Introduction to Pandas Library

- Pandas is an open source library in python which is know for its rich applications and utilities for all kinds of mathematical, financial and statistical functions
- It is useful in data manipulation and analysis
- It provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data

Installing pandas

In [ ]:
```
!pip install pandas
```

Importing pandas

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Series

Series -

- are one-dimensional ndarray with axis labels (homogenous data)
- labels need not be unique but must be of immutable type

Creating Series

Ex. Create series using the given list of names

In [ ]:
```
names = pd.Series(["Jack", "Jane", "George"])
names
```

In [ ]:
```
names.index  # Labels of the series
```

In [ ]:
```
names.values # Values in the series
```

Note - An ordered sequence eg - list, tuple, dict, array can only be converted into a series

In [ ]:
```
salaries = pd.Series(np.random.randint(30000, 60000, 5))
salaries
```

Assign names as labels

In [ ]:
```
salaries.index = ["Jane", "Jack", "George", "Rosie", "Dori"]
salaries
```

In [ ]:
```
# Assigning labels while creating series
salaries = pd.Series(np.random.randint(30000, 60000, 5), index = ["Jane", "Jack", "
```

Extracting elements from series

Indexing

In [ ]:
```
salaries.iloc[1]  # indexing based on index position
```

In [ ]:
```
salaries["Jane"]  # extracting value based on labels
```

Slicing

In [ ]:
```
salaries.iloc[0:3]
```

In [ ]:
```
salaries["Jane" : "George"]
```

Conditional Indexing

In [ ]:
```
salaries[salaries > 50000]
```

Operations on Series

Ex. Increment the salaries by 10%

In [ ]:
```
salaries * 1.10
```

Ranking and Sorting

- series.sort_values(`ascending=True` , `inplace=False` , `na_position = {"first","last"}`)
- series.sort_index(`ascending=True` , `inplace=False`)
- series.rank(`ascending=False` , `method={"average","min","dense"}` , `na_option = {"top","bottom"}`)

In [ ]:
```
salaries = pd.Series(np.random.randint(30000, 60000, 5), index = ["Jane", "Jack", "
salaries = pd.concat((salaries, pd.Series([np.nan, np.nan], index = ["Janet", "Sam"
salaries
```

Ex. Sort by values

In [ ]:
```
salaries.sort_values(ascending=False, na_position="first", ignore_index=True)
```

Ex. Sort by index

In [ ]:
```
salaries.sort_index(ascending=False)
```

In [ ]:
```
salaries
```

Ex. Rank the series

In [ ]:
```
salaries.rank(method="min", na_option="bottom", ascending=False).astype(int)
```

In [ ]:
```
salaries
```

In [ ]:
```
marks = pd.Series([80, 90, 80, 70, 60, 60, 50])
marks
```

In [ ]:
```
marks.rank(ascending=False, method="min").astype(int)
```

Note - to modify original series/dataframe inplace can be set to True

Working with NULLs

In [ ]:
```
salaries.isna()
```

In [ ]:
```
salaries.isna().sum()
```

In [ ]:
```
salaries.isna().any()
```

In [ ]:
```
salaries.isna().all()
```

In [ ]:
```
salaries.fillna(0)
```

In [ ]:
```
salaries.ffill()
```

In [ ]:
```
salaries.bfill()
```

Dataframe

A DataFrame is two dimensional data structure where the data is arranged in the tabular format in rows and columns

DataFrame features:

- Columns can be of different data types
- Size of dataframe can be changes
- Axes(rows and columns) are labeled

- Arithmetic operations can be performed on rows and columns

Creating Dataframes

In [ ]:
```
employees = {"Name" : ["Jack", "Bill", "Lizie", "Jane", "George"],
             "Designation" : ["HR", "Manager", "Developer", "Intern", "Manager"],
             "Salary": [40000, 60000, 25000, 12000, 70000]}

df = pd.DataFrame(employees)
df
```

