AN INITIATIVE OF:

science, technology & innovation
Department: Science, Technology and Innovation
REPUBLIC OF SOUTH AFRICA

CSIR
Touching lives through innovation
80th anniversary

NICIS
NATIONAL INTEGRATED CYBERINFRASTRUCTURE SYSTEM
DIRISA

Introduction to Data Exploration and Preprocessing

Miss Boitshepo Sebusho

24 June 2025

Introduction

After a problem is identified and

- defined,
- the data acquired and
- read into the appropriate platform (e.g., Google Colab, Jupyter notebook)

Next the data should be

- checked for quality,
- explored for a better understanding,
- cleaned and
- processed if needed*

01

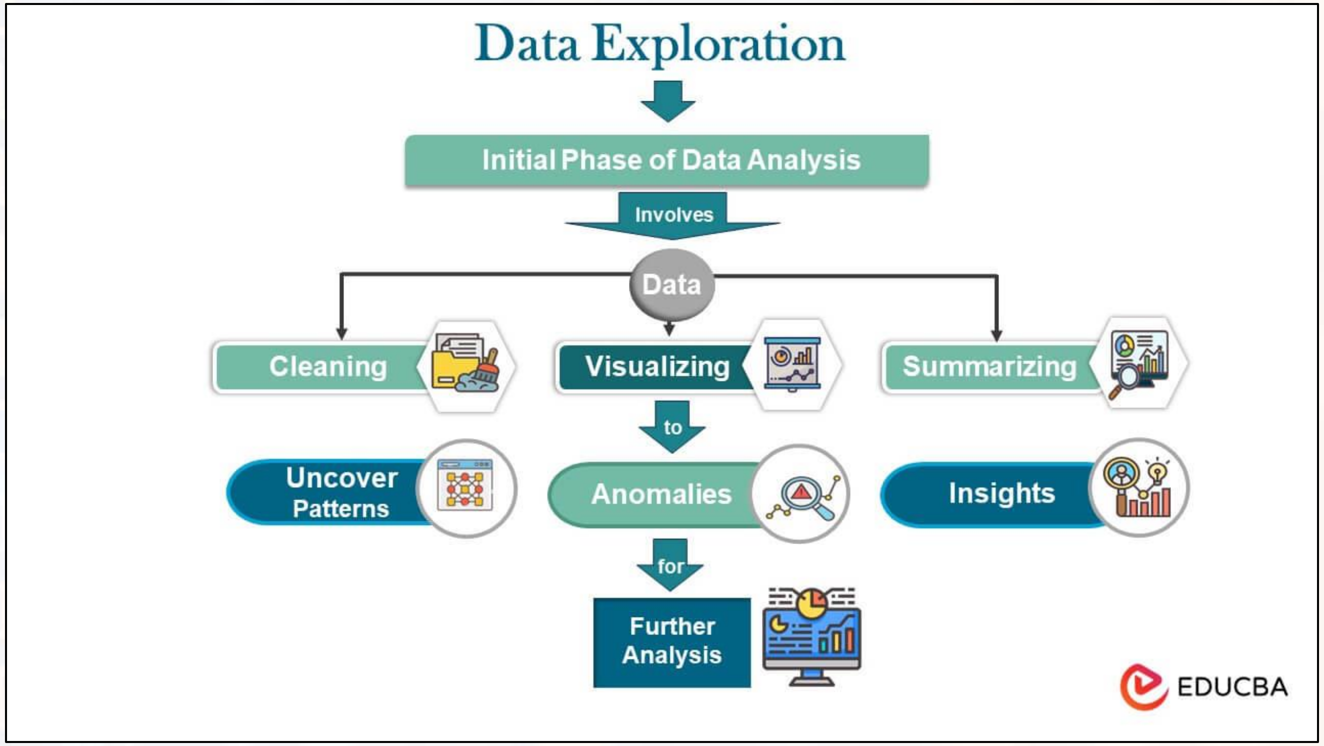
Data Exploration

What is Data Exploration

- An important step in any data analysis or machine learning project
- Involves understanding the structure, patterns, and anomalies in data
- Preliminary investigation to get a feel of the data and prepare it for further steps
- Often called **Exploratory Data Analysis (EDA)**

Goals of Data Exploration

- Understand data types and distributions
- Identify missing values or outliers
- Detect patterns or trends
- Generate initial hypotheses
- Select features for modeling



Source: [1]

Understanding Your Dataset

By Structure

	Structured Data	Unstructured Data
Description	Clearly defined fields, rows, and columns (e.g., spreadsheets, SQL databases).	No fixed format; may include text, images, audio, video.
Example	Sales records, customer tables.	Emails, social media posts, YouTube videos.
Tools	Excel, SQL, Pandas.	NLP libraries, CV models

By Source

	Primary Data	Secondary Data
Description	Collected firsthand through experiments, surveys, or sensors.	Previously collected and available through other sources.
Example	Survey responses, lab experiment results.	Open government data, public datasets (e.g., Kaggle, UCI).

Understanding Your Dataset

By Time Component

	Cross-Sectional Data	Time Series Data	Panel (Longitudinal) Data
Description	Collected at a single point in time.	Data collected over time (daily, monthly, yearly).	Multiple observations over time for the same subjects.
Example	Census data, customer data snapshot.	Stock prices, weather logs.	Tracking patients' health over several years.

