Introduction to Pandas

What is Pandas?

Pandas is a Python library widely used for data manipulation and analysis. It offers intuitive data structures like Series and DataFrame, akin to labeled arrays and tables, respectively. With Pandas, users can effortlessly perform tasks like cleaning, reshaping, and aggregating data. It seamlessly integrates with other Python libraries such as NumPy, Matplotlib, and Scikit-learn, making it a versatile tool for data analysis and visualization.

Why Pandas for Data Science?

Pandas is widely preferred for data science due to several reasons:

Easy Data Manipulation: Pandas simplifies data manipulation tasks such as cleaning, filtering, transforming, and analyzing data.
Flexible Data Structures: With Series and DataFrame, Pandas provides flexible and intuitive data structures that can handle a wide variety of data types and formats, including structured, semi-structured, and time series data.
Integration with Other Libraries: Pandas seamlessly integrates with other Python libraries such as NumPy, Matplotlib, and Scikit-learn, enabling users to leverage the full ecosystem of Python tools for data analysis, visualization, and machine learning.
Rich Functionality: Pandas offers a rich set of functions and methods for data manipulation, including powerful tools for handling missing data, reshaping datasets, grouping and aggregating data, and performing time series analysis.
Community Support: Being an open-source project, Pandas benefits from a large and active community of users and contributors, ensuring continuous development, support, and improvement of the library.

Installing Pandas

Installing Pandas is straightforward using Python's package manager, pip.
Open your terminal or command prompt and run the following command (you can copy & paste):

In [1]:
```
# pip install pandas
```

This will download and install the latest version of Pandas along with any required dependencies.
Once installed, you can import Pandas into your Python scripts or notebooks using the following import statement:

In [2]:
```
import pandas as pd
```

- ## Loading Data to Pandas

Pandas provides convenient functions to read data from various file formats into DataFrame objects. For this tutorial, we will focus on reading data from CSV files. However, Pandas also supports reading data from other formats such as Excel, JSON, SQL databases, and more.

In [3]:
```
# Reading data from a CSV file into a DataFrame
df = pd.read_csv('clothing_prices.csv')
```

Data Exploration and Manipulation

Data Viewing and Accessing: head(), tail(), loc(), iloc()

- ### Display the first rows of the dataframe

In [4]:
```
# By default, it returns the first 5 rows
df.head()
```

Out[4]:

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182.0
1	New Balance	Jeans	Black	XS	Silk	57.0
2	Under Armour	Dress	Red	M	Wool	127.0
3	NaN	Shoes	Green	M	Cotton	77.0
4	NaN	Sweater	White	M	Nylon	113.0

- ### Display the last rows of the dataframe

In [5]:
```
# By default, it returns the last 5 rows
df.tail()
```

Out[5]:

	Brand	Category	Color	Size	Material	Price
995	Puma	Jeans	Black	NaN	Polyester	176.0
996	Puma	Jacket	Red	XXL	Silk	110.0
997	Reebok	Sweater	Blue	NaN	Denim	127.0
998	NaN	Sweater	Black	XXL	Denim	NaN
999	NaN	NaN	Yellow	XS	Wool	NaN

- ### Display a specific number of rows

In [6]:
```
# e.g. return the first 4 rows
df.head(4)
```

Out[6]:

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182.0
1	New Balance	Jeans	Black	XS	Silk	57.0
2	Under Armour	Dress	Red	M	Wool	127.0
3	NaN	Shoes	Green	M	Cotton	77.0

- Display specific columns

In [7]:
```
# Select specific columns (and show the first 3 results)
df[['Brand', 'Category', 'Price']].head(3)
```

Out[7]:

	Brand	Category	Price
0	New Balance	Dress	182.0
1	New Balance	Jeans	57.0
2	Under Armour	Dress	127.0

- Using loc[] to access data by label

In [8]:
```
# Access the row with index label 3
df.loc[3]
```

Out[8]:
```
Brand          NaN
Category     Shoes
Color        Green
Size             M
Material    Cotton
Price         77.0
Name: 3, dtype: object
```

In [9]:
```
# Select rows 23 to 27 and all columns using label-based indexing
df.loc[23:27]
```

Out[9]:

	Brand	Category	Color	Size	Material	Price
23	Puma	Jacket	Red	XXL	Wool	158.0
24	Puma	Jacket	Black	L	Denim	125.0
25	Adidas	Jeans	White	XXL	Nylon	89.0
26	Adidas	Dress	Green	L	Polyester	185.0
27	New Balance	Shoes	Blue	M	Cotton	92.0

- Using iloc[] to access data by position

In [10]:
```
# Access the row at position 2 (zero-based index)
df.iloc[2]
```

Out[10]:
```
Brand       Under Armour
Category           Dress
Color                Red
Size                   M
Material            Wool
Price              127.0
Name: 2, dtype: object
```

In [11]:
```
# Select rows 1 to 4 and all columns using integer-based indexing
df.iloc[1:5]
```