Accessing Dataframes

In [ ]:
```
df["Name"]
```

In [ ]:
```
df.Name
```

Operations on dataframes

Ex. Average Salary

In [ ]:
```
df.Salary.mean()
```

Ex. Average Salary of managers

In [ ]:
```
df[df.Designation == "Manager"]
```

Concataneting and Merging Dataframes

In [ ]:
```
df_jan = pd.DataFrame({"Order ID" : range(101, 111), "Sales" : np.random.randint(10
df_feb = pd.DataFrame({"Order ID" : range(111, 121), "Sales" : np.random.randint(10
df_mar = pd.DataFrame({"Order ID" : range(121, 131), "Sales" : np.random.randint(10
```

Concatenate

pd.concat(`tuple of dfs` , `ignore_index = False` , `axis=0`)

In [ ]:
```
pd.concat((df_jan, df_feb, df_mar), ignore_index=True)
```

In [ ]:
```
pd.concat((df_jan, df_feb, df_mar), axis=1)
```

Merging Dataframes

`df1.merge(df2, how="", left_on="", right_on="", left_index= "" , right_index="")`

In [ ]:
```
df_emp = pd.DataFrame({"Name" : ["Jack", "Bill", "Lizie", "Jane", "George"],
            "Designation" : ["HR", "Manager", "Developer", "Intern", "Manager"]})
df_emp
```

In [ ]:
```
base_salaries = pd.DataFrame({"Post" : ["HR", "Developer", "Manager", "Senior Manag
            "Salary": [40000, 25000, 70000, 1000000]})
base_salaries
```

Inner Merge - returns rows present in both tables

In [ ]:
```
df_emp.merge(base_salaries, how="inner", left_on="Designation", right_on="Post") #
```

Left Merge - returns data from left table and corresponding data from right, returns NAN for non matching values

In [ ]:
```
df_emp.merge(base_salaries, how="left", left_on="Designation", right_on="Post") # u
```

Right Merge

In [ ]:
```
df_emp.merge(base_salaries, how="right", left_on="Designation", right_on="Post") #
```

Outer Merge

In [ ]:
```
df_emp.merge(base_salaries, how="outer", left_on="Designation", right_on="Post") #
```

Reading data from Data Sources

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Reading data from MYSQL or SQLITE3

In [ ]:
```
pip install sqlalchemy
```

Syntax for MSSQL

connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}"

In [ ]:
```
from sqlalchemy import create_engine
conn = create_engine(r"sqlite:///employee.sqlite3")
```

In [ ]:
```
conn
```

In [ ]:
```
pd.read_sql("Employee", conn)
```

In [ ]:
```
df = pd.read_sql_query("Select * from Employee where Designation = 'Manager'", conn
df
```

In [ ]:
```
df.to_sql("Employee", conn, if_exists="replace")
```

Examples using Coffee Shop Dataset

Ex. Connecting to Excel File

In [ ]:
```
pd.read_excel("filename.xlsx", sheet_name="sheet_name")  # demo syntax
```

Note - may generate ModuleNotFoundError - openpyxl.

pip install openpyxl

Ex. Read data from `coffee_sales.csv`

In [ ]:
```
df = pd.read_csv("coffee_sales.csv")
df
```

Cleaning DataFrame

Approach 1

pd.read_csv("coffee_sales.csv", header=3, usecols=[1, 5, 6], skiprows=range(4248, 4253))

In [ ]:
```
df = pd.read_csv("coffee_sales.csv", header=3)  # Read Data
df
```

In [ ]:
```
df.columns # get column names
```

In [ ]:
```
df.drop(columns= ['Unnamed: 0'], inplace=True)  # Drop column by name
df
```

Approach 2

In [ ]:
```
df = pd.read_csv("coffee_sales.csv")  # Read Data
df
```

df.dropna(`axis = 0` , `how = "any"` , `inplace = False`)

- axis 0 for row or 1 for column
- how - {any or all}

In [ ]:
```
# Remove null rows
df.dropna(how="all", axis=0, inplace=True) # Delete a row with all null value - rem
df.columns = df.iloc[0]
```

```
In [ ]: # Remove null cols
        df.dropna(axis=1, how="all", inplace=True)
```

```
In [ ]: df = df.iloc[1:]  # using slicing to extract all rows except first
```

```
In [ ]: df.reset_index(drop= True, inplace=True)
```

Rename Column Headers

Ex. Rename all the column names

```
In [ ]: headers = ["Year/Month", "ShopID", "Product", "City", "Sales", "Profit", "Target Pr
        df.columns = headers
        df
```