Understanding Your Dataset

- It is important to know the type of data before exploration and processing because **different types** of data require **different handling**.
- Knowing the type ensures you apply the right operations, avoid errors and get meaningful insights.
- Datasets used in this presentation:
  - Housing.csv – Secondary dataset (Sourced from Kaggle)
    - A collection of housing data in California. Used to predict housing prices
  - AirPassengers.csv – Secondary dataset (Sourced from Kaggle)
    - Contains the monthly totals of internation airline passengers from January 1949 to December 1960

Examples of Tools for Data Exploration

- **Programming Languages:** Python, R
- **Python Libraries**: Pandas, Matplotlib, Seaborn, plotly, Numpy
- **R**: ggplot2, dplyr, tidyr
- **Notebooks**: Jupyter Notebooks, RStudio

1. Load Libraries and Datasets

```
import pandas as pd #For data loading, manipulation, and analysis(e.g., working with tables or DataFrames)
import seaborn as sns# For advanced statistical visualizations (built on top of matplotlib)
import matplotlib.pyplot as plt # For basic plotting and chart customization (e.g., line charts, bar plots)
```

Initial Data Examination

- If the data is in numpy array, it is beneficial to convert it to a pandas dataframe for easier exploration and manipulation.

- Convert NumPy Array to Pandas Data Frame

✅ Advantages of Using a DataFrame over a NumPy Array

Feature	NumPy Array	pandas DataFrame
Column names	❌ No support	✅ Supported
Data types per column	❌ Uniform only	✅ Heterogeneous (different types per column)
Missing values handling	🚫 Limited	✅ `NaN`, `isnull()`, `fillna()`, etc.
Descriptive statistics	⚠️ Manual via `np.mean()` etc.	✅ Easy via `.describe()`, `.mean()`, `.value_counts()`
Data selection & slicing	✅ Array-based indexing	✅ Label-based (`.loc`, `.iloc`) + flexible
Grouping and aggregation	🚫 Not built-in	✅ Powerful `.groupby()` support
File I/O (CSV, Excel, etc.)	❌ Manual parsing	✅ Built-in readers/writers

```
import numpy as np
import pandas as pd

data = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

df = pd.DataFrame(data, columns=['A', 'B', 'C'])
print(df)
```

```
   A  B  C
0  1  2  3
1  4  5  6
```

Initial Data Examination

- If data is in pandas DataFrame:
  - Read the file as a dataframe
  - Preview the first few rows of the DataFrame: **df.head()**

```
[1] import pandas as pd
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files housing.csv

- **housing.csv**(text/csv) - 1423529 bytes, last modified: 6/9/2025 - 100% done

```
Saving housing.csv to housing.csv
```

```
#Reading in and viewing the data
df = pd.read_csv('housing.csv')
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Initial Data Examination

- **df.info() -** Concise summary of the DataFrame:
  - The names of each column.
  - The number of non-null values in each column, which helps identify missing data.
  - The data type (dtype) of each column.
- **df.shape -** Return a tuple representing the shape of the DataFrame. The shape is the number of rows and columns of the DataFrame:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
#Understanding the data
print(df.shape)
```

```
(20640, 10)
```

Initial Data Examination

- Descriptive statistics: **df.describe()**
- For numeric columns:
  - **count**: Number of non-null observations.
  - **mean**: The average value.
  - **std**: Standard deviation, a measure of data dispersion.
  - **min**: Minimum value.
  - **25%**: The 25th percentile (first quartile, Q1).
  - **50%**: The 50th percentile (median, Q2).
  - **75%**: The 75th percentile (third quartile, Q3).
  - **max**: Maximum value.

```
df.describe()
```

	longitude	latitude	housing_median_age
count	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486
std	2.003532	2.135952	12.585558
min	-124.350000	32.540000	1.000000
25%	-121.800000	33.930000	18.000000
50%	-118.490000	34.260000	29.000000
75%	-118.010000	37.710000	37.000000
max	-114.310000	41.950000	52.000000

Initial Data Examination

- For Categorical Columns:
  - **count**:
    Number of non-null entries
  - **unique**:
    Number of unique values
  - **top**:
    Most frequent (mode) value
  - **freq**:
    Frequency of the top value

```python
import pandas as pd

df = pd.DataFrame({
    'City': ['Cape Town', 'Durban', 'Cape Town', 'Johannesburg', 'Durban'],
    'Category': ['A', 'B', 'A', 'A', 'C']
})

print(df.describe())
```

```
             City Category
count           5        5
unique          3        3
top     Cape Town        A
freq            2        3
```

Initial Data Examination

- Generates descriptive statistics for a specific column in a Dataframe: **df[‘column_name’].describe()**

```
[11] print(df['ocean_proximity'].describe())

count           20640
unique              5
top         <1H OCEAN
freq             9136
Name: ocean_proximity, dtype: object
```

- Returns all unique values in a column: **df[‘column_name’].unique()**

```
[12] print(df['ocean_proximity'].unique()) # all unique entries in 'ocean_proximity column

['NEAR BAY' '<1H OCEAN' 'INLAND' 'NEAR OCEAN' 'ISLAND']
```

- Returns counts of unique values in a column: **df[‘column_name’].value_counts()**

```
print(df['ocean_proximity'].value_counts())

ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

Initial Data Examination

- **df.iloc** is used for integer-location based indexing and allows you to select data based on their numerical position starting from 0

```
df.iloc[5]
```

	5
Avg. Area Income	80175.75416
Avg. Area House Age	4.988408
Avg. Area Number of Rooms	6.104512
Avg. Area Number of Bedrooms	4.04
Area Population	26748.42842
Price	1068138.074
Address	06039 Jennifer Islands Apt. 443\nTracyport, KS...

