50 PYTHON EXERCISES

NUMPY
PANDAS

Daniel Christofis

50 PYTHON EXERCISES NUMPY & PANDAS

A Practical Guide - Mastering DataFrames, Data Manipulation and Creating Graphs

Daniel Christofis

CONTENTS

A QUICK NOTE

We hope you are enjoying your recent purchase from Amazon! We wanted to take a moment to kindly remind you about the importance of leaving a review for the product you bought. Your feedback is incredibly valuable to us and other potential buyers.

By sharing your thoughts and experiences with the product, you help others make informed decisions. Your review can be a guiding light for fellow customers, ensuring they choose the best products for their needs.

We would greatly appreciate it if you could spare a few moments to write an honest review. Your feedback helps us improve our products and services, making it a win-win situation for everyone.

Thank you for being a valued customer and for considering writing a review. We truly appreciate your support.

Introducing my personal blog. Stay connected!

https://premiumbookself.wordpress.com/

Premium Bookshelf


HOW TO LEARN
PYTHON IN
3 DAYS
MORE THAN 200
EXAMPLES & EXERCISES
DANIEL CHRISTOFIS


50
PYTHON
EXERCISES
MACHINE LEARNING
NEURAL NETWORKS
Daniel Christofis


50
PYTHON
EXERCISES
BEGINNER
INTERMEDIATE
Daniel Christofis

INTRODUCTION

NumPy and Pandas emerge as the champions of the Python data environment in the enormous area of data research. These libraries allow not only professionals to easily manage and alter data, but also newcomers to dig deep into the field of data analysis with ease and intuitively. "50 Python Exercises: NumPy and Pandas" has been carefully designed to give you a taste of real-world issues, allowing you to flex your problem-solving muscles and hone your coding skills.

This book provides a wide range of activities organised into eight projects. Each project focuses on a distinct topic, bringing you from the fundamentals of data manipulation to the depths of feature engineering.

Project 1: Introduction to Pandas and NumPy Basics

Grasp the foundations of data handling. Dive into tasks ranging from loading datasets to basic computations, setting the stage for more advanced projects.

Project 2: Data Filtering & Sorting

Learn how to sift through data and present it in a way that makes sense. This section emphasizes the importance of structuring and organizing data for better analysis.

Project 3: Data Grouping & Aggregation

Delve deeper into the data, dissecting it into meaningful groups and drawing aggregated insights that can drive business decisions or power data models.

Project 4: Data Cleaning & Exploration

Every data scientist knows that raw data is often messy. Here, we take on the vital tasks of cleaning, transforming, and exploring datasets to prepare them for further analysis or modeling.

Project 5: Merging, Joining & Concatenation

Data doesn't always come in a single, neat package. Discover how to combine datasets, be it stacking them up or joining them based on common attributes.

Project 6: Visualization & Descriptive Statistics

Pictures often speak louder than numbers. Dive into the art and science of visualizing data, alongside wielding descriptive statistics to capture the essence of your dataset.

Project 7: Advanced Pandas Functions & Techniques

Step into the realm of advanced data manipulation. This section delves into some of the more intricate functionalities that Pandas offers, which can immensely streamline your data processing workflow.

Project 8: Feature Engineering & Selection

In the world of machine learning and predictive modeling, the features you provide to your model are paramount. Master the art of crafting, selecting, and transforming features to boost the performance of subsequent models.

INSTALLING REQUIRED LIBRARIES

Before diving into the exercises, it's essential to ensure that you have all the required libraries installed. This book primarily leverages the power of NumPy and Pandas, along with a few additional libraries for specific tasks.

To install the necessary libraries, you can use Python's package manager, pip. Simply open your terminal or command prompt and enter the following command:

```
pip install numpy pandas seaborn matplotlib scikit-learn statsmodels
```

Once the installation is complete, you're all set to embark on the exciting journey of "50 Python Exercises: NumPy and Pandas". **Happy coding!**

PROJECT 1: INTRODUCTION TO PANDAS AND NUMPY BASICS

Import the 'NumPy' and 'Pandas' libraries, and the 'Titanic' dataset from seaborn.

```
import seaborn as sns
import numpy as np
import pandas as pd
```

```
titanic = sns.load_dataset('titanic')
```

1-1. Load the Titanic dataset using Seaborn and display the first 5 rows.

1-2. Display the last 5 rows of the dataset.

1-3. Get the data types of each column.

1-4. Convert the 'sex' column to a category type.

1-5. Compute the sum of the 'fare' column.

1-6. Get the mean and standard deviation of the 'age' column.

1-7. Create a NumPy array from the 'age' column of the Titanic DataFrame.

PROJECT 2: DATA FILTERING & SORTING

Import the 'NumPy' and 'Pandas' libraries, and the 'Titanic' dataset from seaborn**.**

```
import seaborn as sns
import numpy as np
import pandas as pd
```

```
titanic = sns.load_dataset('titanic')
```

2-1. Filter out rows where the 'age' column's value is missing.

2-2. Retrieve rows where the age is greater than 50.

2-3. Filter data based on multiple conditions (e.g., age > 30 and 'sex' is 'female').

2-4. Sort the dataset based on the 'fare' column in ascending order.