Out[11]:

	Brand	Category	Color	Size	Material	Price
1	New Balance	Jeans	Black	XS	Silk	57.0
2	Under Armour	Dress	Red	M	Wool	127.0
3	NaN	Shoes	Green	M	Cotton	77.0
4	NaN	Sweater	White	M	Nylon	113.0

In [12]:
```
# Return row 2 and column 5 (In Python indexing starts at 0)
df.iloc[1,4]
```

Out[12]:
```
'Silk'
```

Data Filtering and Selection: boolean indexing

- Equals To

In [13]:
```
# Selecting rows where the Brand is "Nike"
df[df['Brand'] == 'Nike'].head()
```

Out[13]:

	Brand	Category	Color	Size	Material	Price
10	Nike	Jacket	White	XL	Silk	98.0
38	Nike	Dress	Yellow	XS	Silk	137.0
45	Nike	Jacket	Black	XS	Silk	38.0
48	Nike	Jeans	White	XS	Silk	138.0
68	Nike	Shoes	White	L	Cotton	21.0

- Greater Than

In [14]:
```
# Selecting rows where the Price is greater than 100
df[df['Price'] > 100].head()
```

Out[14]:

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182.0
2	Under Armour	Dress	Red	M	Wool	127.0
4	NaN	Sweater	White	M	Nylon	113.0
11	Puma	Jacket	White	XL	Silk	150.0
15	Under Armour	Jacket	Green	XXL	Silk	184.0

- ## Is In

In [15]:
```
# Selecting rows where the Category is either "Dress" or "Sweater"
df[df['Category'].isin(['Dress', 'Sweater'])].head()
```

Out[15]:

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182.0
2	Under Armour	Dress	Red	M	Wool	127.0
4	NaN	Sweater	White	M	Nylon	113.0
7	Adidas	Dress	Red	XS	Denim	46.0
8	Reebok	Dress	Black	S	Wool	97.0

- ## Logical AND

In [16]:
```
# Selecting rows where the Size is "XS" and the Material is "Nylon"
df[(df['Size'] == 'XS') & (df['Material'] == 'Nylon')].head()
```

Out[16]:

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182.0
42	Adidas	Jacket	Black	XS	Nylon	173.0
128	Nike	Dress	Red	XS	Nylon	133.0
195	Puma	Sweater	White	XS	Nylon	93.0
215	Under Armour	Dress	Yellow	XS	Nylon	139.0

- ## Logical OR

In [17]:
```
# Select rows where Price is less than or equal to 50 or the Color is "Black"
df[(df['Price'] <= 50) | (df['Color'] == 'Black')].head()
```

Out[17]:

	Brand	Category	Color	Size	Material	Price
1	New Balance	Jeans	Black	XS	Silk	57.0
5	Reebok	Jacket	Red	XL	Nylon	19.0
6	Puma	Jacket	Red	XXL	Polyester	31.0
7	Adidas	Dress	Red	XS	Denim	46.0
8	Reebok	Dress	Black	S	Wool	97.0

Data Manipulation by Adding, Deleting, Renaming columns

- ### Adding a new column

In this code, a new column named 'Discounted_Price' is created in the DataFrame 'df'. The values in this new column are calculated by multiplying the values in the 'Price' column by 0.5. This operation effectively applies a 50% discount to each price value. After executing this code, if we display the first few rows of the DataFrame using df.head(), we'll see the original 'Price' column alongside the newly created 'Discounted_Price' column, which contains the discounted prices.

In [18]:
```
df['Discounted_Price'] = df['Price'] * 0.5
df.head()
```

Out[18]:

	Brand	Category	Color	Size	Material	Price	Discounted_Price
0	New Balance	Dress	White	XS	Nylon	182.0	91.0
1	New Balance	Jeans	Black	XS	Silk	57.0	28.5
2	Under Armour	Dress	Red	M	Wool	127.0	63.5
3	NaN	Shoes	Green	M	Cotton	77.0	38.5
4	NaN	Sweater	White	M	Nylon	113.0	56.5

- ### Deleting an existing column

This code removes the 'Material' column from the DataFrame 'df' using the drop method. The columns parameter specifies the name of the column to be dropped, which is 'Material' in this case. The inplace=True argument ensures that the operation is performed on the DataFrame itself and the changes are reflected in place. After executing this code, if we display the first few rows of the DataFrame using df.head(), we'll see that the 'Material' column has been removed from the DataFrame.

In [19]:
```
df.drop(columns=['Material'], inplace=True)
df.head()
```

Out[19]:

	Brand	Category	Color	Size	Price	Discounted_Price
0	New Balance	Dress	White	XS	182.0	91.0
1	New Balance	Jeans	Black	XS	57.0	28.5
2	Under Armour	Dress	Red	M	127.0	63.5
3	NaN	Shoes	Green	M	77.0	38.5
4	NaN	Sweater	White	M	113.0	56.5

- ### Renaming an existing column

This code renames the 'Category' column to 'Product_Category' in the DataFrame 'df' using the rename method. The columns parameter specifies a dictionary where the keys are the current column names ('Category') and the values are the new column names ('Product_Category'). The inplace=True argument ensures that the operation is performed on the DataFrame itself and the changes are reflected in place.

