merge(max_salary_df, on=['departmentId', 'salary'])
27
28 # Select the required columns
29 result = result_df[['name_department', 'name_employee', 'salary']]
30 result.columns = ['Department', 'Employee', 'Salary']
31
32 # Print the result
33 print(result)
34
```

```
↗
  Department Employee Salary
0          IT      Jim    90000
1        Sales   Henry    80000
2          IT      Max    90000
```

✓ 10.

```
+-----+-----+
| Column Name | Type   |
+-----+-----+
| id           | int    |
| score        | decimal|
+-----+-----+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains the score of a game. Score is a floating point value with two decimal places.

Write a solution to find the rank of the scores. The ranking should be calculated according to the following rules:

The scores should be ranked from the highest to the lowest.

If there is a tie between two scores, both should have the same ranking.

After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by score in descending order.

The result format is in the following example.

Example 1:

Input:

Scores table:

```
+-----+
| id | score |
+-----+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+-----+
```

Output:

```
+-----+
| score | rank |
+-----+
| 4.00  | 1    |
| 4.00  | 1    |
| 3.85  | 2    |
| 3.65  | 3    |
| 3.65  | 3    |
| 3.50  | 4    |
+-----+
```

<https://leetcode.com/problems/rank-scores/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>

```
1 import pandas as pd
2
3 # Sample data for Scores
4 scores_data = {
5     'id': [1, 2, 3, 4, 5, 6],
6     'score': [3.50, 3.65, 4.00, 3.85, 4.00, 3.65]
7 }
8 scores = pd.DataFrame(scores_data)
9
10 # Sort the DataFrame by score in descending order
11 scores = scores.sort_values(by='score', ascending=False)
12
13 # Assign ranks to the scores
14 scores['rank'] = scores['score'].rank(method='dense', ascending=False).astype(int)
15
16 # Select the required columns
17 result = scores[['score', 'rank']]
18
19 # Print the result
20 print(result)
21
```

```
↗
  score rank
2  4.00    1
4  4.00    1
3  3.85    2
1  3.65    3
5  3.65    3
0  3.50    4
```

✓ SQL Questions Turing

Q1. nth Highest Salary.

```
CREATE TABLE Employee (
  id INT PRIMARY KEY,
```

```

        salary INT
    );

INSERT INTO Employee (id, salary) VALUES (1, 100), (2, 200), (3, 300);

CREATE FUNCTION getNthHighestSalary(n INT) RETURNS INT
BEGIN
    RETURN (
        SELECT DISTINCT salary
        FROM Employee
        ORDER BY salary DESC
        LIMIT 1 OFFSET n-1
    );
END;

-- Example 1
SELECT getNthHighestSalary(2); -- Output: 200

-- Example 2
DELETE FROM Employee WHERE id IN (2, 3);
SELECT getNthHighestSalary(2); -- Output: null

```

Q.2. Given an SQL table with columns `id` and `item`, find the items that appear consecutively three times in a row, with the `id` values being in incremental order by one.

Example table:

id	item
1	1
2	1
3	1
4	2
5	2
6	3
7	3
8	4
9	4
10	5

Expected answer: 1

ANSWER:

```

SELECT DISTINCT t1.item
FROM table_name t1
JOIN table_name t2 ON t1.id = t2.id - 1
JOIN table_name t3 ON t1.id = t3.id - 2
WHERE t1.item = t2.item AND t2.item = t3.item
ORDER BY t1.item;

```

✓ Q.3. Customer who bought all products

```

1
2 #Code:-
3
4 import pandas as pd
5
6 def find_customers(customer: pd.DataFrame, product: pd.DataFrame) -> pd.DataFrame:
7     # Merge Customer and Product tables
8     merged_df = pd.merge(customer, product, on='product_key')
9
10    # Group by customer_id and count distinct product_key
11    grouped_df = merged_df.groupby('customer_id')['product_key'].nunique()
12
13    # Filter customers who bought all products
14    all_products_customers = grouped_df[grouped_df == len(product)]
15

```

```

16 # Return DataFrame with customer_ids who bought all products
17 return pd.DataFrame(all_products_customers.index, columns=['customer_id'])
18