2-5. Sort the dataset by multiple columns (e.g., first by 'class' and then by 'age').

2-6. Reset the index of the DataFrame after sorting.

2-7. Find and display rows where 'embarked' column value is 'C' (Cherbourg) and 'fare' is within the 90th percentile.

PROJECT 3: DATA GROUPING & AGGREGATION

Import the 'Pandas' library, and the 'iris' dataset from seaborn.

```
import pandas as pd
import seaborn as sns
```

```
iris = sns.load_dataset('iris')
```

3-1: Display the first 5 rows of the dataset.

3-2: Get a description (count, mean, std, min, max) of each numeric column in the dataset

3-3: Group the dataset by the 'species' column and get the mean value of each column for each species.

3-4: Which species has the highest average sepal length?

3-5: Filter the dataset for the 'setosa' species. How many unique values are there in the 'petal_width' column for this species, and what is the count for each unique value?

3-6: Identify the species with the maximum petal width. Additionally, find its corresponding average sepal width.

3-7: Create a pivot table with species as rows. The table should display the average petal length and sepal length for each species.

PROJECT 4: DATA CLEANING & EXPLORATION

Dataset:

```
import seaborn as sns
import pandas as pd

# Loading the mpg dataset from seaborn
mpg = sns.load_dataset('mpg')
```

4-1: Display the last 7 rows of the dataset.

4-2: How many missing values are there in the 'horsepower' column? Replace them with the median value of that column.

4-3: Filter the dataset for cars with an 'origin' from 'usa' and 'cylinders' equal to 4. How many such cars are there in the dataset?

4-4: Identify the make and model of the car with the highest 'mpg'. What is its mpg?

4-5: Create a new column 'displacement_to_power' as the ratio of 'displacement' to 'horsepower'. Which car make and model has the highest 'displacement_to_power' ratio?

4-6: Group the dataset by 'origin' and calculate the average 'mpg' for each group. Which 'origin' has the highest average mpg?

4-7: Filter cars with an 'mpg' lower than the 10th percentile. How many cars remain in the dataset post this operation?

PROJECT 5: MERGING, JOINING & CONCATENATION

Import both datasets:

```python
import seaborn as sns
import pandas as pd

# Loading the datasets
tips = sns.load_dataset('tips')
flights = sns.load_dataset('flights')
```

5-1: Concatenate 'tips' and 'flights' DataFrames vertically (keep in mind the datasets are different; this is more for practice than utility).

5-2: Merge 'tips' and a DataFrame that has an additional column, say "discount", based on a common column "total_bill".

5-3: Perform a left join between the 'tips' DataFrame and another DataFrame containing "total_bill" and a new column "feedback".

5-4: Perform an outer join between 'tips' and a DataFrame with columns "total_bill" and "restaurant_review".

5-5: Concatenate 'tips' and 'flights' DataFrames horizontally.

5-6: Handle overlapping columns when merging two DataFrames: 'tips' and a DataFrame containing "total_bill", "tip", and "dining_experience".

5-7: Merge 'tips' and a DataFrame using a multi-index based on columns "total_bill" and "tip".

PROJECT 6: VISUALIZATION & DESCRIPTIVE STATISTICS

Dataset and extra libraries used:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
tips = sns.load_dataset('tips')
```

6-1. Generate a histogram for the 'total_bill' column using Pandas.

6-2. Compute and display the correlation matrix for the entire tips dataset.

6-3. Generate a scatter plot between the 'total_bill' and 'tip' columns using Pandas.

6-4. Calculate and display the skewness and kurtosis for the 'total_bill' column.

6-5. Use the value_counts method to display the frequency of unique values in the 'day' column. Subsequently, visualize it as a bar chart.

6-6. Generate a box plot for the 'total_bill' column to visualize outliers.

6-7. Compute the IQR (Interquartile Range) for the 'total_bill' column.

PROJECT 7: ADVANCED PANDAS FUNCTIONS & TECHNIQUES

Dataset:

```
import seaborn as sns
import numpy as np
import pandas as pd
```

```
diamonds = sns.load_dataset('diamonds')
```

7-1. Use the query method to filter rows from the diamonds dataset where the cut is "Ideal" and the price is greater than $500.

7-2. Extract rows from the diamonds dataset where the color column matches the regex pattern "^[EF]$".

7-3. Set the 'color' column of the diamonds dataset as the index, and then reset it back to a column.

7-4. Use the eval function to compute the price divided by carat and store the result in a new column named "price_per_carat".

7-5. Create a new column named 'high_price' in the diamonds dataset, which should be True if the price is greater than $1000 and False otherwise. Use the np.where function for this.

7-6. Find and drop columns from the diamonds dataset with a variance below a threshold of 1.

7-7. Use the melt function to reshape the diamonds dataset, with 'color' and 'cut' as ID variables.

PROJECT 8: FEATURE ENGINEERING & SELECTION PROJECT
(Optional)

Dataset and extra libraries used:

```
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from statsmodels.stats.outliers_influence import
variance_inflation_factor

titanic = sns.load_dataset('titanic')
```

8-1. Based on existing columns in the Titanic dataset, namely 'sibsp' which denotes the number of siblings/spouses aboard, and 'parch' which signifies the number of parents/children aboard, can you create a new column called 'family_size' to represent the total family size of each passenger on board?

8-2. The 'class' column in the Titanic dataset refers to the passenger class and is a categorical column. Can you transform this column using one-hot encoding, ensuring that you drop the first category to avoid multicollinearity?

8-3. Age is an essential continuous variable in the Titanic dataset. Can you normalize the 'age' column using the z-score normalization technique? This helps to express ages in terms of their deviation from the mean age.

8-4. Dimensionality reduction can help in reducing noise and improving computational efficiency. Using PCA (Principal Component Analysis), can you reduce the dimensions of the dataset? For this task, consider only the 'age' and 'fare' columns.

8-5. Multicollinearity can be an issue when building predictive models. For the Titanic dataset, can you compute the Variance Inflation Factor (VIF) for the 'age', 'fare', and the one-hot encoded 'class' columns? Features with a VIF greater than 10 are typically considered to have high multicollinearity.

8-6. For better interpretability and categorization, can you bin the 'age' column of the Titanic dataset into custom groups: 'child' (0-18 years), 'young_adult' (18-35 years), 'adult' (35-60 years), and 'senior' (60+ years)?

8-7. Target mean encoding, also known as likelihood encoding, is a way to encode categorical columns based on the mean of the target variable. Using this technique, can you encode the 'embark_town' column in the Titanic dataset with respect to the 'survived' column?

PROJECT 1 SOLUTIONS

```python
import seaborn as sns
import numpy as np
import pandas as pd

# Load the dataset
titanic = sns.load_dataset('titanic')

# 1-1. Load the Titanic dataset using Seaborn and display the first 5 rows.
print("#1-1:\n", titanic.head())

# 1-2. Display the last 5 rows of the dataset.
print("#1-2:\n", titanic.tail())

# 1-3. Get the data types of each column.
print("#1-3:\n", titanic.dtypes)

# 1-4. Convert the 'sex' column to a category type.
titanic['sex'] = titanic['sex'].astype('category')
print("#1-4:\n", titanic['sex'].dtypes)

# 1-5. Compute the sum of the 'fare' column.
fare_sum = titanic['fare'].sum()
print("#1-5: Sum of Fare Column:", fare_sum)

# 1-6. Get the mean and standard deviation of the 'age' column.
age_mean = titanic['age'].mean()
age_std = titanic['age'].std()
print("#1-6:\nMean Age:", age_mean, "\nStandard Deviation of Age:",
age_std)

# 1-7. Create a NumPy array from the 'age' column of the Titanic
DataFrame.
age_array = titanic['age'].to_numpy()
print("#1-7:\n", age_array)
```

Project 1 Output

#1-1:
```
   survived  pclass     sex   age  sibsp  parch ...
0         0       3    male  22.0      1      0
1         1       1  female  38.0      1      0
2         1       3  female  26.0      0      0
3         1       1  female  35.0      1      0
4         0       3    male  35.0      0      0
```

#1-2:
```
     survived  pclass     sex   age  sibsp  parch ...
886         0       2    male  27.0      0      0
887         1       1  female  19.0      0      0
888         0       3  female   NaN      1      2
889         1       1    male  26.0      0      0
890         0       3    male  32.0      0      0
```

#1-3:
```
survived         int64
pclass           int64
sex             object
age            float64
sibsp            int64
parch            int64
fare           float64
embarked        object
class         category
who             object
adult_male        bool
deck          category
embark_town     object
alive           object
alone             bool
dtype: object
```

#1-4:

category

#1-5: Sum of Fare Column: 28693.9493

#1-6:
Mean Age: 29.69911764705882
Standard Deviation of Age: 14.526497332334044

#1-7:
[22. 38. 26. ... nan 26. 32.]

PROJECT 2 SOLUTIONS

```python
import seaborn as sns
import numpy as np
import pandas as pd

titanic = sns.load_dataset('titanic')

# 2-1. Filter out rows where the 'age' column's value is missing.
age_not_missing = titanic.dropna(subset=['age'])
print("#2-1", age_not_missing.head())

# 2-2. Retrieve rows where the age is greater than 50.
age_above_50 = titanic[titanic['age'] > 50]
print("#2-2", age_above_50.head())

# 2-3. Filter data based on multiple conditions (e.g., age > 30 and 'sex' is
'female').
females_above_30 = titanic[(titanic['age'] > 30) & (titanic['sex'] ==
'female')]
print("#2-3", females_above_30.head())

# 2-4. Sort the dataset based on the 'fare' column in ascending order.
sorted_by_fare = titanic.sort_values(by='fare')
print("#2-4", sorted_by_fare.head())

# 2-5. Sort the dataset by multiple columns (e.g., first by 'class' and then by
'age').
sorted_by_class_age = titanic.sort_values(by=['class', 'age'])
print("#2-5", sorted_by_class_age.head())

# 2-6. Reset the index of the DataFrame after sorting.
reset_index_df = sorted_by_class_age.reset_index(drop=True)
print("#2-6", reset_index_df.head())
```

```
# 2-7. Find and display rows where 'embarked' column value is
'C'  (Cherbourg) and 'fare' is within the 90th percentile.
cherbourg_high_fare = titanic[(titanic['embarked'] == 'C') &
(titanic['fare'] > titanic['fare'].quantile(0.90))]
print("#2-7", cherbourg_high_fare)
```

Project 2 Output