In [20]:
```
df.rename(columns={'Category': 'Product_Category'}, inplace=True)
df.head()
```

Out[20]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
0	New Balance	Dress	White	XS	182.0	91.0
1	New Balance	Jeans	Black	XS	57.0	28.5
2	Under Armour	Dress	Red	M	127.0	63.5
3	NaN	Shoes	Green	M	77.0	38.5
4	NaN	Sweater	White	M	113.0	56.5

Data Cleaning and Preparation

Dealing with Missing Data: dropna(), fillna()

- ### check for missing values

This method returns the count of missing values for each column in the DataFrame. If a column does not contain any missing values, the sum for that column will be 0. Otherwise, it will return the count of missing values.

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
Brand              4
Product_Category   1
Color              0
Size               2
Price              2
Discounted_Price   2
dtype: int64
```

- **Drop rows with any missing values**

In [22]:
```
df.dropna()
```

Out[22]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
0	New Balance	Dress	White	XS	182.0	91.0
1	New Balance	Jeans	Black	XS	57.0	28.5
2	Under Armour	Dress	Red	M	127.0	63.5
5	Reebok	Jacket	Red	XL	19.0	9.5
6	Puma	Jacket	Red	XXL	31.0	15.5
...	...	...	...	...	...	...
991	Under Armour	Jacket	Red	XL	18.0	9.0
992	Puma	Sweater	Blue	S	20.0	10.0
993	Puma	Sweater	Green	XXL	15.0	7.5
994	New Balance	Jeans	Black	M	48.0	24.0
996	Puma	Jacket	Red	XXL	110.0	55.0

994 rows × 6 columns

- **Fill Missing Values with a Specific Value**

In [23]:
```
# Fill missing values in the 'Brand' column with 'Unknown'
df.fillna(value={'Brand': 'Unknown'})
```

Out[23]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
0	New Balance	Dress	White	XS	182.0	91.0
1	New Balance	Jeans	Black	XS	57.0	28.5
2	Under Armour	Dress	Red	M	127.0	63.5
3	Unknown	Shoes	Green	M	77.0	38.5
4	Unknown	Sweater	White	M	113.0	56.5
...	...	...	...	...	...	...
995	Puma	Jeans	Black	NaN	176.0	88.0
996	Puma	Jacket	Red	XXL	110.0	55.0
997	Reebok	Sweater	Blue	NaN	127.0	63.5
998	Unknown	Sweater	Black	XXL	NaN	NaN
999	Unknown	NaN	Yellow	XS	NaN	NaN

1000 rows × 6 columns

- ### Replacing Missing Values with the Mean

For all instances where the 'Price' data is missing, replace the missing values with the mean price.

In [24]:
```
# Calculate the mean value of the 'Price' column
average_price = df['Price'].mean().round(2)

# Fill missing values in the 'Price' column with the mean value (106.26)
df['Price'].fillna(value=average_price, inplace=True)
```

In [25]:
```
df
```

Out[25]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
0	New Balance	Dress	White	XS	182.00	91.0
1	New Balance	Jeans	Black	XS	57.00	28.5
2	Under Armour	Dress	Red	M	127.00	63.5
3	NaN	Shoes	Green	M	77.00	38.5
4	NaN	Sweater	White	M	113.00	56.5
...	...	...	...	...	...	...
995	Puma	Jeans	Black	NaN	176.00	88.0
996	Puma	Jacket	Red	XXL	110.00	55.0
997	Reebok	Sweater	Blue	NaN	127.00	63.5
998	NaN	Sweater	Black	XXL	106.26	NaN
999	NaN	NaN	Yellow	XS	106.26	NaN

1000 rows × 6 columns

Removing Duplicates

- ### Drop Duplicate Records

Rows are removed that have exactly the same values in all columns as another row in the DataFrame

In [64]:
```
df.drop_duplicates()
```

- ### Removing duplicates based on selected columns

Duplicates are identified and removed based only on the values in the specified subset of columns.

In [65]:
```
df.drop_duplicates(subset=['Brand'])
```

Convert Data Types

- ### Convert Price column from Float to Integer

In [28]:
```
# print the data type of the 'Price' column before conversion
print("Data type of 'Price' column before conversion:", df['Price'].dtype)
```

```
Data type of 'Price' column before conversion: float64
```

In [29]:
```
# convert to Integer
df['Price'] = df['Price'].astype(int)
```

In [30]:
```
# print the data type of the 'Price' column after conversion
print("Data type of 'Price' column after conversion:", df['Price'].dtype)
```

