

Exploratory Data Analysis (EDA) on Restaurant Dataset

✓ What is Pandas?

Pandas is a powerful Python library used for data manipulation and analysis. It provides DataFrames, which allow us to store, clean, and analyze structured data efficiently, similar to an Excel spreadsheet.

✓ 1. Introduction & Setup

✓ Importing Pandas with an Alias

```
import pandas as pd # 'pd' is a commonly used alias for Pandas, making code shorter and readable.
```

✓ What is Aliasing?

Aliasing in Python means assigning a short name to a library to make coding easier. For example, instead of writing `pandas.read_csv()`, we use `pd.read_csv()` after importing Pandas as `import pandas as pd`.

Start coding or [generate](#) with AI.

```
!gdown "https://drive.google.com/uc?export=download&id=1in_bJYfJ5Ahy8y9RcBhdqp0aKHguU7EX" -O dummy_restaurants.csv
```



Downloading...

From: https://drive.google.com/uc?export=download&id=1in_bJYfJ5Ahy8y9RcBhdqp0aKHguU7EX

To: /content/dummy_restaurants.csv

100% 1.50k/1.50k [00:00<00:00, 4.93MB/s]

Double-click (or enter) to edit

✓ Data Loading using Pandas

```
# Loading data using Pandas (recommended)
df = pd.read_csv('dummy_restaurants.csv')

# Display df
df
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
0	Gabbar's Bar & Kitchen	Kolkata	4.3	1500	North Indian, Chinese, Mexican, Italian	False	True
1	Sardaar Ji De Chole Bhature	Delhi NCR	3.6	100	North Indian	True	False
2	YellowKrust	Delhi NCR	3.8	400	Bakery, Desserts	False	False
3	Hungry Minister	Delhi NCR	3.7	400	Continental, Mexican, Fast Food, Chinese	True	False
4	Night Food Xprs	Delhi NCR	3.3	500	North Indian, Chinese	True	False
5	HOP House of Proteins	Delhi NCR	4.8	600	Continental, North Indian, Healthy Food, Bever...	True	False
6	Daana Paani Family Restaurant	Kolkata	3.4	300	North Indian, Chinese	True	False
7	Swad Restaurant	Delhi NCR	3.8	750	North Indian, Chinese, Mughlai	True	False
8	Chai Peeni Hai	Delhi NCR	3.7	200	Tea, Fast Food	True	False
9	Baskin Robbins	Mumbai	3.8	200	Ice Cream, Desserts, Beverages	False	False
10	Spirit	Mumbai	NaN	850	Chinese, North Indian, Mughlai	NaN	NaN
11	Purple Box	Mumbai	3.9	200	Fast Food, Beverages, Desserts	True	False
12	Mio Amore	Kolkata	3.4	150	Bakery	False	False
13	Talli Pangs	Delhi NCR	3.7	300	Fast Food, North Indian	False	False
14	Just Baked	Kolkata	3.3	250	Bakery, Fast Food	False	False

Insight:

Pandas automatically structures the data into a tabular format, unlike plain Python, which reads raw text.

2. Exploring the DataFrame

Introducing the DataFrame

A DataFrame consists of rows (observations) and columns (features). Each row represents a restaurant listing, and each column provides information such as city, cost, and rating.

Common DataFrame Methods

`df.shape` # Returns (rows, columns)

(20, 7)

The `.shape` attribute in Pandas is used to get the dimensions of a DataFrame, returning a tuple with the number of rows and columns in the format (rows, columns).

`df.head()` # Displays the first 5 rows

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
0	Gabbar's Bar & Kitchen	Kolkata	4.3	1500.0	North Indian, Chinese, Mexican, Italian	False	True
1	Sardaar Ji De Chole Bhature	Delhi NCR	3.6	100.0	North Indian	True	False
2	YellowKrust	Delhi NCR	3.8	400.0	Bakery, Desserts	False	False
3	Hungry Minister	Delhi NCR	3.7	400.0	Continental, Mexican, Fast Food, Chinese	True	False

The `.head()` function in Pandas is used to return the first 5 rows of a DataFrame (by default), allowing you to quickly preview the data.

```
df.tail() # Displays the last 5 rows
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
15	Monginis	Mumbai	3.4	200.0	Bakery, Fast Food	True	False
16	The Bohri Kitchen	Mumbai	4.8	600.0	Mughlai, North Indian, Biryani	True	False
17	Italy in a Box	Mumbai	3.9	800.0	Italian, Continental	True	False
18	PizzaExpress	Mumbai	4.3	1900.0	Italian, Pizza	True	True
19	Artusi Ristorante in Piazza Horizon	Delhi NCR	4.3	3500.0	Italian	False	True