Ex. Rename Single Column

```
In [ ]: df.rename({"ShopID" :"Franchise"}, inplace=True, axis=1)  # axis = 1 for columns
        df
```

Understanding Data in Dataframe

- `df.shape` - gives the size of the dataframe in the format (row_count x column_count)
- `df.dtypes` - returns a Series with the data type of each column
- `df.info()` - prints information about a DataFrame including the index dtype and columns, non-null values and memory usage
- `df.head()` - prints the first 5 rows of you dataset including column header and the content of each row
- `df.tail()` - prints the last 5 rows of you dataset including column header and the content of each row

```
In [ ]: df.shape
```

```
In [ ]: df.dtypes
```

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

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

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

```
In [ ]: df.tail()
```

```
In [ ]: df.tail(3)
```

Working with null values

`df.isna()` - Detect missing values. Return a boolean same-sized object indicating if the values are NA.

`df.fillna(value=None, inplace=False)` - Fill NA/NaN values using the specified method.

In [ ]:
```
df.isna().sum()
```

In [ ]:
```
df.dropna()  # Drop rows with null value - loss of data
```

In [ ]:
```
df.fillna({"Target Profit" : "0"}, inplace=True) # modified syntax for new version
df.head(2)
```

Convert string columns to integers

In [ ]:
```
trans_obj = str.maketrans("", "", "$,")
df.Sales = df.Sales.str.translate(trans_obj).astype(float)
df.Profit = df.Profit.str.translate(trans_obj).astype(float)
df["Target Sales"] = df["Target Sales"].str.translate(trans_obj).astype(float)
df["Target Profit"] = df["Target Profit"].str.translate(trans_obj).astype(float)
```

Ex. Total Sales

In [ ]:
```
df.Sales.sum()
```

Ex. Total Sales for Caffe Latte

In [ ]:
```
df[df.Product == "Caffe Latte"].Sales.sum()
```

Ex. Create a new column to check the target status and visualised the performance

In [ ]:
```
df["Sales Target Status"] = np.where(df.Sales >= df["Target Sales"], "Achieved", "N
df.head(2)
```

Ex. Frequency counts - works for any categorial column

In [ ]:
```
df["Sales Target Status"].value_counts()  # Gives the Frequency counts
```

In [ ]:
```
result = (df["Sales Target Status"].value_counts(normalize=True) * 100).round(2)
result
```

Visualise the frequency

In [ ]:
```
# using pandas
result = (df["Sales Target Status"].value_counts(normalize=True) * 100).round(2)
result.plot(kind = "bar")
plt.show()
```

In [ ]:
```
# using seaborn
sns.countplot(data = df, x="Sales Target Status")
plt.show()
```

In [ ]:
```
# using seaborn
plt.figure(figsize=(10, 2))
sns.countplot(data = df, hue="Sales Target Status", x = "Product")
plt.xticks(size = 6, rotation = 10)
plt.yticks(size = 6)
plt.show()
```

Setting and Resetting Index

Seting Index

`df.set_index(keys, drop=True, inplace=False,)` - Set the DataFrame index (row labels) using one or more existing columns or arrays (of the correct length). The index can replace the existing index or expand on it.

Resetting Index

`df.reset_index(level=None, drop=False, inplace=False,)` - Reset the index of the DataFrame, and use the default one instead. If the DataFrame has a MultiIndex, this method can remove one or more levels.

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

In [ ]:
```
# set index
df_label = df.set_index("Franchise")
df_label.head()
```

In [ ]:
```
# reset index
df_label.reset_index()  # demo data not modifed
```

Indexing and Slicing using loc and iloc

Using loc to retrive data

- loc is label-based
- specify the name of the rows and columns that we need to filter out

Ex. Extract data for M1

In [ ]:
```
df_label.loc["M1"]
```

Ex. Extract data for M1, M2, M3

In [ ]:
```
df_label.loc[["M1", "M2", "M3"]]
```

Ex. Extract sales and Profit data for M1, M2, M3

In [ ]:
```
df_label.loc[["M1", "M2", "M3"], ["Sales", "Profit"]]
```

Using iloc to retrive data

- iloc is integer index-based
- specify rows and columns by their integer index.