```
#2-1:
   survived  pclass     sex   age  sibsp  parch ...
0         0       3    male  22.0      1      0
1         1       1  female  38.0      1      0
2         1       3  female  26.0      0      0
3         1       1  female  35.0      1      0
4         0       3    male  35.0      0      0

#2-2:
    survived  pclass     sex   age  sibsp  parch ...
6          0       1    male  54.0      0      0
11         1       1  female  58.0      0      0
15         1       2  female  55.0      0      0

#2-3:
    survived  pclass     sex   age  sibsp  parch     fare ...
1          1       1  female  38.0      1      0  71.2833
3          1       1  female  35.0      1      0  53.1000
11         1       1  female  58.0      0      0  26.5500
#2-4:
     survived  pclass   sex  age  sibsp  parch  fare ...
179         0       3  male  NaN      0      0   0.0
263         0       1  male  NaN      0      0   0.0
302         0       3  male  NaN      0      0   0.0

#2-5:
   survived  pclass     sex      age ...   class    who ...
```

```
20       0       2    male  35.0           Second    man
21       1       2    male  34.0           Second    man
33       0       2    male  66.0           Second    man
```

#2-6:

```
    survived  pclass     sex   age  sibsp  parch  ...
0        0       2    male  35.0      0      0
1        1       2    male  34.0      0      0
2        0       2    male  66.0      0      0
```

#2-7:

```
    survived...  fare embarked ... deck ...  embark_town alive  alone
31       1       146.5208        C      Cherbourg   yes  False
195      1       146.5208        C      Cherbourg   yes   True
305      1       151.5500        C      Cherbourg   yes  False
```

PROJECT 3 SOLUTIONS

```
import pandas as pd
import seaborn as sns

iris = sns.load_dataset('iris')

# 3-1: Display the first 5 rows of the dataset.
print(iris.head())

# 3-2: Get a description (count, mean, std, min, max) of each numeric
column.
print(iris.describe())

# 3-3: Group the dataset by 'species' and get the mean value of each column
per species.
grouped_by_species = iris.groupby('species').mean()
print(grouped_by_species)

# 3-4: Find the species that has the highest average sepal length.
max_sepal_length_species =
grouped_by_species['sepal_length'].idxmax()
print("Species with the highest average sepal length:",
max_sepal_length_species)

# 3-5: Filter the data for setosa species and get the count of each unique
value in the 'petal_width' column.
setosa_petal_width_count =
 iris[iris['species'] == 'setosa']['petal_width'].value_counts()
print(setosa_petal_width_count)

# 3-6: Using aggregation, find the species with the maximum petal width
and its corresponding average sepal width.
agg_data = iris.groupby('species').agg({'petal_width': 'max',
 'sepal_width': 'mean'})
```

```
species_max_petal_width = agg_data['petal_width'].idxmax()
avg_sepal_width = agg_data.loc[species_max_petal_width,
 'sepal_width']
print(f"Species with maximum petal width:
 {species_max_petal_width}, with avg sepal width:
 {avg_sepal_width}")

# 3-7: Create a pivot table with species as rows, and average petal length
and sepal length as values.
pivot_table = iris.pivot_table(index='species', values=['petal_length',
'sepal_length'], aggfunc='mean')
print(pivot_table)
```

Project 3 Output

#3-1:

```
   sepal_length  sepal_width  petal_length  petal_width species
0          5.1          3.5           1.4          0.2  setosa
1          4.9          3.0           1.4          0.2  setosa
2          4.7          3.2           1.3          0.2  setosa
3          4.6          3.1           1.5          0.2  setosa
4          5.0          3.6           1.4          0.2  setosa
```

#3-2:

```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
...
```

(Note: I've abbreviated the describe output for brevity.)

#3-3:

```
            sepal_length  sepal_width  petal_length  petal_width
species
```

setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

#3-4:
Species with the highest average sepal length: virginica

#3-5:
0.2   28
0.4   7
0.3   7
0.1   6
0.5   1
0.6   1
Name: petal_width, dtype: int64

#3-6:
Species with maximum petal width: virginica, with avg sepal width: 2.974

#3-7:

	petal_length	sepal_length
species		
setosa	1.462	5.006
versicolor	4.260	5.936
virginica	5.552	6.588

PROJECT 4 SOLUTIONS

```
import seaborn as sns
import pandas as pd

# Loading the mpg dataset from seaborn
mpg = sns.load_dataset('mpg')

# 4-1: Display the last 7 rows of the dataset.
print("#4-1:")
print(mpg.tail(7))

# 4-2: Find missing values in 'horsepower' and replace with median.
missing_values_count = mpg['horsepower'].isna().sum()
mpg['horsepower'].fillna(mpg['horsepower'].median(), inplace=True)
print("\n#4-2:")
print(f"Number of missing values replaced in 'horsepower':
 {missing_values_count}")

# 4-3: Filter the data for 'usa' origin and 4 cylinders.
filtered_cars = mpg[(mpg['origin'] == 'usa') & (mpg['cylinders'] == 4)]
print("\n#4-3:")
print(f"Number of cars from 'usa' with 4 cylinders:
 {len(filtered_cars)}")