```
Data type of 'Price' column after conversion: int32
```

Data Analysis and Visualization

Descriptive statistics

- describe()

Provides summary statistics for numerical columns in a DataFrame

In [31]:
```
df.describe().round(2)
```

Out[31]:

	Price	Discounted_Price
count	1000.00	998.00
mean	106.26	53.13
std	53.64	26.85
min	10.00	5.00
25%	59.75	29.62
50%	108.00	54.00
75%	150.00	75.00
max	199.00	99.50

- mean()

Calculates the mean of numerical columns

In [32]:
```
# Example: Calculate the mean price of products
df['Price'].mean().round(2)
```

Out[32]:
```
106.26
```

- median()

Calculates the median of numerical columns

In [33]:
```
# Example: Calculate the median price of products
df['Price'].median()
```

Out[33]:
```
108.0
```

- std()

Calculates the standard deviation of numerical column

In [34]:
```
# Example: Calculate the standard deviation of prices
df['Price'].std().round(2)
```

Out[34]:
```
53.64
```

- ## min() and max()

Returns the minimum and maximum values of numerical columns

In [35]:
```
# Example: Get the minimum price
df['Price'].min()
```

Out[35]:
```
10
```

In [36]:
```
# Example: Get the maximum price
df['Price'].max()
```

Out[36]:
```
199
```

- ## value_counts()

Counts the occurrences of each unique value in a column

In [37]:
```
df['Product_Category'].value_counts()
```

Out[37]:
```
Product_Category
Jacket     190
Shoes      172
Jeans      167
Dress      166
Sweater    160
T-shirt    144
Name: count, dtype: int64
```

- ## unique()

Returns an array of unique values in a column

In [38]:
```
df['Color'].unique()
```

Out[38]:
```
array(['White', 'Black', 'Red', 'Green', 'Yellow', 'Blue'], dtype=object)
```

- ## nunique()

Returns the number of unique values in a column

In [39]:
```
df['Brand'].nunique()
```

Out[39]:
```
6
```

Data Sorting and Grouping

- ## Sorting Data

In [40]:
```
# Sort the DataFrame by the 'Price' column in ascending order.
df.sort_values(by='Price')
```

Out[40]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
519	Puma	T-shirt	Blue	XXL	10	5.0
779	Adidas	Jeans	Red	XXL	10	5.0
115	Under Armour	Jeans	Yellow	XS	10	5.0
271	Reebok	Shoes	Green	XXL	10	5.0
102	Reebok	Jeans	Green	XXL	10	5.0
...	...	...	...	...	...	...
472	New Balance	Shoes	Red	L	198	99.0
498	Adidas	Jacket	Red	XS	198	99.0
194	Puma	Jacket	White	XL	199	99.5
223	Nike	Jeans	White	XS	199	99.5
277	Puma	Dress	Yellow	XS	199	99.5

1000 rows × 6 columns

In [41]:
```
# Sort the DataFrame by the 'Price' column in descending order.
df.sort_values(by='Price', ascending=False)
```

Out[41]:

	Brand	Product_Category	Color	Size	Price	Discounted_Price
223	Nike	Jeans	White	XS	199	99.5
277	Puma	Dress	Yellow	XS	199	99.5
194	Puma	Jacket	White	XL	199	99.5
946	New Balance	Dress	Yellow	S	198	99.0
498	Adidas	Jacket	Red	XS	198	99.0
...	...	...	...	...	...	...
779	Adidas	Jeans	Red	XXL	10	5.0
368	Under Armour	Shoes	Yellow	M	10	5.0
102	Reebok	Jeans	Green	XXL	10	5.0
115	Under Armour	Jeans	Yellow	XS	10	5.0
529	Adidas	Jeans	Black	S	10	5.0

1000 rows × 6 columns

- ## Grouping Data

In [42]:
```
# Group DataFrame by'Brand' column and calculate average price for each brand
df.groupby('Brand')['Price'].mean().round(2)
```

Out[42]:
```
Brand
Adidas          104.00
New Balance     115.10
Nike            102.06
Puma            106.14
Reebok          106.49
Under Armour    104.16
Name: Price, dtype: float64
```

In [43]:
```
# Group DataFrame by'Color' column and count the number of items for each color
df.groupby('Color').size()
