1. Getting Familiar with Pandas

```python
#importing pandas and numpy packages
import pandas as pd
import numpy as np

#Creating a Series from list
data = [1, 2, 3, 4, 5]
series_from_list = pd.Series(data)
print(series_from_list)

#From a dictionary:
data = {'a': 1, 'b': 2, 'c': 3}
series_from_dict = pd.Series(data)
print(series_from_dict)

#From a scalar value:
scalar_series = pd.Series(5, index=[0, 1, 2, 3])
print(scalar_series)

#Using a Numpy Array
data=np.random.randint(1,10,10)
pd.Series(data)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
a    1
b    2
c    3
dtype: int64
0    5
1    5
2    5
3    5
dtype: int64
```

```
0    1
1    8
2    6
3    7
4    1
5    6
6    6
7    2
```

```
8    8
9    2
dtype: int64
```

```
#Creating a DataFrame
#You can create a DataFrame from lists, dictionaries, or even CSV
files:

#From a list of lists:
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
df_from_list = pd.DataFrame(data, columns=['A', 'B', 'C'])
print(df_from_list)
```

```
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
#From a dictionary:
data = {'A': [1, 4, 7], 'B': [2, 5, 8], 'C': [3, 6, 9]}
df_from_dict = pd.DataFrame(data)
print(df_from_dict)
```

```
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
#From a CSV file:
df_from_csv = pd.read_csv('/content/Toy-Sales-dataset.csv')
print(df_from_csv)
```

```
    Month  Sales  PromExp  Price  AdExp
0       1  73959    61.13   8.75  50.04
1       2  71544    60.19   8.99  50.74
2       3  78587    59.16   7.50  50.14
3       4  80364    60.38   7.25  50.27
4       5  78771    59.71   7.40  51.25
5       6  71986    59.88   8.50  50.65
6       7  74885    60.14   8.40  50.87
7       8  73345    60.08   7.90  50.15
8       9  76659    59.90   7.25  48.24
9      10  71880    59.68   8.70  50.19
10     11  73598    59.83   8.40  51.11
11     12  74893    59.77   8.10  51.49
12     13  69003    59.29   8.40  50.10
13     14  78542    60.40   7.40  49.24
14     15  72543    59.89   8.00  50.04
15     16  74247    60.06   8.30  49.46
16     17  76253    60.51   8.10  51.62
17     18  72582    58.93   8.20  49.78
```

```
18      19  69022    60.09   8.99  48.60
19      20  76200    61.00   7.99  49.00
20      21  69701    59.00   8.50  48.00
21      22  77005    59.50   7.90  54.00
22      23  70987    58.00   7.99  48.70
23      24  75643    60.50   8.25  50.00
```

```python
#Selecting Data
#Selecting a column (Series) from a DataFrame:
col_a = df_from_dict['A']
print(col_a)
```

```
0    1
1    4
2    7
Name: A, dtype: int64
```

```python
#Selecting multiple columns:
cols_ab = df_from_dict[['A', 'B']]
print(cols_ab)
```

```
   A  B
0  1  2
1  4  5
2  7  8
```

```python
#Selecting rows by index:
first_row = df_from_dict.iloc[0]
print(first_row)
```

```
A    1
B    2
C    3
Name: 0, dtype: int64
```

```python
#Selecting rows and columns:
specific_data = df_from_dict.iloc[0, 1]  # First row, second column
print(specific_data)
```

```
2
```

```python
#Filtering Rows
#Filtering rows based on a condition:

filtered_df = df_from_dict[df_from_dict['A'] > 4]
print(filtered_df)
```

```
   A  B  C
2  7  8  9
```

```python
#Modifying Data
#Adding a new column:
```

```
df_from_dict['D'] = df_from_dict['A'] + df_from_dict['B']
print(df_from_dict)
```

```
   A  B  C   D
0  1  2  3   3
1  4  5  6   9
2  7  8  9  15
```

```
#Modifying existing column values:
df_from_dict['A'] = df_from_dict['A'] * 2
print(df_from_dict)
```

```
    A  B  C   D
0   2  2  3   3
1   8  5  6   9
2  14  8  9  15
```

```
#Dropping a column:
df_dropped = df_from_dict.drop('D', axis=1)
print(df_dropped)
```

```
    A  B  C
0   2  2  3
1   8  5  6
2  14  8  9
```

```
#Renaming columns:
df_renamed = df_from_dict.rename(columns={'A': 'Alpha', 'B': 'Beta'})
print(df_renamed)
```

```
   Alpha  Beta  C   D
0      2     2  3   3
1      8     5  6   9
2     14     8  9  15
```