# 4-4: Identify the car with the highest mpg.
highest_mpg_row = mpg[mpg['mpg'] == mpg['mpg'].max()]
print("\n#4-4:")
print(f"""Car with the highest mpg:
 {highest_mpg_row['name'].values[0]} - MPG:
 {highest_mpg_row['mpg'].values[0]}""")

# 4-5: Create 'displacement_to_power' and find the car with the highest
ratio.
mpg['displacement_to_power'] = mpg['displacement'] /
```

```
 mpg['horsepower']
highest_ratio_row = mpg[mpg['displacement_to_power'] ==
mpg['displacement_to_power'].max()]
print("#4-5:")
print(f"Car with the highest 'displacement_to_power' ratio:
{highest_ratio_row['name'].values[0]} - Ratio:
{highest_ratio_row['displacement_to_power'].values[0]}")

# 4-6: Group by 'origin' and calculate average mpg.
avg_mpg_by_origin = mpg.groupby('origin')['mpg'].mean()
highest_avg_mpg_origin = avg_mpg_by_origin.idxmax()
print("\n#4-6:")
print(f"Origin with the highest average mpg:
 {highest_avg_mpg_origin} - Average MPG:
 {avg_mpg_by_origin.max():.2f}")

# 4-7: Filter cars with mpg lower than 10th percentile.
mpg_10th_percentile = mpg['mpg'].quantile(0.10)
filtered_mpg = mpg[mpg['mpg'] > mpg_10th_percentile]
print("\n#4-7:")
print(f"Number of cars remaining after removing those with mpg below
10th percentile: {len(filtered_mpg)}")
```

Project 4 Output

```
#4-1:
      mpg  cylinders  displacement  horsepower ...
387  27.0       4        140.0        86.0
388  44.0       4         97.0        52.0
389  32.0       4        135.0        84.0
390  28.0       4        120.0        79.0
391  31.0       4        119.0        82.0

#4-2:
Number of missing values replaced in 'horsepower': 6

#4-3:
```

Number of cars from 'usa' with 4 cylinders: 72

#4-4:
Car with the highest mpg:
mazda glc - MPG: 46.6

#4-5:
Car with the highest 'displacement_to_power' ratio:
peugeot 504 - Ratio: 2.5396825396825395

#4-6:
Origin with the highest average mpg:
europe - Average MPG: 27.89

#4-7:
Number of cars remaining after removing those
with mpg below 10th percentile: 356

PROJECT 5 SOLUTIONS

```
import seaborn as sns
import pandas as pd

# Loading the datasets
tips = sns.load_dataset('tips')
flights = sns.load_dataset('flights')

#5-1
concat_vertically = pd.concat([tips, flights], axis=0)
print("#5-1:")
print(concat_vertically.head())

#5-2
df_discount = pd.DataFrame({
    'total_bill': [16.99, 10.34, 21.01],
    'discount': [10, 5, 7]
})
merged_df = pd.merge(tips, df_discount, on='total_bill',
 how='inner')
print("\n#5-2:")
print(merged_df.head())

#5-3
df_feedback = pd.DataFrame({
    'total_bill': [16.99, 10.34, 21.01],
    'feedback': ['Good', 'Average', 'Excellent']
})
left_joined = pd.merge(tips, df_feedback, on='total_bill', how='left')
print("\n#5-3:")
print(left_joined.head())

#5-4
```

```
df_review = pd.DataFrame({
    'total_bill': [16.99, 10.34, 15.00],
    'restaurant_review': ['4 stars', '3 stars', '5 stars']
})
outer_joined = pd.merge(tips, df_review, on='total_bill', how='outer')
print("\n#5-4:")
print(outer_joined.head())

#5-5
concat_horizontally = pd.concat([tips, flights], axis=1)
print("\n#5-5:")
print(concat_horizontally.head())

#5-6
df_experience = pd.DataFrame({
    'total_bill': [16.99, 10.34, 21.01],
    'tip': [1.01, 1.66, 3.50],
    'dining_experience': ['Casual', 'Formal', 'Casual']
})
merged_overlapping = pd.merge(tips, df_experience, on=['total_bill', 'tip'],
how='inner')
print("\n#5-6:")
print(merged_overlapping.head())

#5-7
multi_index_df = pd.DataFrame({
    'total_bill': [16.99, 10.34, 21.01],
    'tip': [1.01, 1.66, 3.50],
    'meal_preference': ['Veg', 'Non-Veg', 'Non-Veg']
}).set_index(['total_bill', 'tip'])
merged_multiindex = pd.merge(tips.set_index(['total_bill', 'tip']),
multi_index_df, left_index=True, right_index=True)
print("\n#5-7:")
print(merged_multiindex.head())
```

Project 5 Output

#5-1:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner   2.0
1      10.34  1.66    Male     No  Sun  Dinner   3.0
2      21.01  3.50    Male     No  Sun  Dinner   3.0
3      23.68  3.31    Male     No  Sun  Dinner   2.0
4      24.59  3.61  Female     No  Sun  Dinner   4.0
```

#5-2:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner     2
1      10.34  1.66    Male     No  Sun  Dinner     3
2      21.01  3.50    Male     No  Sun  Dinner     3
```

#5-3:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner     2
1      10.34  1.66    Male     No  Sun  Dinner     3
```