```

Out[43]:
```
Color
Black     163
Blue      163
Green     162
Red       168
White     171
Yellow    173
dtype: int64
```

In [44]:
```
# Calculate the total sales (sum of prices) for each category
df.groupby('Product_Category')['Price'].sum()
```

Out[44]:
```
Product_Category
Dress      19008
Jacket     19466
Jeans      16825
Shoes      18674
Sweater    17122
T-shirt    15057
Name: Price, dtype: int32
```

In [45]:
```
# Calculate the mean price for each combination of brand and category
df.groupby(['Brand', 'Product_Category'])['Price'].mean().round(2)
```

Out[45]:
```
Brand          Product_Category
Adidas         Dress               113.37
               Jacket              100.46
               Jeans                92.10
               Shoes               111.96
               Sweater              84.79
               T-shirt             122.67
New Balance    Dress               127.67
               Jacket              110.33
               Jeans               114.75
               Shoes               103.35
               Sweater             123.32
               T-shirt             115.46
Nike           Dress                97.68
               Jacket              102.30
               Jeans               106.45
               Shoes               101.25
               Sweater             103.61
               T-shirt             100.96
Puma           Dress               121.33
               Jacket              121.19
               Jeans                97.79
               Shoes               117.78
               Sweater              84.37
               T-shirt              89.54
Reebok         Dress               107.08
               Jacket               93.61
               Jeans               122.54
               Shoes               102.63
               Sweater             116.92
               T-shirt              85.90
Under Armour   Dress               120.27
               Jacket               88.91
               Jeans                81.93
               Shoes               113.00
               Sweater             125.22
               T-shirt             109.48
Name: Price, dtype: float64
```

In [46]:
```
# Calculate both the mean and sum of prices for each brand
df.groupby('Brand')['Price'].agg(['mean', 'sum']).round(2)
```

Out[46]:

Brand	mean	sum
Adidas	104.00	17160
New Balance	115.10	18761
Nike	102.06	16738
Puma	106.14	17831
Reebok	106.49	16826
Under Armour	104.16	18540

Teaser - Data Visualization with Matplotlib

Matplotlib is a Python library used for creating static, interactive, and animated visualizations in Python. It offers a wide range of plotting functions for various types of charts, including line plots, bar charts, histograms, scatter plots, and more. Matplotlib can work seamlessly with Pandas

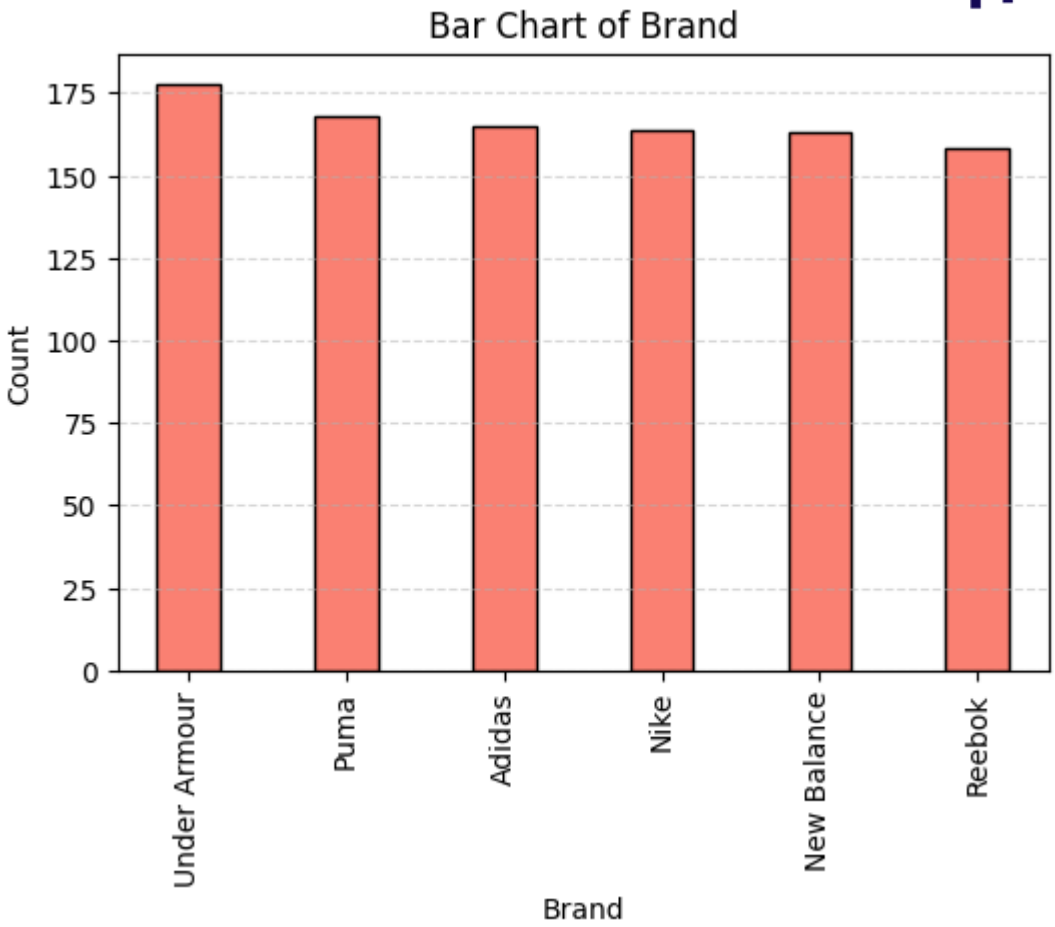
- ### Import Matploplib

In [47]:
```
import matplotlib.pyplot as plt
```

- ### Bar chart

This code snippet creates a bar chart to display the counts of different brands in a DataFrame. It customizes the appearance of the chart by setting the color, width, and edge color of the bars, as well as adding labels to the axes and a title. Additionally, it includes a dashed grid on the y-axis for better visualization.

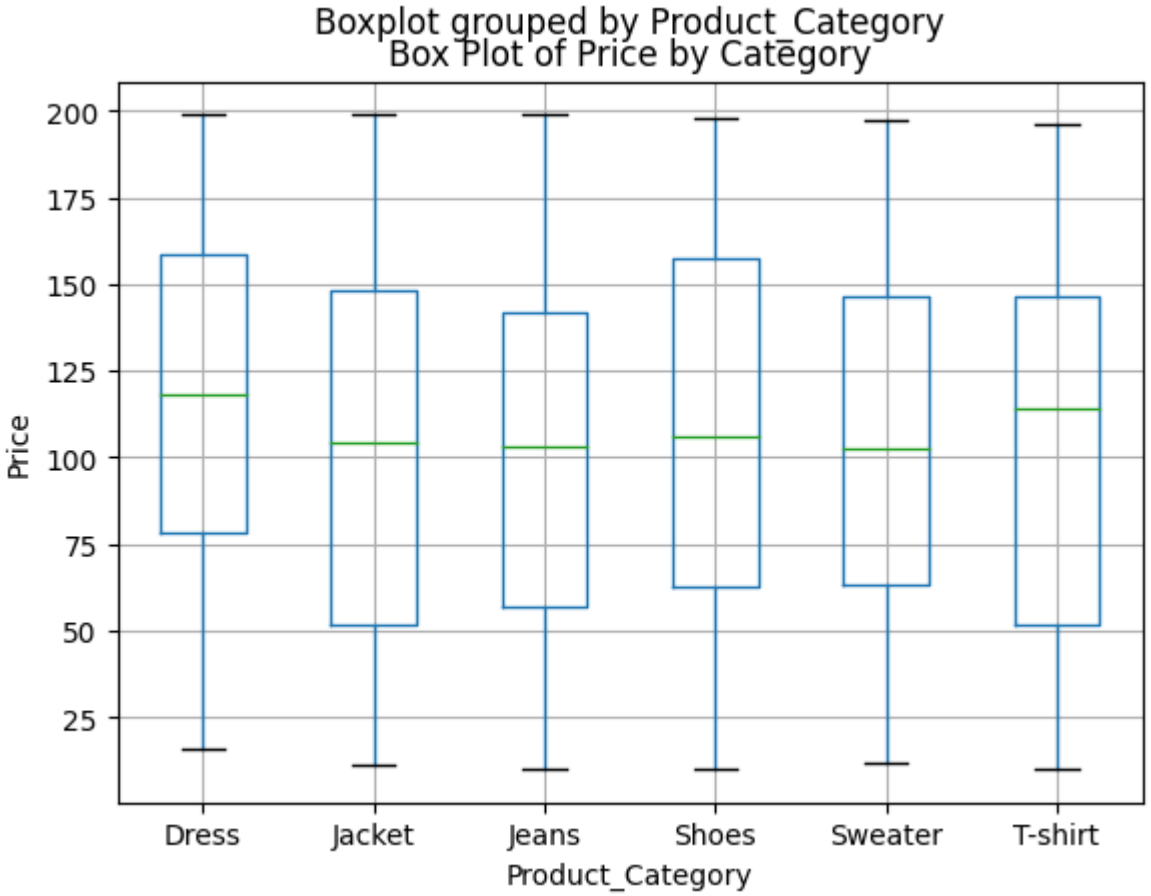
In [48]:
```
# Count the number of different brands in the DataFrame
brand_counts = df['Brand'].value_counts()