dtype: object

- **df.iloc[start:end]** returns rows between a specified range using slicing

```
df.iloc[0:6]
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms
0	79545.45857	5.682861	7.009188
1	79248.64245	6.002900	6.730821
2	61287.06718	5.865890	8.512727
3	63345.24005	7.188236	5.586729
4	59982.19723	5.040555	7.839388
5	80175.75416	4.988408	6.104512

02

Data Preprocessing / Data Cleaning

What is Data Preprocessing?

- The process of **cleaning** and **transforming raw data** to a usable format.
- By processing raw data, we turn it into useful information
- Prepares data for modeling and analysis
- **Importance of Data Preprocessing:**
  - Real world data is often messy
  - Models require numerical and clean input
  - Better preprocessing → better model performance
  - Overall, we do this to get meaningful insights for informed decision-making

Key Data Preprocessing Steps

- **Handle Missing Values**
  - Imputation (mean, median, mode)
  - Dropping rows/columns
- **Encoding Categorical Variables**
  - One-hot encoding
  - Label encoding
- **Feature Scaling**
  - Normalization
  - Standardization
- **Outlier Detection and Removal**
  - Z-Score, IQR, Visualization
- **Feature Engineering**
  - Create new features or transforming existing ones

Checking for Missing Values

- Check if there are any missing values: **df.isnull()**.

- The existence of any NaN entries in the entire dataframe can be identified: **df.isnull().values.any()**

- The rows (index) containing any NaNs can also be found: **df[df.isnull().any(axis=1)].index**

Cleaning The Data : Missing Values

```
df.isnull().sum() #checking for missing values
```

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	207
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity	0

dtype: int64

Only one column has missing values.

🧠 Think: How should we deal with this? Is it OK to fill? Should we drop?

Option 1: Drop

· ✅ Quick and simple.

· ⚠️ Risk: You lose 207 rows. Only use if data loss is acceptable.

Option 2: Fill with Median or Mean

· ✅ Keeps all rows.

· 🔍 Median is more robust than mean if outliers exist.

Cleaning The Data: Missing Values (Impute)

- Fill with Median or Mean:
  - ✅ Keeps all rows.
  - 🔍 Median is more robust than mean if outliers exist.

```
median_bedrooms = df['total_bedrooms'].median()
df['total_bedrooms'].fillna(median_bedrooms, inplace=True)
```

```
df.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

Cleaning The Data: Missing Values (Impute)

- Using simpleImputer to impute the data:

```
[ ]: from sklearn.preprocessing import StandardScaler, OneHotEncoder
     from sklearn.impute import SimpleImputer
```

```
#Handle missing values
imputer = SimpleImputer(strategy='mean')
df['age'] = imputer.fit_transform(df[['age']])
```

- Fill with unknown:

```
# Fill missing directors with "Unknown"
df['director'].fillna("Unknown", inplace=True)
```

- Forward fill:

```
# Fill missing values using forward fill
df_filled = df_missing.fillna(method='ffill')
```

Cleaning The Data: Missing Values (Drop)

- Drops rows with atleast one Nan: **df.dropna(inplace=True)**.
  - ***Inplace=True*** is used to modify the original DataFrame.
- Drops all columns with any missing values: **df.dropna(axis=1)**
- Drop rows where all elements are missing: **df.dropna(how=‘all’)**
- Removes all the ROWS with missing values in specific column:

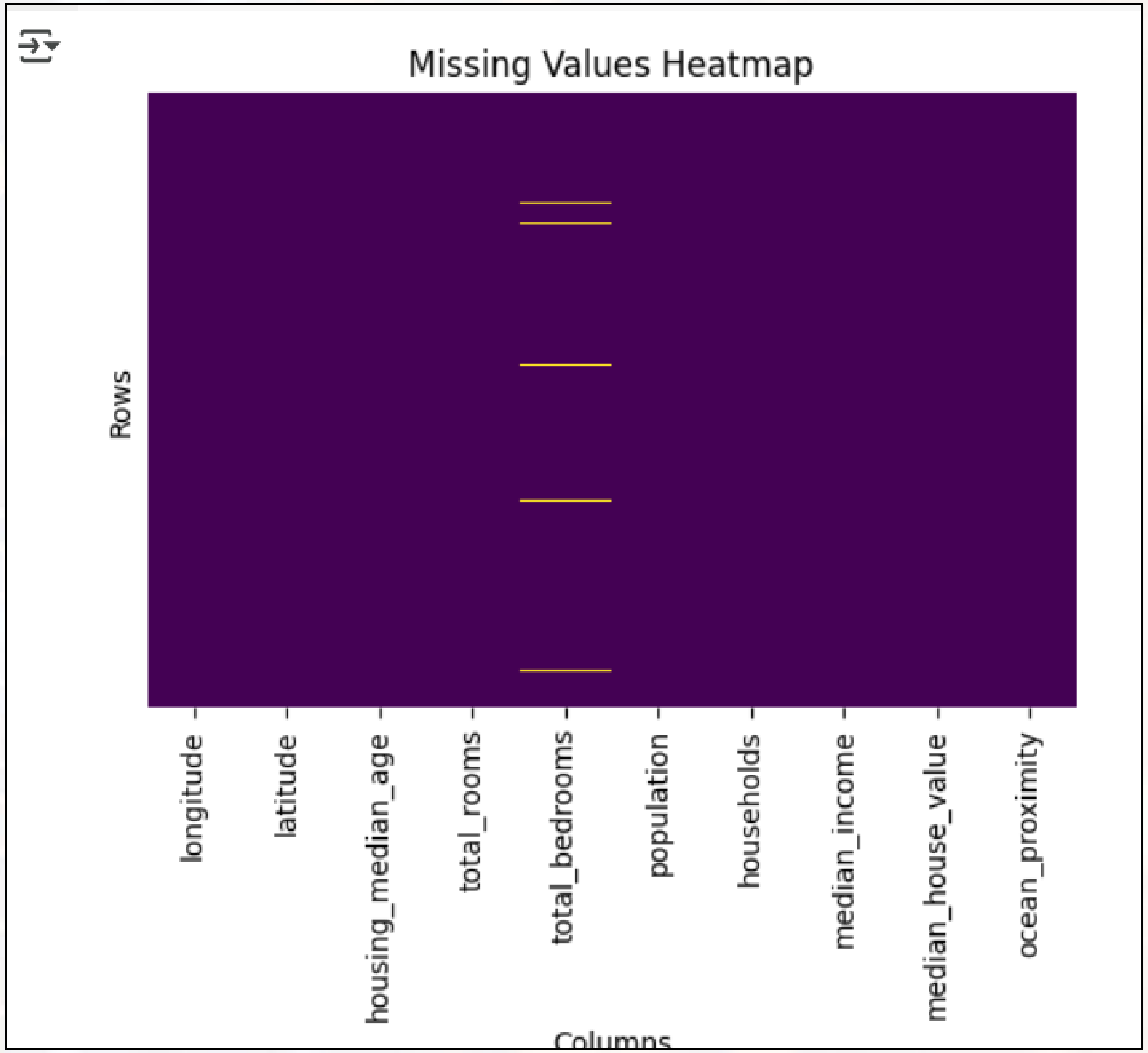
```
df.dropna(subset=['colum_name'], inplace = True)
```