```

✓ Q.4. Top three Salaries

```

1 #Code:-
2
3 import pandas as pd
4
5 def top_three_salaries(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:
6     # Merge Employee and Department tables
7     merged_df = pd.merge(employee, department, left_on='departmentId', right_on='id')
8
9     # Group by department and find top three salaries
10    top_salaries_df = merged_df.groupby('name_y')['salary'].apply(lambda x: x.nlargest(3)).reset_index()
11
12    # Rename columns
13    top_salaries_df.columns = ['Department', 'Employee', 'Salary']
14
15    # Return DataFrame
16    return top_salaries_df
17
18 # Example usage:
19 employee_data = {
20     'id': [1, 2, 3, 4, 5, 6, 7],
21     'name': ['Joe', 'Henry', 'Sam', 'Max', 'Janet', 'Randy', 'Will'],
22     'salary': [85000, 80000, 60000, 90000, 69000, 85000, 70000],
23     'departmentId': [1, 2, 2, 1, 1, 1, 1]
24 }
25
26 department_data = {
27     'id': [1, 2],
28     'name': ['IT', 'Sales']
29 }
30
31 employee_df = pd.DataFrame(employee_data)
32 department_df = pd.DataFrame(department_data)
33
34 result_df = top_three_salaries(employee_df, department_df)
35 print(result_df)
36

```

	Department	Employee	Salary
0	IT	3	90000
1	IT	0	85000
2	IT	5	85000
3	Sales	1	80000
4	Sales	2	60000

✓ Q.5. Second Highest Salary

```

1
2 #Code:-
3
4 import pandas as pd
5
6 def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
7     # Sort salaries in descending order and drop duplicates
8     sorted_salaries = employee['salary'].sort_values(ascending=False).drop_duplicates()
9
10    # If there's no second highest salary, return None
11    if len(sorted_salaries) < 2:
12        return pd.DataFrame({"SecondHighestSalary": [None]})
13
14    # Get the second highest salary
15    second_highest_salary = sorted_salaries.iloc[1]
16
17    # Return DataFrame with second highest salary
18    return pd.DataFrame({"SecondHighestSalary": [second_highest_salary]})
19
20 # Example usage:
21 employee_data = {
22     'id': [1, 2, 3],
23     'salary': [100, 200, 300]
24 }
25
26 employee_df = pd.DataFrame(employee_data)

```

```

27
28 result_df = second_highest_salary(employee_df)
29 print(result_df)

```

```

↗ SecondHighestSalary
  0                200

```

✓ Q.6. Shortest Palindrome

```

1
2
3 #Code:-
4
5 class Solution(object):
6     def shortestPalindrome(self, s):
7         """
8         :type s: str
9         :rtype: str
10        """
11        # Step 1: Reverse the string s
12        rev_s = s[::-1]
13
14        # Step 2: Concatenate s with its reverse using a special character in between
15        new_s = s + '#' + rev_s
16
17        # Step 3: Compute the KMP table (LPS array) for new_s
18        n = len(new_s)
19        lps = [0] * n
20        j = 0
21        for i in range(1, n):
22            while (j > 0 and new_s[i] != new_s[j]):
23                j = lps[j - 1]
24            if new_s[i] == new_s[j]:
25                j += 1
26            lps[i] = j
27
28        # Step 4: Length of the longest palindromic prefix in s
29        longest_palindromic_prefix_len = lps[-1]
30
31        # Step 5: Characters to add in front of s
32        chars_to_add = rev_s[:len(s) - longest_palindromic_prefix_len]
33
34        # Step 6: Return the shortest palindrome
35        return chars_to_add + s
36
37 # Example usage
38 solution = Solution()
39 print(solution.shortestPalindrome("aacecaaa")) # Output: "aaacecaaa"
40 print(solution.shortestPalindrome("abcd"))    # Output: "dcbabcd"
41

```

```

↗ aaacecaaa
  dcbabcd

```

```

=====
#####
#####

```

+-----+-----+		
Column Name	Type	
+-----+-----+		
id	int	
name	varchar	
department	varchar	
managerId	int	
+-----+-----+		

id is the primary key (column with unique values) for this table.

Each row of this table indicates the name of an employee, their department, and the id of their manager.