#5-4:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner     2
1      10.34  1.66    Male     No  Sun  Dinner     3
```

#5-5:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner   2.0
1      10.34  1.66    Male     No  Sun  Dinner   3.0
2      21.01  3.50    Male     No  Sun  Dinner   3.0
3      23.68  3.31    Male     No  Sun  Dinner   2.0
4      24.59  3.61  Female     No  Sun  Dinner   4.0
```

#5-6:
```
   total_bill   tip     sex smoker  day    time  size ...
0      16.99  1.01  Female     No  Sun  Dinner     2
1      10.34  1.66    Male     No  Sun  Dinner     3
```

#5-7:
```
              sex smoker  day    time  size ...
```

total_bill tip
16.99 1.01 Female No Sun Dinner 2
10.34 1.66 Male No Sun Dinner 3

PROJECT 6 SOLUTIONS

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# Load the dataset
tips = sns.load_dataset('tips')

#6-1:
tips['total_bill'].hist(edgecolor='black')
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.title('Histogram of Total Bill')
plt.show()

#6-2:
correlation_matrix = tips.corr()
print(correlation_matrix)

#6-3:
tips.plot.scatter(x='total_bill', y='tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.title('Scatter plot of Total Bill vs Tip')
plt.show()

#6-4:
skewness = tips['total_bill'].skew()
kurtosis = tips['total_bill'].kurt()
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis}")

#6-5:
```

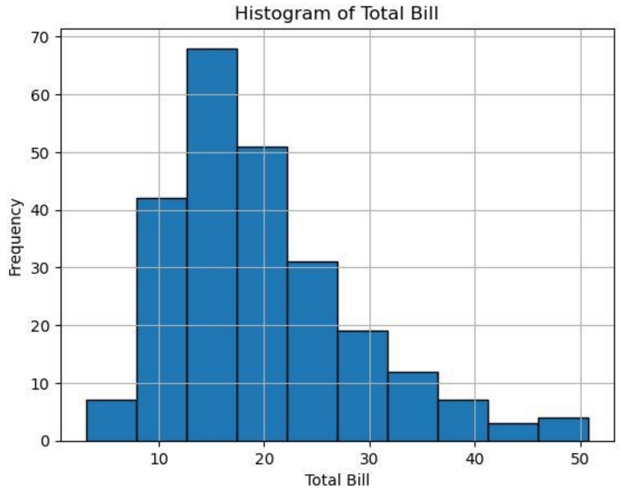
```
value_counts = tips['day'].value_counts()
print(value_counts)
value_counts.plot(kind='bar')
plt.xlabel('Day')
plt.ylabel('Frequency')
plt.title('Frequency of unique values in Day column')
plt.show()
```

```
#6-6:
tips.boxplot(column='total_bill')
plt.xlabel('Total Bill')
plt.ylabel('Value')
plt.title('Box plot of Total Bill')
plt.show()
```

```
#6-7:
Q1 = tips['total_bill'].quantile(0.25)
Q3 = tips['total_bill'].quantile(0.75)
IQR = Q3 - Q1
print(f"IQR for total_bill: {IQR}")
```

Project 6 Output

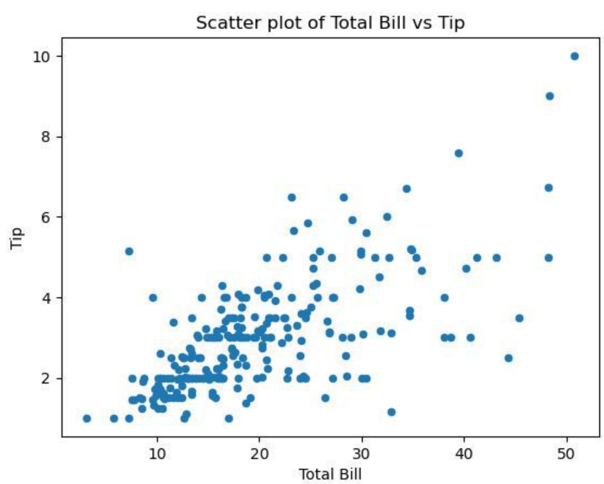
#6-1:



#6-2:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

#6-3:



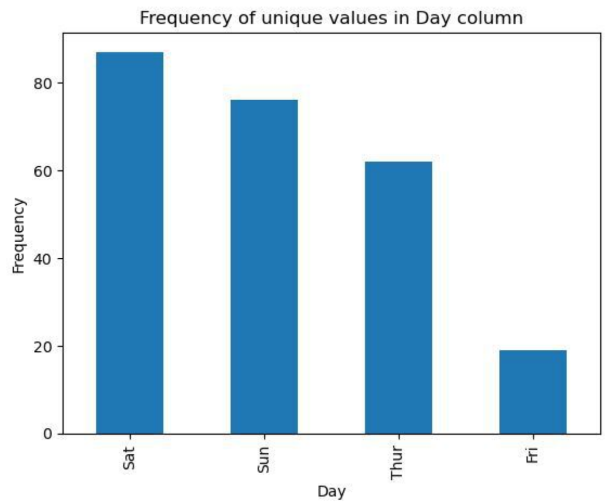
#6-4:
Skewness: 1.1332130376158205
Kurtosis: 1.2184840156638854

#6-5:

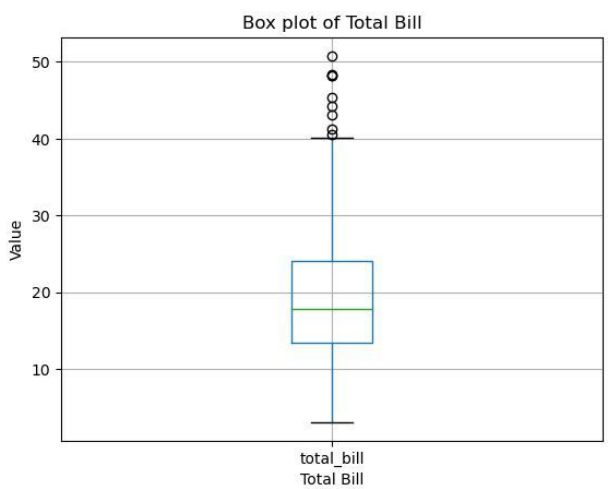
Sat 87
Sun 76
Thur 62
Fri 19
Name: day, dtype: int64
Displays bar chart with days

#6-5:

```
Sat     87
Sun     76
Thur    62
Fri     19
Name: day, dtype: int64
```



#6-6:



#6-7:
IQR for total_bill: 10.779999999999998

PROJECT 7 SOLUTIONS

```
import seaborn as sns
import numpy as np
import pandas as pd

# Load the dataset
diamonds = sns.load_dataset('diamonds')

#7-1:
filtered_diamonds = diamonds.query('cut == "Ideal" and price > 500')
print(filtered_diamonds)

#7-2:
pattern_matched_rows =
 diamonds[diamonds['color'].str.match('^[EF]$')]
print(pattern_matched_rows)

#7-3:
diamonds_with_color_index = diamonds.set_index('color')
reset_diamonds = diamonds_with_color_index.reset_index()
print(reset_diamonds)

#7-4:
diamonds['price_per_carat'] = diamonds.eval('price / carat')
print(diamonds[['price', 'carat', 'price_per_carat']])

#7-5:
diamonds['high_price'] = np.where(diamonds['price'] > 1000, True, False)
print(diamonds[['price', 'high_price']])

#7-6:
low_variance_cols =
 diamonds.var()[diamonds.var() < 1].index.tolist()
diamonds_dropped = diamonds.drop(columns=low_variance_cols)
```

```
print(diamonds_dropped)
```

```
#7-7:
melted_diamonds = diamonds.melt(id_vars=['color', 'cut'])
print(melted_diamonds)
```

Project 7 Output

#7-1:

```
        carat   cut color clarity  depth  table  price ...
60      0.35  Ideal     I     VS1   60.9   57.0    552
62      0.30  Ideal     D     SI1   62.5   57.0    552
63      0.30  Ideal     D     SI1   62.1   56.0    552
65      0.28  Ideal     G    VVS2   61.4   56.0    553
66      0.32  Ideal     I    VVS1   62.0   55.3    553
...      ...    ...   ...     ...    ...    ...    ...
53925   0.79  Ideal     I     SI1   61.6   56.0   2756
53926   0.71  Ideal     E     SI1   61.9   56.0   2756
53929   0.71  Ideal     G     VS1   61.4   56.0   2756
53935   0.72  Ideal     D     SI1   60.8   57.0   2757
53939   0.75  Ideal     D     SI2   62.2   55.0   2757

[20919 rows x 10 columns]
```

#7-2

```
        carat        cut color clarity  depth  table  price ...
0       0.23      Ideal     E     SI2   61.5   55.0    326
1       0.21    Premium     E     SI1   59.8   61.0    326
2       0.23       Good     E     VS1   56.9   65.0    327
8       0.22       Fair     E     VS2   65.1   61.0    337
12      0.22    Premium     F     SI1   60.4   61.0    342
...      ...        ...   ...     ...    ...    ...    ...
53928   0.79    Premium     E     SI2   61.4   58.0   2756
53930   0.71    Premium     E     SI1   60.5   55.0   2756
53931   0.71    Premium     F     SI1   59.8   62.0   2756
53932   0.70  Very Good     E     VS2   60.5   59.0   2757
```

```
53933   0.70  Very Good     E     VS2   61.2   59.0   2757

[19339 rows x 10 columns]
```

#7-3

```
      color  carat        cut clarity  depth  table  price ...
0         E   0.23      Ideal     SI2   61.5   55.0    326
1         E   0.21    Premium     SI1   59.8   61.0    326
2         E   0.23       Good     VS1   56.9   65.0    327
3         I   0.29    Premium     VS2   62.4   58.0    334
4         J   0.31       Good     SI2   63.3   58.0    335
...     ...    ...        ...     ...    ...    ...    ...
53935     D   0.72      Ideal     SI1   60.8   57.0   2757
53936     D   0.72       Good     SI1   63.1   55.0   2757
53937     D   0.70  Very Good     SI1   62.8   60.0   2757
53938     H   0.86    Premium     SI2   61.0   58.0   2757
53939     D   0.75      Ideal     SI2   62.2   55.0   2757

