Data Scientist Cheatsheet

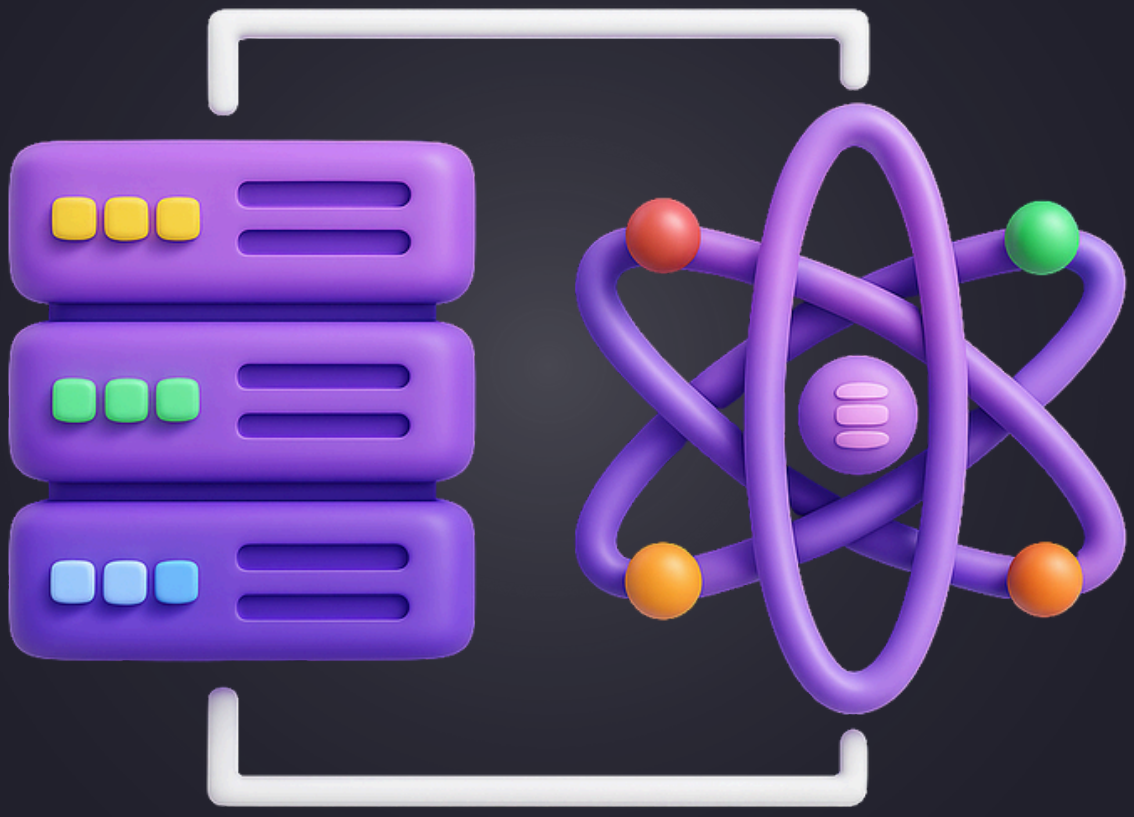


TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION TO DATA SCIENCE

1.1 What is Data Science?

Basic Definition:

- Data Science is the study and practice of extracting knowledge and insights from data. It uses math, coding, and domain understanding to solve real-world problems.
- You can think of it like this:
- "You collect data → You understand the data → You find patterns → You use those patterns to make smart decisions."

In Simple Terms:

Data Science is a combination of:

- Statistics – to understand numbers and trends
- Programming – to write code and work with data
- Visualization – to create graphs and charts
- Domain Knowledge – to understand the problem you're solving

Real-Life Example:

- Let's say you own a small shop.
- You write down what you sell every day in a notebook.

Over time, this notebook becomes:

- A source of data (your sales)
- With that data, you can ask questions like:
  - What is my best-selling product?
  - On which days do I sell the most?
  - When should I give discounts?
- Answering these questions using your data = Data Science.

Why is it Important?