If managerId is null, then the employee does not have a manager.

No employee will be the manager of themselves.

Write a solution to find managers with at least five direct reports.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input:

Employee table:

id	name	department	managerId
101	John	A	null
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

Output:

name
John

```
1 import pandas as pd
2
3 def find_managers(employee: pd.DataFrame) -> pd.DataFrame:
4
5     df = employee.groupby('managerId'
6                           ).size().reset_index(name='idx')
7
8     return df[df.idx >= 5].merge(employee,
9                                   left_on='managerId',
10                                  right_on='id',how='inner').iloc[:,[3]]
11
12
13
```

Approach and Explanation

Here's a detailed breakdown of the approach used in the `find_managers` function:

- Group by `managerId` and Count Direct Reports:
 - `df = employee.groupby('managerId').size().reset_index(name='idx')`
 - Grouping: The data is grouped by `managerId`, which essentially means we are aggregating the number of direct reports for each manager.
 - Counting: The `size()` function counts the number of rows for each group, representing the number of direct reports each manager has.
 - Resetting Index: `reset_index(name='idx')` converts the result into a DataFrame with a column named `idx` representing the count of direct reports.
- Filter Managers with at Least Five Direct Reports:
 - `df[df.idx >= 5]`
 - This step filters the DataFrame to include only those managers who have at least five direct reports. The condition `df.idx >= 5` is used to select rows where the count of direct reports is greater than or equal to 5.
- Merge to Get Manager Names:
 - `.merge(employee, left_on='managerId', right_on='id', how='inner')`
 - Merge Operation: This merges the filtered DataFrame (managers with at least five direct reports) with the original employee DataFrame to get the names of the managers.
 - Join Conditions:
 - `left_on='managerId'`: Specifies that `managerId` from the filtered DataFrame should match the `id` from the employee DataFrame.
 - `right_on='id'`: Specifies that the `id` column in the employee DataFrame should match the `managerId` from the filtered DataFrame.
 - Type of Join: `how='inner'` ensures that only matching rows (managers with their names) are included.
- Select and Return Manager Names:
 - `.iloc[:,[3]]`
 - Selecting Column: `iloc[:,[3]]` selects the column at index 3, which is assumed to be the name column of the managers from the merged DataFrame.
 - Returning Result: This returns the final DataFrame containing only the names of the managers who have at least five direct reports.

Summary

- Group the employees by `managerId` to count the number of direct reports each manager has.
- Filter the managers who have five or more direct reports.

- Merge the filtered list with the original employee data to get the names of these managers.
- Select and return the relevant column (names of the managers).

Double-click (or enter) to edit

Given an array of integers `temperatures` represents the daily temperatures, return an array `answer` such that `answer[i]` is the number of days you have to wait after the `i`th day to get a warmer temperature. If there is no such day, use `0` in the answer.

Example 1:

Input: `temperatures = [73,74,75,71,69,72,76,73]`

Output: `[1,1,4,2,1,1,0,0]`

Example 2:

Input: `temperatures = [30,40,50,60]`

Output: `[1,1,1,0]`

Example 3:

Input: `temperatures = [30,60,90]`

Output: `[1,1,0]`

Constraints:

`1 <= temperatures.length <= 105`

`30 <= temperatures[i] <= 100`

<https://leetcode.com/problems/daily-temperatures/description/?envType=study-plan-v2&envId=leetcode-75>

Thoery :

1. Types of data and how to handle each type of data
2. what are outliers ?, what are the different methods to detect outliers (statistical and graphical both) ?, what are the different methods to handle outlier ?
3. what are the different methods to detect missing values , how to handle missing values ? , what are the different methods to handle outlier ?
4. what is cost function ? , what is loss function? , what are different cost functions ? which cost function is used in which case ?
5.
 - All the ML algorithms
6. what is overfitting and underfitting? , how to handle them? , what is Bias & VAriance? , what is bias variance tradeoff?
7.
 - Pandas and Numpy Fuctions
8. What are Unstack, idx, idxmax, idxmin function and why it is used and what output it gives
9. what are vectors? , how to create them ? , what is vectorization ?
10. Data preprocessing: Cleaning and handling