[53940 rows x 10 columns]
```

#7-4

```
       price  carat  price_per_carat
0        326   0.23      1417.391304
1        326   0.21      1552.380952
2        327   0.23      1421.739130
3        334   0.29      1151.724138
4        335   0.31      1080.645161
...      ...    ...              ...
53935   2757   0.72      3829.166667
53936   2757   0.72      3829.166667
53937   2757   0.70      3938.571429
53938   2757   0.86      3205.813953
53939   2757   0.75      3676.000000

[53940 rows x 3 columns]
```

#7-5

```
       price  high_price
```

```
0       326      False
1       326      False
2       327      False
3       334      False
4       335      False
...     ...        ...
53935   2757       True
53936   2757       True
53937   2757       True
53938   2757       True
53939   2757       True

[53940 rows x 2 columns]
```

#7-6

```
             cut color clarity  depth  table  price ...
0          Ideal     E     SI2   61.5   55.0    326
1        Premium     E     SI1   59.8   61.0    326
2           Good     E     VS1   56.9   65.0    327
3        Premium     I     VS2   62.4   58.0    334
4           Good     J     SI2   63.3   58.0    335
...          ...   ...     ...    ...    ...    ...
53935      Ideal     D     SI1   60.8   57.0   2757
53936       Good     D     SI1   63.1   55.0   2757
53937  Very Good     D     SI1   62.8   60.0   2757
53938    Premium     H     SI2   61.0   58.0   2757
53939      Ideal     D     SI2   62.2   55.0   2757
```

#7-7

```
        color      cut    variable value
0           E    Ideal       carat  0.23
1           E  Premium       carat  0.21
2           E     Good       carat  0.23
3           I  Premium       carat  0.29
4           J     Good       carat  0.31
...       ...      ...         ...   ...
539395      D    Ideal  high_price  True
```

```
539396     D       Good  high_price  True
539397     D  Very Good  high_price  True
539398     H    Premium  high_price  True
539399     D      Ideal  high_price  True

[539400 rows x 4 columns]
```

PROJECT 8 SOLUTIONS

```
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import StandardScaler,
 OneHotEncoder
from sklearn.decomposition import PCA
from statsmodels.stats.outliers_influence
 import variance_inflation_factor

# Load dataset
titanic = sns.load_dataset('titanic')

# 8-1
titanic['family_size'] = titanic['sibsp'] + titanic['parch']
print("#8-1:\n", titanic['family_size'].head())

# 8-2
encoder = OneHotEncoder(drop='first', sparse=False)
encoded_class = encoder.fit_transform(titanic[['class']])
encoded_df = pd.DataFrame(encoded_class,
 columns=encoder.get_feature_names(['class']))
titanic = pd.concat([titanic, encoded_df], axis=1)
print("#8-2:\n", titanic[encoded_df.columns].head())

# 8-3
scaler = StandardScaler()
titanic['age_z'] = scaler.fit_transform(titanic[['age']])
print("#8-3:\n", titanic['age_z'].head())

# 8-4
# Note: Normally, you'd remove categorical variables or one-hot encode
them, but for simplicity, we'll use only 'age' and 'fare' here.
pca = PCA(n_components=1)
```

```
titanic['pca_component'] = pca.fit_transform(titanic[['age', 'fare']])
print("#8-4:\n", titanic['pca_component'].head())

# 8-5
# Computing VIF for age, fare, and the one-hot encoded class columns
features = ['age', 'fare', 'class_Second', 'class_Third']
vif_data = pd.DataFrame()
vif_data['feature'] = features
vif_data['VIF'] = [variance_inflation_factor(titanic[features].values, i) for i
in range(titanic[features].shape[1])]
print("#8-5:\n", vif_data)

# 8-6
bins = [0, 18, 35, 60, 100]
labels = ['child', 'young_adult', 'adult', 'senior']
titanic['age_group'] = pd.cut(titanic['age'], bins=bins, labels=labels,
right=False)
print("#8-6:\n", titanic['age_group'].head())

# 8-7
mean_encode = titanic.groupby('embark_town')['survived'].mean()
titanic['embark_encoded'] =
 titanic['embark_town'].map(mean_encode)
print("#8-7:\n", titanic['embark_encoded'].head())
```

Project 8 Output

```
#8-1:
0    1
1    1
2    0
3    1
4    0
Name: family_size, dtype: int64

#8-2:
    class_Second  class_Third
```

```
0          0.0          1.0
1          0.0          0.0
2          0.0          1.0
3          0.0          0.0
4          0.0          1.0
```

#8-3:

```
0   -0.592481
1    0.638789
2   -0.284663
3    0.407926
4    0.407926
Name: age_z, dtype: float64
```

#8-4:

```
0   -20.493525
1    40.918998
2   -18.387374
3    23.506727
4   -17.887235
Name: pca_component, dtype: float64
```

#8-5:

```
        feature       VIF
0           age  4.395834
1          fare  1.691754
2  class_Second  1.859111
3   class_Third  2.931346
```

#8-6:

```
0    young_adult
1          adult
2    young_adult
3          adult
4          adult
Name: age_group, dtype: category
Categories (4, object): ['child' < 'young_adult' < 'adult' < 'senior']
```

#8-7:
0