# Create a bar chart with custom settings
plt.figure(figsize=(6, 4)) # Set the figure size
brand_counts.plot(kind='bar', color='salmon', width=0.4, edgecolor='black',
                  linewidth=1)
# Set label for x-axis
plt.xlabel('Brand')
# Set label for y-axis
plt.ylabel('Count')
 # Set title of the chart
plt.title('Bar Chart of Brand')
 # Add grid lines on the y-axis
plt.grid(axis='y', linestyle='--', alpha=0.5)
# Display the chart
plt.show()
```



- **Box plot**

This code generates a box plot to visualize the distribution of prices across different categories in the DataFrame. The data is grouped by the 'Category' column, and for each category, a box plot of the 'Price' column is displayed. The x-axis represents the categories, while the y-axis represents the prices. The plot is labeled with appropriate axis labels and a title to provide context and interpretation.

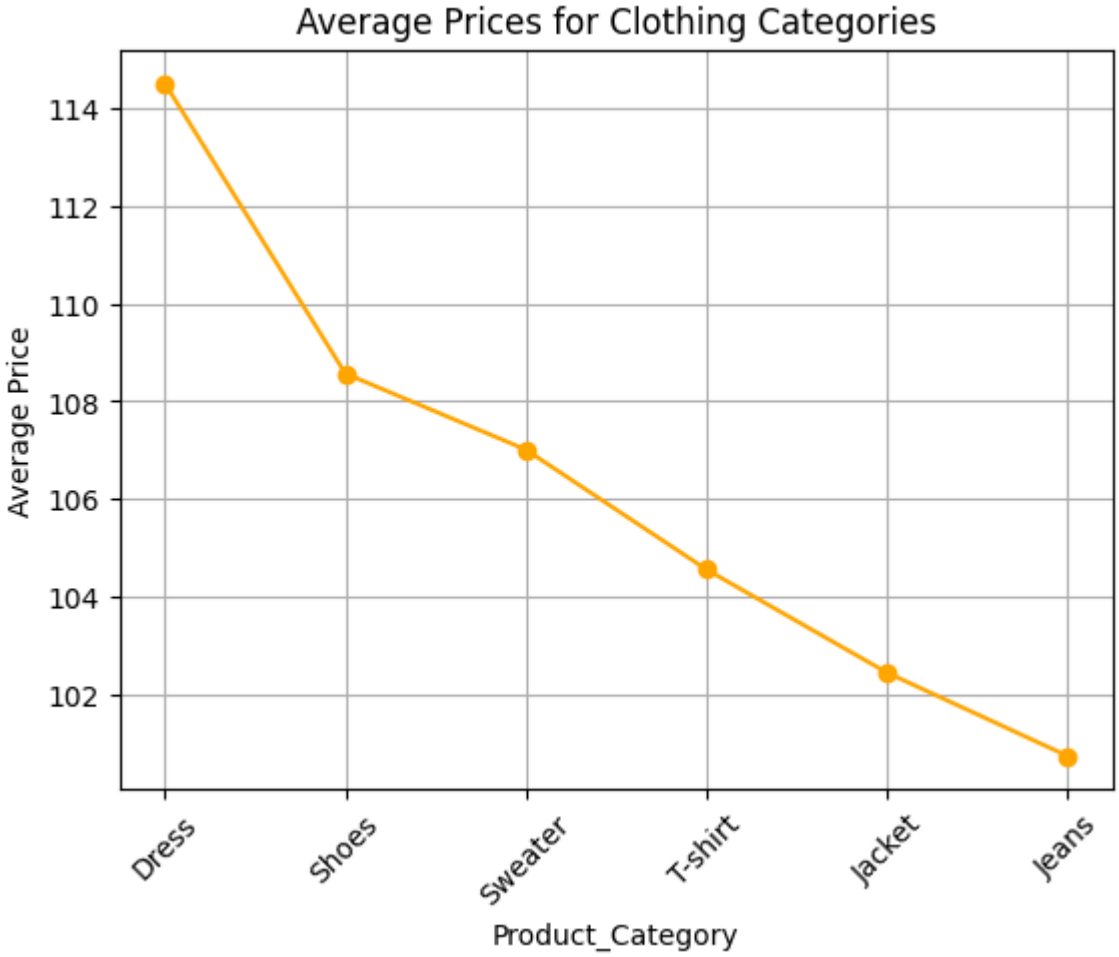
In [49]:
```
# Create a box plot of Price by Category
df.boxplot(column='Price', by='Product_Category')
# Set labels for x-axis and y-axis
plt.xlabel('Product_Category')
plt.ylabel('Price')
# Set title of the plot
plt.title('Box Plot of Price by Category')
# Display the plot
plt.show()
```



- **Line Chart**

This code calculates the average prices for different clothing categories in the DataFrame and then creates a line plot to visualize these average prices. Each category is represented on the x-axis, and the corresponding average price is shown on the y-axis. The marker 'o' indicates individual data points, and the line connects these points to illustrate trends. The plot is titled "Average Prices for Clothing Categories" and includes labels for both axes. Finally, grid lines are added to aid in interpretation.

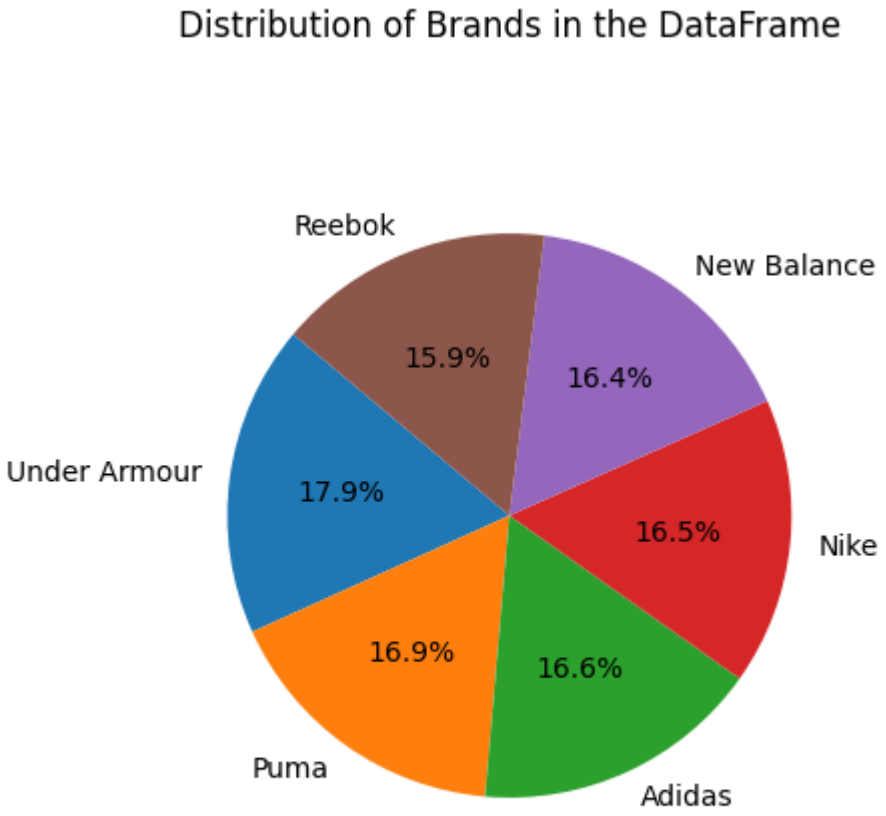
In [59]:
```
# Calculate average prices for all clothing items
average_prices = df.groupby('Product_Category')['Price']
average_prices= average_prices.mean().sort_values(ascending=False)