- **.reset_index** to reset the index of the dataframe

Cleaning The Data: Missing Values (Visualisation)

- Another method for identifying missing values or narrowing down an infilling method is visualisation.

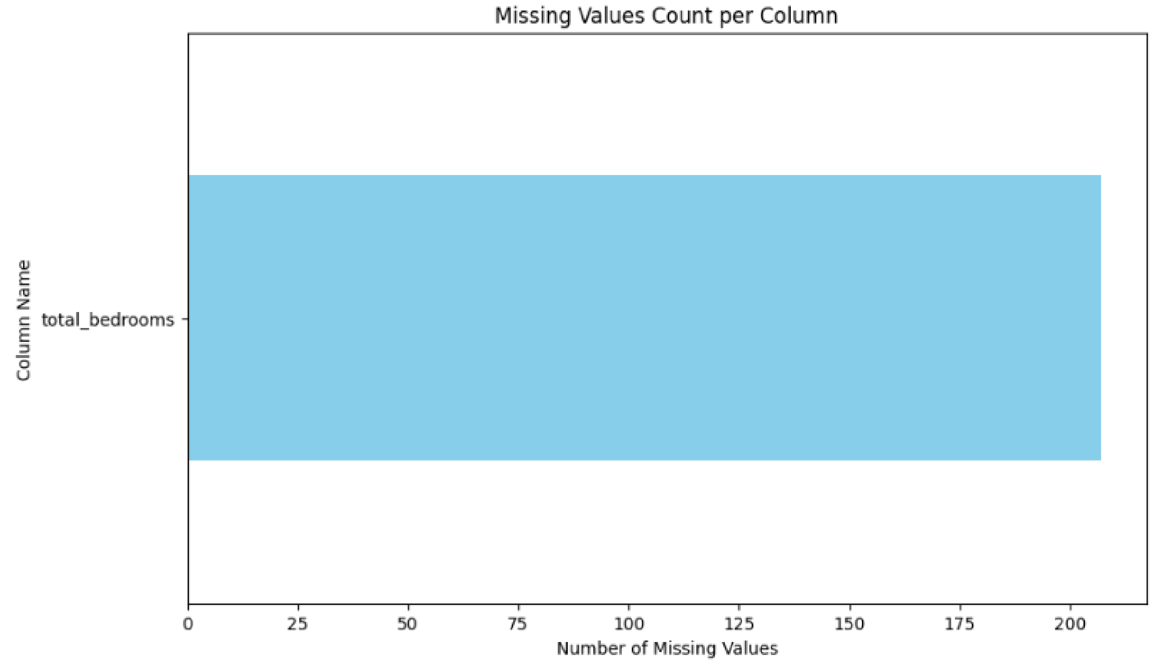
```python
import pandas as pd
import seaborn as sns
# Set up the figure size
plt.figure(figsize=(6, 4))
# Draw a heatmap with missing data
sns.heatmap(df.isnull(), cbar=False, cmap='viridis', yticklabels=False)
plt.title("Missing Values Heatmap")
plt.xlabel("Columns")
plt.ylabel("Rows")
plt.show()
```



Cleaning The Data: Missing Values (Visualisation)

```python
missing = df.isnull().sum()
missing = missing[missing > 0]  # filter only columns with missing data

# Plot
missing.sort_values().plot(kind='barh', figsize=(10, 6), color='skyblue')
plt.title("Missing Values Count per Column")
plt.xlabel("Number of Missing Values")
plt.ylabel("Column Name")
plt.show()
```