- It helps companies make better decisions
- It helps predict the future
- It helps to find problems early
- It improves customer experience

1.2 Data Science Workflow

- The Data Science workflow is a set of steps that every data scientist follows to solve a problem using data.
- Let's understand each step like you're solving a school project.

Step 1: Problem Understanding

- Ask a clear question.

You must understand:

- What is the goal?
- What are we trying to find out?

Example: "Why are customers uninstalling our app?"

1. INTRODUCTION TO DATA SCIENCE

Step 2: Data Collection

- This means gathering the data from different places.

Sources of data:

- Databases (like MySQL, MongoDB)
- Excel/CSV files
- Web scraping (getting data from websites)
- APIs (data from services like weather, maps)

Example: Get data about user logins, actions, purchases, feedback.

Step 3: Data Cleaning (Preprocessing)

- Raw data is often messy.
- This step is about making it correct and usable.

Common cleaning tasks:

- Removing duplicates
- Filling missing values
- Fixing typos or wrong formats
- Removing outliers (very large or very small values)

Example: If someone's age is written as 400 – that needs to be fixed.

Step 4: Data Exploration (EDA – Exploratory Data Analysis)

- You now look at the data to understand it better.

This includes:

- Basic statistics (mean, median, mode)
- Graphs (bar chart, pie chart, line chart)
- Finding patterns and trends

Example: More users uninstall the app after 10 PM — this might mean poor late-night support.

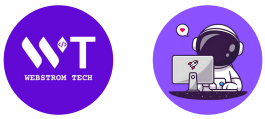
Step 5: Data Modeling

- Here, you use Machine Learning or statistical models to make predictions or classifications.

Types of models:

- Linear Regression (predict numbers)
- Decision Trees (classify options)
- Clustering (grouping similar items)

Example: Predict if a new user will uninstall the app in 7 days.



1. INTRODUCTION TO DATA SCIENCE

Step 6: Evaluation

- Check how well your model is performing.

Use:

- Accuracy
- Precision and Recall
- Confusion Matrix

Example: Your model is 80% accurate — this means it correctly predicted 8 out of 10 cases.

Step 7: Communication (Visualization & Reporting)

- Now it's time to present your findings.

You can use:

- Charts (bar, line, scatter, heatmaps)
- Dashboards (in tools like Power BI or Tableau)
- Reports (slides or PDF summaries)

Example: Show a graph that uninstall rates are high on weekends, and give suggestions.

1.3 Roles & Responsibilities of a Data Scientist

- A Data Scientist is someone who works with data to help people or businesses make better decisions.

They are problem-solvers who use data to:

- Understand what is happening
- Find out why it's happening
- Predict what might happen next

Main Roles & Tasks:

1. Ask the Right Questions
    - Understand the business goal
    - Define what problem needs to be solved
2. Collect Data
    - From different sources like SQL, Excel, APIs
    - Ensure the data is reliable
3. Clean the Data
    - Fix missing or incorrect values
    - Make the data ready for use
4. Explore the Data
    - Find patterns, trends, and relationships
    - Use visualizations to understand better
5. Build Models
    - Use ML or statistical models
    - Train and test the model on real data.

1. INTRODUCTION TO DATA SCIENCE

6. Evaluate Models
    - Check how well the model is working
    - Improve or try different models if needed
7. Present Insights
    - Use graphs and dashboards
    - Explain in simple language what the data shows

Skills a Data Scientist Needs:

- Programming (Python or R)
- Statistics & Math
- Data Handling (Pandas, SQL)
- Machine Learning
- Visualization (Matplotlib, Seaborn, Power BI, Tableau)
- Communication Skills

Who Do They Work With?

Role	How They Help
Data Analyst	Creates reports and dashboards
Data Engineer	Builds pipelines to collect and store data
Business Analyst	Translates business needs into data questions
Machine Learning Engineer	Helps put models into production

2. PYTHON FOR DATA SCIENCE

2.1 Python Basics: Syntax, Variables, Data Types

- What is Python?
- Python is a programming language that is easy to read and write.

It's widely used in data science because:

- It's beginner-friendly
- It has powerful libraries for data
- It looks like plain English

Syntax (How Python Code Looks)

- Syntax means rules of how code should be written.