The `.tail()` function in Pandas is used to return the last 5 rows of a DataFrame (by default), allowing you to quickly preview the end of the data.

```
df.sample(5) # Displays a random sample of 5 rows
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
6	Daana Paani Family Restaurant	Kolkata	3.4	300.0	North Indian, Chinese	True	False
17	Italy in a Box	Mumbai	3.9	800.0	Italian, Continental	True	False
3	Hungry Minister	Delhi NCR	3.7	400.0	Continental, Mexican, Fast Food, Chinese	True	False
7	Swad Restaurant	Delhi NCR	3.8	750.0	North Indian, Chinese, Mughlai	True	False

The `.sample()` function in Pandas is used to randomly select a specified number or fraction of rows from a DataFrame, providing a random sample of the data.

▼ Data Types & Overview

```
df.info() # Shows column names, data types, and non-null counts
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  -
0   name                   20 non-null    object
1   city                   20 non-null    object
2   rating                 20 non-null    float64
3   cost_for_two           20 non-null    float64
4   cuisine                 20 non-null    object
5   online_order           20 non-null    bool
6   table_reservation      20 non-null    bool
dtypes: bool(2), float64(2), object(3)
memory usage: 972.0+ bytes
```

The `.info()` function in Pandas is used to display concise summary information about a DataFrame, including the number of non-null entries, data types of each column, and memory usage.

▼ Descriptive Statistics

```
df.describe() # Summary of dataset
```

	rating	cost_for_two
count	20.000000	20.000000
mean	3.845000	685.000000
std	0.446595	807.057684
min	3.300000	100.000000
25%	3.550000	200.000000
50%	3.750000	400.000000
75%	4.000000	762.500000
max	4.800000	3500.000000

The `.describe()` function in Pandas is used to generate descriptive statistics of the DataFrame, such as count, mean, standard deviation, minimum, and maximum values for columns.

count - The number of not-empty values. in a col (eg rating)

mean - The average (mean) value of all values in a col (eg rating).

std - The standard deviation.

min - the minimum value in a col

25% - The 25% percentile.

50% - The 50% percentile

75% - The 75% percentile

max - the maximum value in a column.

In `df.info` there were 7 columns then why in `df.describe` shows data for two columns only?

What about categorical data? would you like to implement `df.describe` on categorical?

```
df.describe(include=['object']) # Summary of categorical columns
```



	name	city	cuisine
count	20	20	20
unique	20	3	18
top	Gabbar's Bar & Kitchen	Delhi NCR	North Indian, Chinese
freq	1	9	2

The `.describe(include=['object'])` function in Pandas generates descriptive statistics specifically for categorical (object-type) columns, such as count, unique values, top frequency, and most common value. By default, `.describe()` only includes numeric columns, but using `include=['object']` includes non-numeric (text) columns as well.

The metrics are as follows:

count – The total number of non-null values in the column.

unique – The number of unique categories in the column.

top – The most frequently occurring category.

freq – The count of occurrences of the most frequent category.

```
df.describe(include="all")
```

✓ 3. Working with Columns

✓ Selecting Columns

```
df['city'] # Selecting a single column
```

**city**

0	Kolkata
1	Delhi NCR
2	Delhi NCR
3	Delhi NCR
4	Delhi NCR
5	Delhi NCR
6	Kolkata
7	Delhi NCR
8	Delhi NCR
9	Mumbai
10	Mumbai
11	Mumbai
12	Kolkata
13	Delhi NCR
14	Kolkata
15	Mumbai
16	Mumbai
17	Mumbai
18	Mumbai
19	Delhi NCR

dtype: object

```
df['city', 'rating']
```

If we select a single column like `df['city']`, it works fine because Pandas treats it as a valid column name and returns a Series. However, when selecting multiple columns, `df['city', 'rating']` throws an error because Pandas treats `('city', 'rating')` as a tuple, not a list. To correctly select multiple columns, we must use double brackets: `df[['city', 'rating']]`

```
df[['city', 'rating']] # Selecting multiple columns
```




	city	rating
0	Kolkata	4.3
1	Delhi NCR	3.6
2	Delhi NCR	3.8
3	Delhi NCR	3.7
4	Delhi NCR	3.3
5	Delhi NCR	4.8
6	Kolkata	3.4
7	Delhi NCR	3.8
8	Delhi NCR	3.7
9	Mumbai	3.8
10	Mumbai	3.7
11	Mumbai	3.9
12	Kolkata	3.4
13	Delhi NCR	3.7
14	Kolkata	3.3
15	Mumbai	3.4
16	Mumbai	4.8
17	Mumbai	3.9
18	Mumbai	4.3
19	Delhi NCR	4.3

▼ ◆ Basic Functions on Columns

```
df['cost_for_two'].mean() # Mean of 'cost_for_two'
```

 685.0

```
df['rating'].min(), df['rating'].max() # Min and max rating
```

 (3.3, 4.8)

```
df["rating"].min(), df["rating"].max(), round(df["rating"].std(), 2)
```

```
df['city'].value_counts() #frequency of every unique categories
```



	count
city	
Delhi NCR	9
Mumbai	7
Kolkata	4

dtype: int64

```
df['city'].unique() #no.of unique categories
```

 array(['Kolkata', 'Delhi NCR', 'Mumbai'], dtype=object)

▼ ◆ Creating or Modifying Columns

```
df['ROI'] = df['rating'] / df['cost_for_two'] # Creating a new column ROI
```

```
df['Expensive'] = df['cost_for_two'] >= 1000 # Creating a categorical column
```

```
df['Expensive']
```

How can we check the datatypes of all the columns?