Ex. Extract first 5 rows

```
In [ ]: df_label.iloc[0:5]
```

Ex. Extract first 5 rows and first 3 columns

```
In [ ]: df_label.iloc[0:5, 0:3]
```

Group By

`df.groupby(by=None, as_index=True, sort=True, dropna=True)`

Ex. Display total sales by product using bar chart

```
In [ ]: df_products = df.groupby("Product")["Sales"].sum()
        df_products
```

```
In [ ]: # using pandas
        df_products.plot(kind = "bar")
        plt.show()
```

```
In [ ]: df.groupby("Product")[["Sales", "Profit"]].sum().sort_values("Sales", ascending = F
        plt.show()
```

```
In [ ]: # using seaborn
        plt.figure(figsize=(10, 2))
        sns.barplot(data=df, x = "Product", y="Sales", estimator="sum", errorbar=None, hue=
        plt.xticks(size = 6, rotation = 10)
        plt.yticks(size = 6)
        plt.show()
```

Ex. Plot correlation between Sales and Profit

```
In [ ]: sns.scatterplot(data=df, x = "Sales", y= "Profit")
        plt.show()
```

```
In [ ]: sns.lmplot(data=df, x = "Sales", y= "Profit")
        plt.show()
```

Working on Data Column

```
In [ ]: df.Date = pd.to_datetime(df.Date, format="mixed")
```

Ex. Create a line chart top display sales over months and years

```
In [ ]: plt.figure(figsize=(10, 2))
        sns.lineplot(data = df, x="Date", y = "Sales", estimator="sum", errorbar = None)
        plt.show()
```

```
In [ ]: plt.figure(figsize=(10, 2))
        sns.lineplot(data = df, x="Date", y = "Sales", estimator="sum", errorbar = None, hu
        plt.show()
```

Ex. Extract Year and Month as new columns

```
In [ ]: df.insert(1, "Year", df.Date.dt.year)
        df.head(2)
```

```
In [ ]: df.insert(2, "Month", df.Date.dt.month_name())
        df.head(2)
```

Ex. Calculate year wise sales

```
In [ ]: df.groupby("Year")["Sales"].sum()
```

Ex. Visulise Year wise sales

```
In [ ]: sns.barplot(df, x = "Year", y = "Sales")
```

Ex. Calculate year and month wise sales (applicable only in groupby scenarios)

```
In [ ]: df.insert(3, "Month#", df.Date.dt.month)
        df.head(2)
```

Ex. Save transformed data to csv file

```
In [ ]: result = df.groupby(["Year", "Month", "Month#"])[["Sales", "Profit"]].sum().reset_i
        result.sort_values(["Year", "Month#"], inplace=True)
        result.to_csv("Monthy_data.csv", index=False)
```

Ex. Save image as png

```
In [ ]: plt.figure(figsize=(10, 2))
        sns.lineplot(data = df, x="Date", y = "Sales", estimator="sum", errorbar = None, hu
        plt.savefig("linechart.png")
```

Final Approach

```
In [ ]: # All imports
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns

        # Default settings
        plt.rcParams["figure.figsize"] = (3, 2)
```

```python
df = pd.read_csv("coffee_sales.csv", header=3)  # Read Data
df.dropna(how="all", inplace=True)  # Remove null rows
df.dropna(axis= 1, how="all", inplace=True)  # Remove null cols
df.fillna({"Target Profit" : "0"}, inplace=True) # Replace null with default

# Cleaning data - converting str cols to float
trans_obj = str.maketrans("", "", "$,")
df.Sales = df.Sales.str.translate(trans_obj).astype(float)
df.Profit = df.Profit.str.translate(trans_obj).astype(float)
df["Target Sales"] = df["Target Sales"].str.translate(trans_obj).astype(float)
df["Target Profit"] = df["Target Profit"].str.translate(trans_obj).astype(float)
df.Date = pd.to_datetime(df.Date, format="mixed")

df["Sales Target Status"] = np.where(df.Sales >= df["Target Sales"], "Achieved", "N

df.head(2)
```

Analysing Dataframes

- univariate analysis - boxplot, histogram, value_counts(), countplot, describe()
- bivariate analysis
  - categorial X numerical - barchart, piechart
  - 2 numerical - scatter plot
  - 2 categorial - crosstab
- multivariate - pivot table