# Plot average prices as a line chart
average_prices.plot(kind='line', marker='o', color='orange')
# Set title and labels for the plot
plt.title('Average Prices for Clothing Categories')
plt.xlabel('Product_Category')
plt.ylabel('Average Price')
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# Enable gridlines
plt.grid(True)
# Display the plot
plt.show()
```



- **Pie Chart**

This code creates a pie chart that visualizes the distribution of different brands in the DataFrame. It counts the number of occurrences of each brand and then plots the proportions of these counts as slices in the pie chart.

In [60]:
```
# Count the number of different brands in the DataFrame
brand_counts = df['Brand'].value_counts()

# Create a Pie Chart
# Set the figure size
plt.figure(figsize=(4, 6))
plt.pie(brand_counts, labels=brand_counts.index, autopct='%1.1f%%',
        startangle=140)
# Add title to the plot
plt.title('Distribution of Brands in the DataFrame')
# Ensure the pie chart is circular
plt.axis('equal')
# Display the plot
plt.show()
```



Pivot Tables

Pivot tables in Pandas are a way to summarize and analyze data by restructuring it into a more manageable format. They allow you to aggregate and group data based on one or more columns, applying functions to calculate summary statistics or other transformations

- **Pivot table to calculate the average price for each Brand-Category pair**

In [61]:
```
pivot_table_avg_price = df.pivot_table(index='Brand',
                                       columns='Product_Category',
                                       values='Price', aggfunc='mean').round(2)
print(pivot_table_avg_price)
```