Cleaning The Data: Outliers

- Checking outliers by finding the quantile of a specific column: **df[‘column_name’].quantile()**

```
Q1 = df['median_house_value'].quantile(0.25)
Q2 = df['median_house_value'].quantile(0.50)
Q3 = df['median_house_value'].quantile(0.75)
IQR = Q3 - Q1

print('Q1 (25th percentile):', Q1)
print('Q2 (50th percentile):', Q2)
print('Q3 (75th percentile):', Q3)
print('IQR (Interquartile Range):', IQR)
```

```
Q1 (25th percentile): 119600.0
Q2 (50th percentile): 179700.0
Q3 (75th percentile): 264725.0
IQR (Interquartile Range): 145125.0
```

- Filtering out outliers

```
[18] df = df[~((df['median_house_value'] < (Q1 - 1.5 * IQR)) | (df['median_house_value'] > (Q3 - 1.5 * IQR)))]
```

Cleaning The Data: Removing Duplicates and Incorrect Data Entries

- Returns a Boolean of True or False to indicate if there is duplicate: **df.duplicated()**
- Returns the sum total count of duplicate rows: **df.duplicated().sum()**

```
[19] #Handle duplicates
     print("Duplicate rows:", df.duplicated().sum())
     df = df.drop_duplicates()

Duplicate rows: 0
```

- Check for incorrect entries (Example in column with populations):

```
#Detect incorrect data entries
f = df[df['population'] >= 0]
```

Encoding: Converting Categorical Variables to Numeric Form

- Machine learning models work with numbers, not text.
- Ocean proximity is a categorical column (e.g., 'NEAR OCEAN', 'INLAND').
- We convert each category into a numeric code using pandas:
- Result:
  - 'INLAND' might become 0
  - 'NEAR OCEAN' → 1
  - 'ISLAND' → 2, etc.

```
[21] #Encoding
     df['ocean_proximity'] = df['ocean_proximity'].astype('category')
     df['ocean_proximity_code'] = df['ocean_proximity'].cat.codes
```

Feature Engineering

- Creating **new features** helps highlight useful patterns.
- Often, **ratios** or **interactions** are more meaningful than raw values
- Rooms_per_household: how spacious the rooms are
- Bedrooms_per_room: proportion of bedrooms → house structure
- Population_per_household: average household size → can indicate urban density
- These new features often **improve model accuracy** significantly!

```
[22] #Feature Engineering
     df['rooms_per_household'] = df['total_rooms'] / df['households']
     df['bedrooms_per_room'] = df['total_bedrooms'] / df['total_rooms']
     df['population_per_household'] = df['population'] / df['households']
```

Data Processing - Randomisation

- Data grouping or clusters may influence model accuracy due to the training and testing data split for the model construction E.g., Lower house prices in the first part of the dataset.
- This leads to the model to be trained on different data than it will be tested and used on.
- The solution for this is to randomize* the rows (samples) before the data is split for ML.

```python
# Randomize rows
df_shuffled = df.sample(frac=1, random_state=42).reset_index()

# Split into train/test sets (80/20 split)
train_df, test_df = train_test_split(df_shuffled, test_size=0.2, random_state=42)

# Preview
print("Training sample:")
print(train_df.head())

print("\nTesting sample:")
print(test_df.head())
```

```
Training sample:
          Month  Passengers
124  1952-02-01         180
31   1957-03-01         356
98   1949-03-01         132
36   1958-04-01         348
16   1951-04-01         163

Testing sample:
          Month  Passengers
117  1953-10-01         211
19   1955-02-01         233
82   1954-11-01         203
97   1957-06-01         422
56   1950-11-01         114
```

Normalization

- In many cases, there is a large difference between the different variables.
- This can cause some variables to have a larger effect on the prediction due to the disparity.
- The solution for this is to normalise the dataset
- Normalize Numerical Columns
  - Some features (like total_rooms) have very large values.
  - Others (like bedrooms_per_room) are tiny.
  - This imbalance can **skew model performance**.

Normalization

- There are built in functions for normalization
- e.g., sklearn.preprocessing.normalize(..) and MinMaxScaler().
- We apply **Min-Max Scaling** to bring values into a range of 0–1:

```
[38] #Normalization
     from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
num_features = ['housing_median_age','total_rooms','total_bedrooms','population',
                'households','median_income','rooms_per_household','bedrooms_per_rc
                'population_per_household']
```

Copy the data so that there is a non- normalised copy for later use

```
[44] # Apply normalization
     df_scaled = df.copy()
     df_scaled[num_features] = scaler.fit_transform(df[num_features])
```

Normalization

- The inverse scaling can be used to transform the data (or results from the model) back to the original values.

```
[45] # Inverse the scaling
     df_original = df_scaled.copy()
     df_original[num_features] = scaler.inverse_transform(df_scaled[num_features])
```

03

Let us look at a time-series dataset and examine more data exploration and cleaning steps

Time Series Example: AirPassenger Dataset

- The AirPassengers dataset contains the monthly totals of international airline passengers from January 1949 to December 1960.
- Rows: 144 (one per month)
- Columns:
  - **Month**: Date (formatted as YYYY-MM)
  - **Passengers**: Number of airline passengers that month (in thousands)

Data Exploration

```
[4] import pandas as pd
```

```
[5] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files  airline-passengers.csv
• airline-passengers.csv(text/csv) - 2180 bytes, last modified: 6/17/2025 - 100% done
Saving airline-passengers.csv to airline-passengers.csv
```

```
df = pd.read_csv('airline-passengers.csv')
# Preview
# Display the first 5 rows using iloc
print(df.iloc[0:5])
```

```
     Month  Passengers
0  1949-01         112
1  1949-02         118
2  1949-03         132
3  1949-04         129
4  1949-05         121
```

```
[8] # Display the 7th row (index position 4)
    print(df.iloc[6])
```

```
Month         1949-07
Passengers        148
Name: 6, dtype: object
```