2.Data Handling with Pandas:

```
# Sample data: creating a DataFrame with some missing values and
duplicates
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank',
'Alice'],
    'Age': [25, np.nan, 17, 40, 35, np.nan, 25],
    'City': ['New York', 'Los Angeles', 'New York', 'Chicago', 'Los
Angeles', 'New York', 'New York']
}

df = pd.DataFrame(data)
```

```python
# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Handling missing data
# Fill missing values with a specific value (e.g., mean of the column
for numerical data)
df['Age'] = df['Age'].fillna(df['Age'].mean())

# Alternatively, drop rows with missing values
# df = df.dropna()

print("\nDataFrame after handling missing values:")
print(df)

# Removing duplicates
df = df.drop_duplicates()

print("\nDataFrame after removing duplicates:")
print(df)

# Data type conversions
# Convert 'Age' to integer type (if needed)
df['Age'] = df['Age'].astype(int)

print("\nDataFrame after type conversion:")
print(df)

# Additional transformations
# For example, adding a new column based on existing data
df['Is_Adult'] = df['Age'] >= 18

print("\nDataFrame after adding new column 'Is_Adult':")
print(df)
```

```
Original DataFrame:
      Name   Age         City
0    Alice  25.0     New York
1      Bob   NaN  Los Angeles
2  Charlie  17.0     New York
3    David  40.0      Chicago
4      Eve  35.0  Los Angeles
5    Frank   NaN     New York
6    Alice  25.0     New York

DataFrame after handling missing values:
      Name   Age         City
0    Alice  25.0     New York
1      Bob  28.4  Los Angeles
2  Charlie  17.0     New York
```

```
3    David  40.0      Chicago
4      Eve  35.0  Los Angeles
5    Frank  28.4     New York
6    Alice  25.0     New York

DataFrame after removing duplicates:
      Name   Age         City
0    Alice  25.0     New York
1      Bob  28.4  Los Angeles
2  Charlie  17.0     New York
3    David  40.0      Chicago
4      Eve  35.0  Los Angeles
5    Frank  28.4     New York

DataFrame after type conversion:
      Name  Age         City
0    Alice   25     New York
1      Bob   28  Los Angeles
2  Charlie   17     New York
3    David   40      Chicago
4      Eve   35  Los Angeles
5    Frank   28     New York

DataFrame after adding new column 'Is_Adult':
      Name  Age         City  Is_Adult
0    Alice   25     New York      True
1      Bob   28  Los Angeles      True
2  Charlie   17     New York     False
3    David   40      Chicago      True
4      Eve   35  Los Angeles      True
5    Frank   28     New York      True
```

```
# Reading data from a CSV file
df1= pd.read_csv('/content/Titanic-Dataset.csv')
print(df1)
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age
```

```
SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0
1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                               Heikkinen, Miss. Laina  female  26.0
0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                             Allen, Mr. William Henry    male  35.0
0
..                                                 ...     ...   ...
...
886                              Montvila, Rev. Juozas    male  27.0
0
887                       Graham, Miss. Margaret Edith  female  19.0
0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN
1
889                              Behr, Mr. Karl Howell    male  26.0
0
890                                Dooley, Mr. Patrick    male  32.0
0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

```
print(df1.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
df1.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
#handling missing values
df1['Age'] = df1['Age'].fillna(df1['Age'].mean())
```

```
df1.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
```

```
Embarked        2
dtype: int64
```

3.Data Analysis with Pandas

```python
import pandas as pd

# Load a DataFrame
df = pd.read_csv('/content/Titanic-Dataset.csv')

# Summary statistics for numerical columns
summary = df.describe()
print(summary)

# Summary statistics for categorical columns
categorical_summary = df.describe(include=['object'])
print(categorical_summary)

# Get the mean of a specific column
mean_value = df['Fare'].mean()
print(mean_value)

# Get the median of a specific column
median_value = df['Fare'].median()
print(median_value)

# Get the standard deviation of a specific column
std_dev = df['Fare'].std()
print(std_dev)
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699118    0.523008   
std     257.353842    0.486592    0.836071   14.526497    1.102743   
min       1.000000    0.000000    1.000000    0.420000    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
```

```
max      6.000000  512.329200
                           Name   Sex  Ticket    Cabin Embarked
count                       891   891     891      204      889
unique                      891     2     681      147        3
top     Braund, Mr. Owen Harris  male  347082  B96 B98        S
freq                          1   577       7        4      644
32.204207968574636
14.4542
49.6934285971809
```

```
df = pd.read_csv('/content/Toy-Sales-dataset.csv')
# Group by a specific column and calculate aggregate statistics
grouped = df.groupby('Sales')

# Apply aggregation functions
mean_values = grouped.mean()
print(mean_values)

sum_values = grouped.sum()
print(sum_values)

count_values = grouped.count()
print(count_values)
```

```
       Month  PromExp  Price  AdExp
Sales                              
69003   13.0    59.29   8.40  50.10
69022   19.0    60.09   8.99  48.60
69701   21.0    59.00   8.50  48.00
70987   23.0    58.00   7.99  48.70
71544    2.0    60.19   8.99  50.74
71880   10.0    59.68   8.70  50.19
71986    6.0    59.88   8.50  50.65
72543   15.0    59.89   8.00  50.04
72582   18.0    58.93   8.20  49.78
73345    8.0    60.08   7.90  50.15
73598   11.0    59.83   8.40  51.11
73959    1.0    61.13   8.75  50.04
74247   16.0    60.06   8.30  49.46
74885    7.0    60.14   8.40  50.87
74893   12.0    59.77   8.10  51.49
75643   24.0    60.50   8.25  50.00
76200   20.0    61.00   7.99  49.00
76253   17.0    60.51   8.10  51.62
76659    9.0    59.90   7.25  48.24
77005   22.0    59.50   7.90  54.00
78542   14.0    60.40   7.40  49.24
78587    3.0    59.16   7.50  50.14
78771    5.0    59.71   7.40  51.25
80364    4.0    60.38   7.25  50.27
```