```
Product_Category   Dress  Jacket   Jeans   Shoes  Sweater  T-shirt
Brand
Adidas            113.37  100.46   92.10  111.96    84.79   122.67
New Balance       127.67  110.33  114.75  103.35   123.32   115.46
Nike               97.68  102.30  106.45  101.25   103.61   100.96
Puma              121.33  121.19   97.79  117.78    84.37    89.54
Reebok            107.08   93.61  122.54  102.63   116.92    85.90
Under Armour      120.27   88.91   81.93  113.00   125.22   109.48
```

- Pivot table showing the total count of products of each brand in each category

In [62]:
```
pivot_table_count = df.pivot_table(index='Brand',
                                   columns='Product_Category',
                                   values='Price', aggfunc='count')
print(pivot_table_count)
```

```
Product_Category  Dress  Jacket  Jeans  Shoes  Sweater  T-shirt
Brand
Adidas               30      28     39     25       19       24
New Balance          27      33     24     31       22       26
Nike                 22      30     22     32       31       27
Puma                 21      32     28     36       27       24
Reebok               36      23     24     19       36       20
Under Armour         30      44     30     28       23       23
```

- Count of Items by Category and Size

In [63]:
```
pivot_table_item_count = df.pivot_table(index='Product_Category',
                                        columns='Size', values='Price',
                                        aggfunc='count')
pivot_table_item_count
```

Out[63]:

Size	L	M	S	XL	XS	XXL
Product_Category						
Dress	27	21	27	25	41	25
Jacket	27	25	38	35	31	34
Jeans	24	24	20	24	37	37
Shoes	21	32	31	26	36	26
Sweater	26	27	26	25	26	29
T-shirt	15	28	24	32	23	22