```
print(df.dtypes)
```

```
name          object
city          object
rating        float64
cost_for_two  float64
cuisine       object
online_order  bool
table_reservation  bool
Expensive     bool
dtype: object
```

✓ 5. Working with Rows

✓ Row Operations

```
df[df['city'] == 'Mumbai'] # Filtering restaurants in Mumbai
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation	ROI	Expensive
9	Baskin Robbins	Mumbai	3.8	200.0	Ice Cream, Desserts, Beverages	False	False	0.019000	False
10	Spirit	Mumbai	3.7	850.0	Chinese, North Indian, Mughlai	True	True	0.004353	False
11	Purple Box	Mumbai	3.9	200.0	Fast Food, Beverages, Desserts	True	False	0.019500	False
15	Monginis	Mumbai	3.4	200.0	Bakery, Fast Food	True	False	0.017000	False
16	The Bohri Kitchen	Mumbai	4.8	600.0	Mughlai, North Indian, Biryani	True	False	0.008000	False

```
df[(df['city'] == 'Delhi NCR') & (df['cost_for_two'] >= 500)] # Filtering with multiple conditions
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation	ROI	Expensive
4	Night Food Xprs	Delhi NCR	3.3	500.0	North Indian, Chinese	True	False	0.006600	False
5	HOP House of Proteins	Delhi NCR	4.8	600.0	Continental, North Indian, Healthy Food, Bever...	True	False	0.008000	False
7	North Indian, Chinese	Delhi	3.3	500.0	North Indian, Chinese	True	False	0.006600	False

```
#Filter restaurants in Mumbai with rating >4
df[(df['city'] == 'Mumbai') & (df['rating']>4)]
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation	Expensive
16	The Bohri Kitchen	Mumbai	4.8	600.0	Mughlai, North Indian, Biryani	True	False	False
18	PizzaExpress	Mumbai	4.3	1900.0	Italian, Pizza	True	True	True

```
#restaurants who offer only italian cuisine
df[df['cuisine']=='Italian']
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
19	Artusi Ristorante in Piazza Horizon	Delhi NCR	4.3	3500.0	Italian	False	True

```
#restaurants who also offers italian cuisine, Italian is present in the string
df[df['cuisine'].str.contains('Italian')]
```

	name	city	rating	cost_for_two	cuisine	online_order	table_reservation
0	Gabbar's Bar & Kitchen	Kolkata	4.3	1500.0	North Indian, Chinese, Mexican, Italian	False	True
17	Italy in a Box	Mumbai	3.9	800.0	Italian, Continental	True	False
18	PizzaExpress	Mumbai	4.3	1900.0	Italian, Pizza	True	True
19	Artusi Ristorante in Piazza	Delhi	4.3	3500.0	Italian	False	True

The `.str.contains()` function in Pandas is used to check if each element in a column (Series) contains a specified substring or pattern. It's a very useful method for filtering data or identifying rows that match a particular pattern in text data.

```
#number of restaurants who also offer italian cuisine
df[df['cuisine'].str.contains('Italian')].shape[0]
```

↔ 4

Now can you tell the difference between using `'=='` operator and `.str.contains()`?

The `==` operator checks for an exact match in a column, meaning only rows with exactly the same string are returned (case-sensitive).

In contrast, `.str.contains()` checks if a substring exists anywhere within the column, allowing partial matches and case-insensitive searches (if specified with `case=False`), making it more flexible

✓ In Pandas, there are two main ways to access data in a DataFrame:

By label (column name)

By position (row index)

◆ Row Operations with `iloc`

✓ 📌 What is `iloc`?

`iloc` stands for integer-location-based indexing. It allows selecting rows and columns using index numbers instead of names.

```
df.iloc[3] # Selecting the 4th row
```

↔

	3
name	Hungry Minister
city	Delhi NCR
rating	3.7
cost_for_two	400.0
cuisine	Continental, Mexican, Fast Food, Chinese
online_order	True
table_reservation	False
ROI	0.00925
Expensive	False

dtype: object

What if i write `df[1]`?

`df[1]` does NOT access rows.

Instead, `df[1]` tries to retrieve a column named 1.

If column 1 does not exist, it raises a `KeyError`.

```
df.iloc[2:6] # Selecting rows from 3rd to 6th
```