Data Exploration

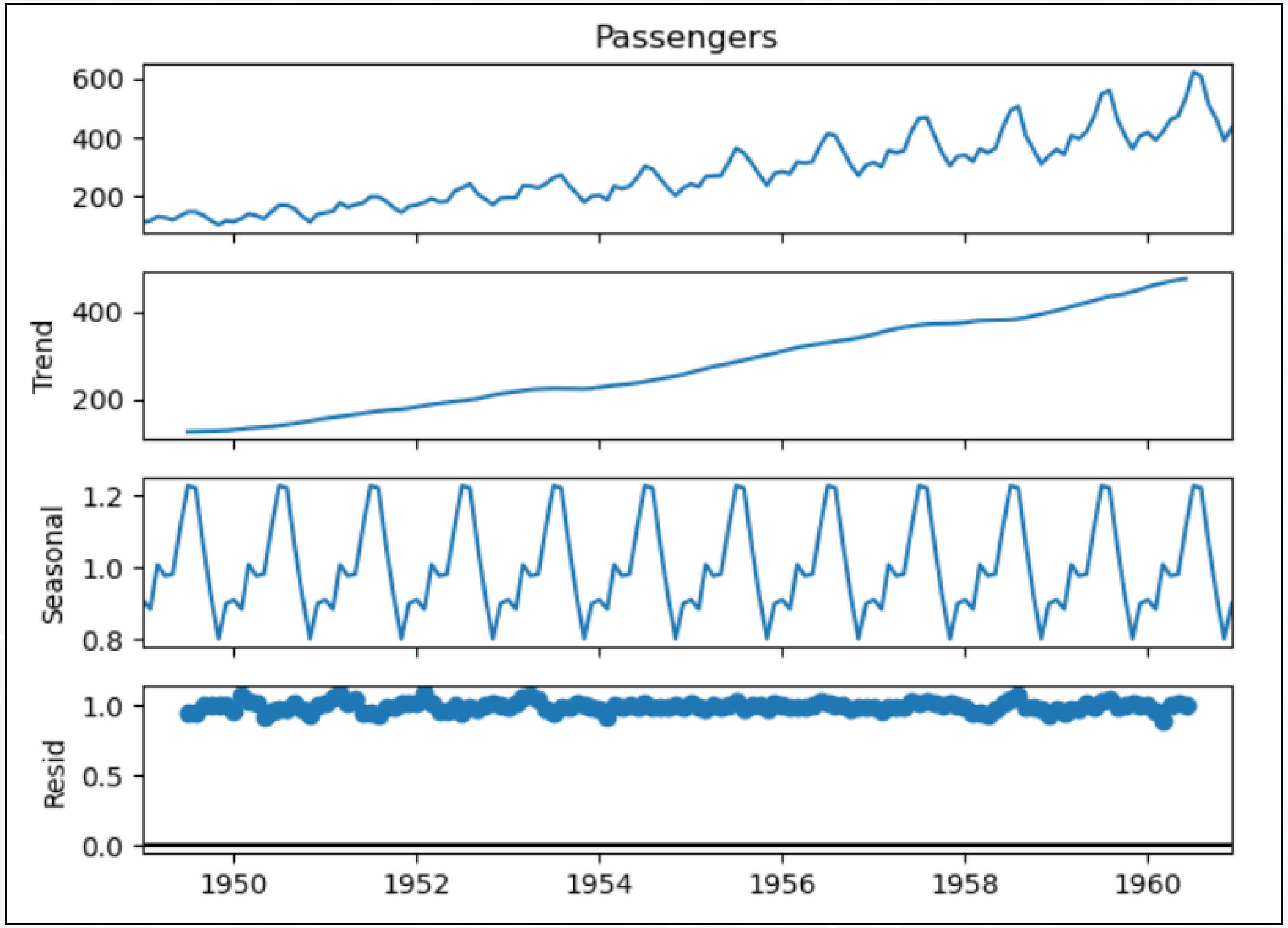
Exploring Trends, Seasonality, and Noise