Univariate Analysis

- Numeric Columns
  - df.describe()
  - historgam
  - boxplot - outlier analysis
- Categorial Column
  - value_counts()
  - df["col"].unique() - discrete values in a column

`df.value_counts(normalize = False)` **- returns a Series containing counts of unique rows in the DataFrame**

In [222…

```python
df.Product.unique()
```

Out[222…

```
array(['Amaretto', 'Caffe Latte', 'Caffe Mocha', 'Chamomile', 'Colombian',
       'Darjeeling', 'Decaf Espresso', 'Decaf Irish Cream', 'Earl Grey',
       'Green Tea', 'Lemon Tea', 'Mint Tea', 'Regular Espresso'],
      dtype=object)
```

Summary Statistics

`df.describe()` - Generates descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values. Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided.

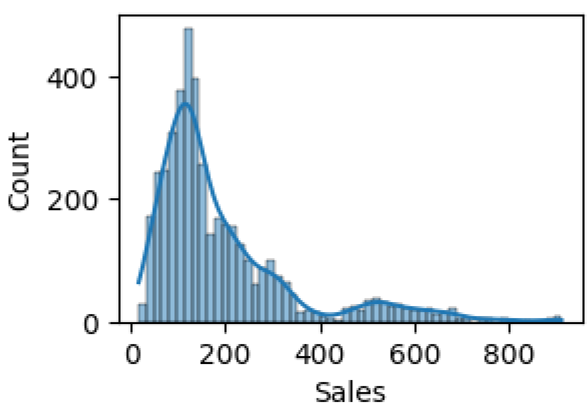
In [221…
```
df[["Sales", "Profit"]].describe()
```

Out[221…

	Sales	Profit
count	4248.000000	4248.000000
mean	192.987524	61.097693
std	151.133127	101.708546
min	17.000000	-638.000000
25%	100.000000	17.000000
50%	138.000000	40.000000
75%	230.000000	92.000000
max	912.000000	778.000000

Histogram

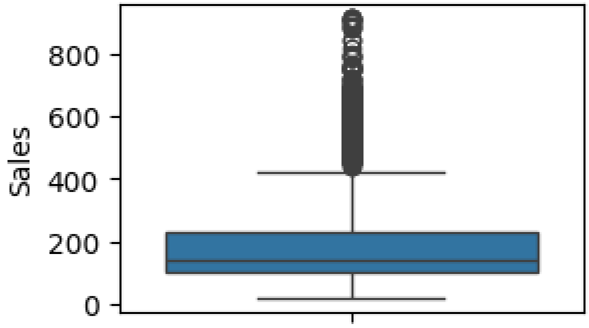
In [224…
```
sns.histplot(df, x = "Sales", kde = True)
plt.show()
```



Box and Whisker Plot

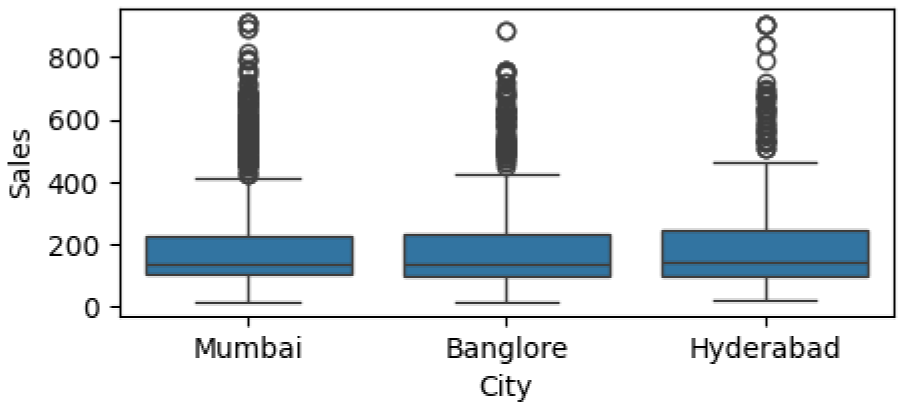
- Box and Whisker chart uses IQR technique to calculate outliers