```
      Month  PromExp  Price  AdExp
Sales                             
69003    13    59.29   8.40  50.10
69022    19    60.09   8.99  48.60
69701    21    59.00   8.50  48.00
70987    23    58.00   7.99  48.70
71544     2    60.19   8.99  50.74
71880    10    59.68   8.70  50.19
71986     6    59.88   8.50  50.65
72543    15    59.89   8.00  50.04
72582    18    58.93   8.20  49.78
73345     8    60.08   7.90  50.15
73598    11    59.83   8.40  51.11
73959     1    61.13   8.75  50.04
74247    16    60.06   8.30  49.46
74885     7    60.14   8.40  50.87
74893    12    59.77   8.10  51.49
75643    24    60.50   8.25  50.00
76200    20    61.00   7.99  49.00
76253    17    60.51   8.10  51.62
76659     9    59.90   7.25  48.24
77005    22    59.50   7.90  54.00
78542    14    60.40   7.40  49.24
78587     3    59.16   7.50  50.14
78771     5    59.71   7.40  51.25
80364     4    60.38   7.25  50.27
      Month  PromExp  Price  AdExp
Sales                             
69003     1        1      1      1
69022     1        1      1      1
69701     1        1      1      1
70987     1        1      1      1
71544     1        1      1      1
71880     1        1      1      1
71986     1        1      1      1
72543     1        1      1      1
72582     1        1      1      1
73345     1        1      1      1
73598     1        1      1      1
73959     1        1      1      1
74247     1        1      1      1
74885     1        1      1      1
74893     1        1      1      1
75643     1        1      1      1
76200     1        1      1      1
76253     1        1      1      1
76659     1        1      1      1
77005     1        1      1      1
78542     1        1      1      1
```

```
78587      1        1      1      1
78771      1        1      1      1
80364      1        1      1      1

#Merging DataFrames
#Merging combines two DataFrames based on a key column:

# Create DataFrames
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value1': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['A', 'B', 'D'], 'value2': [4, 5, 6]})

# Merge DataFrames on a key column
merged_df = pd.merge(df1, df2, on='key', how='inner')  # other
options: 'outer', 'left', 'right'
print(merged_df)

  key  value1  value2
0   A       1       4
1   B       2       5

#Joining DataFrames
#Joining is similar to merging but is often used with the index:

# Create DataFrames with index
df1 = pd.DataFrame({'value1': [1, 2, 3]}, index=['A', 'B', 'C'])
df2 = pd.DataFrame({'value2': [4, 5, 6]}, index=['A', 'B', 'D'])

# Join DataFrames on index
joined_df = df1.join(df2, how='inner')  # other options: 'outer',
'left', 'right'
print(joined_df)

   value1  value2
A       1       4
B       2       5

#Concatenating DataFrames
#Concatenation stacks DataFrames either vertically or horizontally:

# Create DataFrames
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

# Concatenate DataFrames vertically
concat_df = pd.concat([df1, df2], axis=0)  # axis=0 for vertical,
axis=1 for horizontal

# Concatenate DataFrames horizontally
concat_df_horizontal = pd.concat([df1, df2], axis=1)

print("Vertical Concatenation:")
print(concat_df)

print("Horizontal Concatenation:")
print(concat_df_horizontal)
```

```
Vertical Concatenation:
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
Horizontal Concatenation:
   A  B  A  B
0  1  3  5  7
1  2  4  6  8
```

4.Application in Data Science:

Key Advantages of Using Pandas in Data Science

1. **Efficient Data Structures**:
   - **DataFrame** and **Series** for handling structured data.
   - Optimized for performance with large datasets.
2. **Powerful Data Manipulation**:
   - Advanced indexing, selection, and filtering.
   - Aggregation and grouping functionalities.
3. **Data Cleaning and Preparation**:
   - Methods for handling missing values and data transformation.
4. **Time Series Analysis**:
   - Robust support for date and time data, including resampling and frequency conversion.
5. **Data Merging and Joining**:
   - Easy combination of data from multiple sources.
6. **Integration with Other Libraries**:
   - Seamless integration with NumPy, scikit-learn, Matplotlib, and Seaborn.

Real-World Examples

1. **Data Cleaning**:
   - Removing duplicates, handling missing values, and type conversion.
2. **Exploratory Data Analysis (EDA)**:
   - Generating descriptive statistics and visualizing data patterns.
3. **Feature Engineering**:
   - Creating new features and normalizing data for modeling.
4. **Time Series Analysis**:
   - Analyzing trends, seasonal patterns, and forecasting with datetime data.

Pandas enhances efficiency and effectiveness in handling, analyzing, and visualizing data, making it an essential tool for data science professionals.