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df['Passengers'], model='multiplicative'
result.plot()
plt.tight_layout()
plt.show()
```

- Components:
  - Trend: Long-term direction in the data (e.g., steady increase)
  - Seasonality: Repeating patterns at regular intervals (e.g., yearly cycles)
  - Residual/Noise: Irregular, random fluctuations not explained by trend or seasonality
- This is part of **data exploration** — it helps you understand the structure and behavior of the time series before modeling or forecasting.
- Visualising the data helps to identify missing valuesor narro down an infilling method.

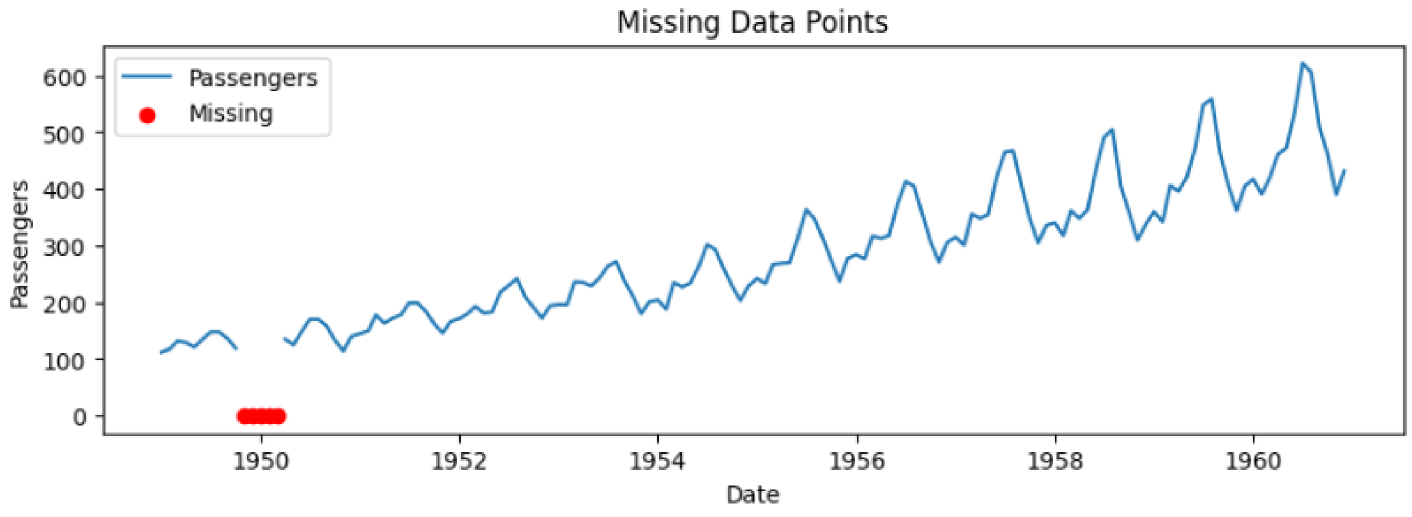
Data Exploration



- **Plot 1: Observed**
  - The original data: monthly passenger counts from 1949 to 1960
  - Shows an upward trend and seasonal fluctuations
- **Plot 2: Trend**
  - The long-term direction in the data
  - Passenger numbers are steadily increasing over time
- **Plot 3: Seasonal**
  - Regular repeating patterns (seasonality)
  - Clear yearly cycle: passenger numbers peak and drop at the same months each year
- **Plot 4: Residual**
  - The random noise left after removing trend and seasonality
  - Appears centered around 1 (because it's multiplicative), with no strong pattern, which is ideal
- **Data Visualisation to be covered in next section!!**

Data Cleaning: Missing Values

```
# Highlight missing values in red
plt.figure(figsize=(10, 3))
plt.plot(df_missing.index, df_missing['Passengers'], label='Passengers')
plt.scatter(df_missing.index[df_missing['Passengers'].isna()],
            [0]*df_missing['Passengers'].isna().sum(),
            color='red', label='Missing')
plt.title("Missing Data Points")
plt.ylabel("Passengers")
plt.xlabel("Date")
plt.legend()
plt.show()
```



Data Cleaning: Missing Values

Handling Missing Data (If Any)

```
# Simulate missing values
df_missing = df.copy()
df_missing.iloc[5:8] = None

# Check missing values
print(df_missing.isna().sum())

# Fill missing values using forward fill
df_filled = df_missing.fillna(method='ffill')
```

```
Passengers    3
dtype: int64
```

Data Processing: Working with Dates

Parsing Dates and Setting Index

```python
# Convert 'Month' to datetime and set as index
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)

# Check info
df.info()
```

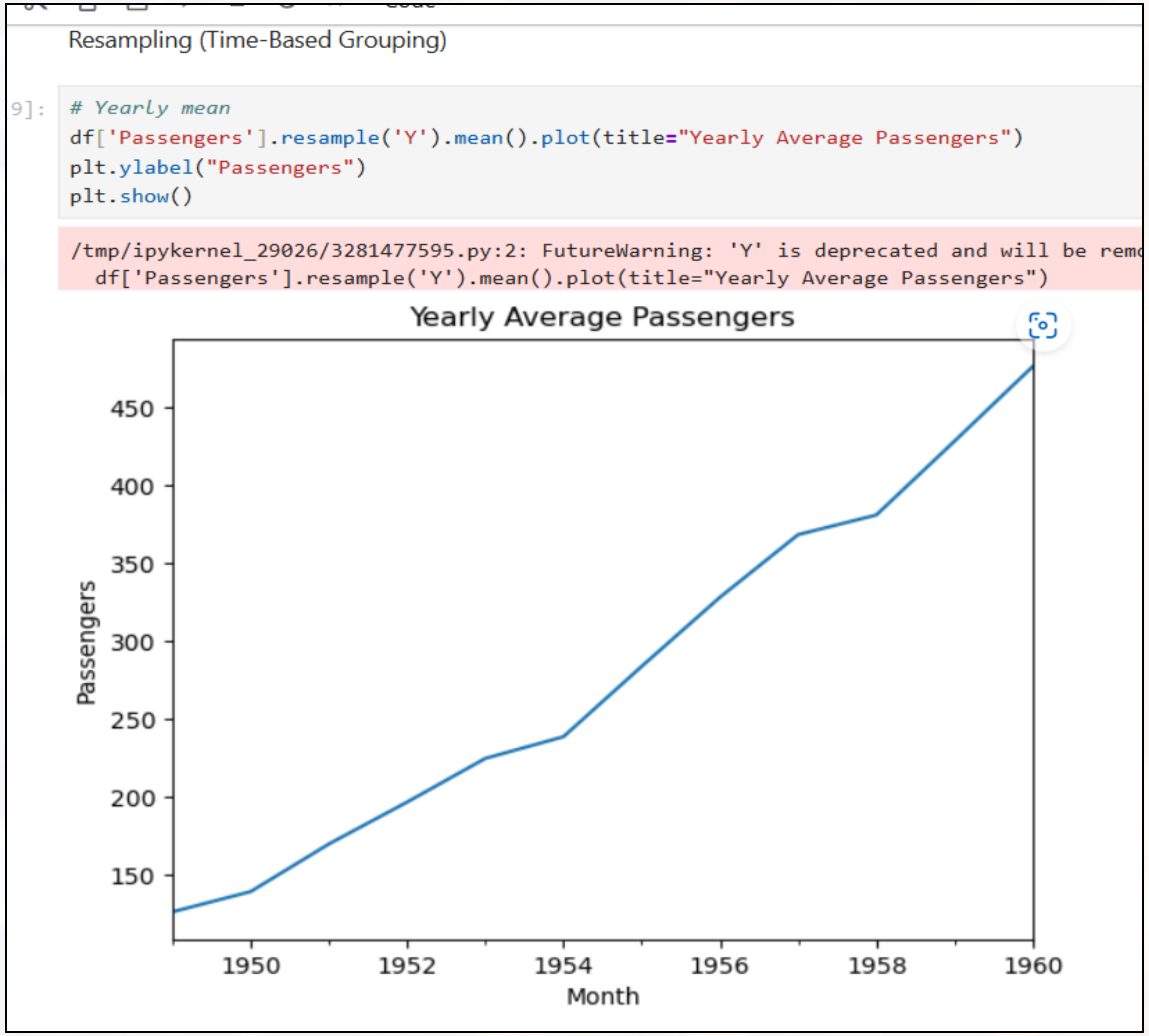
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Passengers  144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
```