In [227…
```
sns.boxplot(df, y = "Sales")
plt.show()
```



In [239…

```
plt.figure(figsize=(5, 2))
sns.boxplot(df, y = "Sales", x = "City")
plt.show()
```



In [228…

```
df.Sales.describe()
```

Out[228…

```
count    4248.000000
mean      192.987524
std       151.133127
min        17.000000
25%       100.000000
50%       138.000000
75%       230.000000
max       912.000000
Name: Sales, dtype: float64
```

In [231…

```
Q1 = 100
Q3 = 230
IQR = Q3 - Q1
min_w = Q1 - 1.5 * IQR
max_w = Q3 + 1.5 * IQR
```

In [240…

```
df[df.Sales > max_w]
```

Out[240…

	Date	Year	Month	Month#	Franchise	City	Product	Sales	Profit	Target Profit
201	2025-10-01	2025	October	10	M3	Mumbai	Amaretto	567.0	291.0	290.0
217	2021-02-01	2021	February	2	M1	Mumbai	Caffe Latte	456.0	140.0	150.0
226	2021-11-01	2021	November	11	M1	Mumbai	Caffe Latte	457.0	142.0	150.0
235	2022-08-01	2022	August	8	M1	Mumbai	Caffe Latte	478.0	149.0	150.0
244	2023-05-01	2023	May	5	M1	Mumbai	Caffe Latte	478.0	148.0	150.0
...	...	...	...	...	...	...	...	...	...	...
4234	2025-11-01	2025	November	11	M1	Mumbai	Regular Espresso	538.0	247.0	190.0
4237	2026-02-01	2026	February	2	M1	Mumbai	Regular Espresso	604.0	332.0	240.0
4240	2026-05-01	2026	May	5	M1	Mumbai	Regular Espresso	815.0	646.0	450.0
4243	2026-08-01	2026	August	8	M1	Mumbai	Regular Espresso	719.0	565.0	390.0
4246	2026-11-01	2026	November	11	M1	Mumbai	Regular Espresso	700.0	463.0	320.0

406 rows × 12 columns

Bivariate Analysis

- groupby()
- Bar, line, scatter

`pd.crosstab(index, columns, values=None, aggfunc=None normalize=False)` - **Computes a simple cross tabulation of two (or more) factors. By default computes a frequency table of the factors unless an array of values and an aggregation function are passed.**

In [245…

```
df_emp = pd.read_csv("employees.csv")
df_emp.head(2)
```

Out[245...

	Name	Salary	Designation	Age	Gender	Owns Car
0	Claire	88962	Manager	35	Female	Yes
1	Darrin	67659	Team Lead	26	Male	No

In [252...

```
pd.crosstab(index = df_emp["Gender"], columns = df_emp["Owns Car"])
```

Out[252...

Owns Car	No	Yes
Gender		
Female	2	7
Male	8	13

`df.pivot_table(values=None, index=None, columns=None, aggfunc='mean')` **- creates a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.**

Ex. Total Sales per product in each city

In [254...

```
df.pivot_table(index="City", columns="Product", values="Sales", aggfunc="sum")
```

Out[254...

Product	Amaretto	Caffe Latte	Caffe Mocha	Chamomile	Colombian	Darjeeling	Decaf Espresso	De I Cre
City								
Banglore	NaN	11923.0	25079.0	28726.0	37735.0	27123.0	28487.0	238
Hyderabad	NaN	NaN	13866.0	NaN	21644.0	14463.0	13193.0	I
Mumbai	30425.0	23976.0	37523.0	42168.0	57168.0	27202.0	40762.0	345

Ex. Number of Franchises in each city where the product is sold

In [259...

```
df.pivot_table(columns="Product", index="City", values="Franchise", aggfunc="nuniqu
```

Out[259...

Product	Amaretto	Caffe Latte	Caffe Mocha	Chamomile	Colombian	Darjeeling	Decaf Espresso	Decaf Irish Cream
City								
Banglore	0	1	2	2	2	2	2	2
Hyderabad	0	0	1	0	1	1	1	0
Mumbai	3	2	3	3	3	2	3	3

In [ ]:
```
with open("filename.txt", "w") as file :
    file.write(string)
```

In [ ]:
```
import CSV
```

In [ ]:
```
import openpyxl
```

In [ ]:
```

```