Visualizing the Time Series

- **Parsing Dates**: Converts date strings (e.g., "1949-01") into proper datetime objects so Python can understand and work with them as actual dates.
- **Setting Index**: Makes the date column the index of the DataFrame, which is essential for time-based operations like resampling, rolling averages, and plotting trends over time.

Data Processing

- Resampling (Time-Based Grouping)
- Resampling involves changing the frequency of your time series data — for example, converting monthly data to yearly data by averaging or summing values over each year.



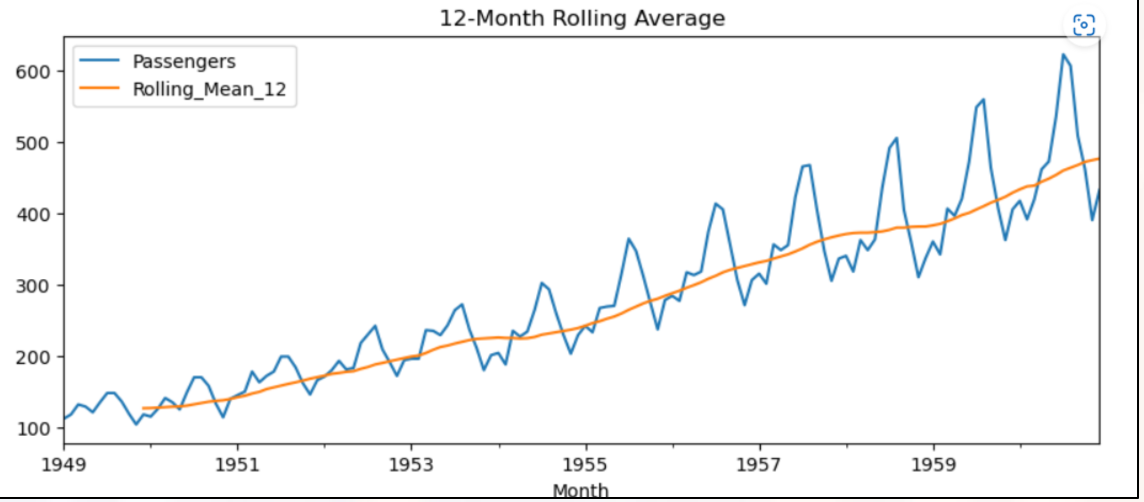
Data Processing

- Rolling Statistics (Smoothing)
  - Rolling statistics calculate values (like mean or std) over a moving window of fixed size. This is used to smooth short-term fluctuations and highlight long-term trends.
  - window=12: Looks at 12 months (1 year)
  - Common for identifying trends in time series data
- This is a **processing technique** that supports **trend detection** and helps reduce visual noise in plots.

Rolling Statistics (Smoothing)

```
df['Rolling_Mean_12'] = df['Passengers'].rolling(window=12).mean()

# Plot original + smoothed
df[['Passengers', 'Rolling_Mean_12']].plot(figsize=(10,4))
plt.title("12-Month Rolling Average")
plt.show()
```



05

Conclusion

Summary

- Data exploration helps you to **understand** your dataset
- Data Preprocessing or Data Cleaning is essential to clean and prepare your dataset
- Invest time in these steps to save effort later in modeling

Final Thoughts

- Every method that we discussed today has multiple variations. I have shown you one way to do most of the tasks in this presentation but if you were to play with it a bit or look online you will see other methods.
- The internet is a great tool – code for almost anything that you want to do now or in the future have already been posted by someone else online. Just don’t give up immediately and experiment with different search terms.
- Tools such as ChatGPT can be valuable when coding (as long as you are only using it when allowed), but don’t take everything provided as gospel. Sometimes additional work is needed to get the code to do exactly what is needed, and sometimes it can be way of the mark

References

- [1] https://www.educba.com/data-exploration/

Thank You

Email: bsebusho@csir.co.za

NATIONAL INTEGRATED
CYBERINFRASTRUCTURE SYSTEM
DIRISA