Example:

```python
print("Hello, World")
```

- No semicolons at the end of lines
- Indentation (spaces) is used to define blocks of code
- Case-sensitive — Name and name are different

Variables

- A variable is like a box that stores information.

Example:

```python
name = "Rudra"
age = 19
```

Here:

- name is a variable storing the text “Rudra”
- age stores the number 19

You can change a variable anytime:

```python
age = 20
```

Data Types

- Python has different types of data:

2. PYTHON FOR DATA SCIENCE

Data Types

- Python has different types of data:

Type	Example	Description
int	10	Whole numbers
float	10.5	Decimal numbers
str	"hello"	Text
bool	True, False	Yes/No type values
list	[1, 2, 3]	Collection of items
dict	{"a": 1}	Key-value pairs (dictionary)

2.2 Loops, Conditionals, and Functions

Conditionals

- Used to make decisions in code.
- They check if something is True or False.

Example:

```python
age = 18

if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

Loops

- Loops help you repeat code.

For loop:

```python
for i in range(5):
    print(i)
```

Output:

```
0
1
2
3
4
```

2. PYTHON FOR DATA SCIENCE

While loop:

```python
count = 0
while count < 3:
    print("Hi")
    count += 1
```

Functions

- A function is a block of code you can reuse.

Example:

```python
def greet(name):
    print("Hello", name)

greet("Rudra")
```

- Functions make code shorter and cleaner.

2.3 List Comprehensions

- List comprehension is a shorter way to create lists.

Without list comprehension:

```python
squares = []
for i in range(5):
    squares.append(i * i)
```

With list comprehension:

```python
squares = [i * i for i in range(5)]
```

- This line does the same work in one line.
- It's faster and looks cleaner.

More examples:

```python
evens = [x for x in range(10) if x % 2 == 0]
```

2.4 Useful Libraries: NumPy, Pandas, Matplotlib, Seaborn

- Libraries are ready-made tools in Python that help you do tasks faster.
- NumPy (Numerical Python)
- Used for math, arrays, and numbers.

Example:

2. PYTHON FOR DATA SCIENCE

2.4 Useful Libraries: NumPy, Pandas, Matplotlib, Seaborn

Example:

```python
import numpy as np

a = np.array([1, 2, 3])
print(a.mean())  # Average of array
```

Key Features:

- Arrays (like a list but faster)
- Math operations
- Linear algebra

Pandas

- Used for data tables (rows and columns), like Excel.

Example:

```python
import pandas as pd

data = pd.DataFrame({
    "Name": ["Rudra", "Parth"],
    "Score": [90, 85]
})

print(data.head())
```

Key Features:

- Read data from CSV, Excel
- Filter, sort, clean data
- Handle missing data

Matplotlib

- Used to make basic graphs like line charts and bar charts.

Example:

```python
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Simple Graph")
plt.show()
```

Seaborn

- Used for beautiful and advanced graphs (built on top of Matplotlib).

Example:

```python
import seaborn as sns

sns.barplot(x=["A", "B", "C"], y=[3, 7, 5])
```

2. PYTHON FOR DATA SCIENCE

Seaborn

Key Features:

- Easy to use
- Looks better than plain Matplotlib
- Useful for data exploration

3. DATA MANIPULATION WITH PANDAS

- Pandas is a powerful Python library used to work with structured data like tables (rows and columns).
- It helps you to analyze, clean, filter, and modify data easily.

3.1 Series and DataFrame

Series

- A Pandas Series is like a single column of data.

Example:

```python
import pandas as pd

numbers = pd.Series([10, 20, 30])
print(numbers)
```

Output:

```go
0    10
1    20
2    30
dtype: int64
```

- It has index numbers on the left (0, 1, 2)
- And values on the right (10, 20, 30)

DataFrame

- A DataFrame is like a whole Excel sheet — a table with rows and columns.

Example:

```Python
data = pd.DataFrame({
    "Name": ["Rudra", "Ravi"],
    "Score": [90, 85]
})
print(data)
```

Output:

```markdown
    Name  Score
0  Rudra     90
1  Ravi      85
```

3.2 Indexing, Slicing, Filtering

Indexing

Use .loc[] and .iloc[] to access rows.

- .loc[] uses label/index name
- .iloc[] uses row number

3. DATA MANIPULATION WITH PANDAS

3.2 Indexing, Slicing, Filtering

Example:

```python
print(data.loc[0])     # Row with label 0
print(data.iloc[1])    # Row at position 1
```

Slicing

- You can select a portion of the data using slicing.

Example:

```python
print(data[0:1])   # First row only
```

Filtering

- Use conditions to filter rows.

Example:

```python
print(data[data["Score"] > 85])
```

Output:

```
    Name  Score
0  Rudra     90
```

3.3 GroupBy and Aggregations

GroupBy

- groupby() is used to group rows by a column and then apply a function.

Example:

```python
df = pd.DataFrame({
    "Class": ["A", "A", "B", "B"],
    "Marks": [80, 90, 70, 60]
})

grouped = df.groupby("Class")["Marks"].mean()
print(grouped)
```

Output:

```
Class
A    85.0
B    65.0
Name: Marks, dtype: float64
```

3. DATA MANIPULATION WITH PANDAS

Here:

- Students are grouped by class
- Then their average marks are calculated

Aggregations

Functions like:

- .sum()
- .mean()
- .count()
- .max(), .min()

Example:

```python
print(df["Marks"].sum())   # Total marks
```

3.4 Merging, Joining, and Concatenation

- You often need to combine different tables.

Merging

- Like SQL joins.
- You combine two DataFrames based on a common column.

Example:

```python
students = pd.DataFrame({
    "ID": [1, 2],
    "Name": ["Rudra", "Ravi"]
})

scores = pd.DataFrame({
    "ID": [1, 2],
    "Score": [90, 85]
})

merged = pd.merge(students, scores, on="ID")
print(merged)
```

Output:

```nginx
   ID   Name  Score
0   1  Rudra     90
1   2  Ravi      85
```

Joining

- Uses index instead of column.
- Usually works with .join() function.

3. DATA MANIPULATION WITH PANDAS

Concatenation

- Used to stack multiple DataFrames together.

Example (Row-wise):

```python
df1 = pd.DataFrame({"A": [1, 2]})
df2 = pd.DataFrame({"A": [3, 4]})

result = pd.concat([df1, df2])
print(result)
```

Output:

```css
   A
0  1
1  2
0  3
1  4
```

3.5 Handling Missing Data

- Real-world data often has missing or null values.

Detect Missing Data

```python
df.isnull()       # Shows True for missing
df.isnull().sum() # Count of missing values per column
```

Drop Missing Values

```python
df.dropna()       # Removes rows with any missing values
```

Fill Missing Values

```python
df.fillna(0)      # Fills missing with 0
df.fillna("Unknown")  # Fill with string
```

You can also use:

```python
df["column"].fillna(df["column"].mean())  # Fill with average
```

4. DATA VISUALIZATION

- Data Visualization is the process of turning numbers into pictures — so it's easier to see patterns, trends, and insights.

In Python, the most popular libraries used are:

- Matplotlib – for basic charts
- Seaborn – for more advanced and beautiful charts

Let's go step by step:

4.1 Line, Bar, Pie Charts (Matplotlib)

Line Chart

- Used to show changes over time (like stock price, temperature, sales, etc.).

Example:

```python
import matplotlib.pyplot as plt

days = [1, 2, 3, 4, 5]
sales = [100, 200, 150, 300, 250]

plt.plot(days, sales)
plt.title("Daily Sales")
plt.xlabel("Day")
plt.ylabel("Sales")
plt.show()
```

Bar Chart

- Used to compare categories or groups (like marks of students or items sold).

Example:

```python
names = ["Ravi", "Sneha", "Anjali"]
scores = [85, 90, 95]

plt.bar(names, scores)
plt.title("Student Scores")
plt.xlabel("Name")
plt.ylabel("Score")
plt.show()
```

Pie Chart

- Used to show parts of a whole (like percentage of sales by product).

Example:

```python
products = ["Mobile", "Laptop", "Tablet"]
sales = [300, 500, 200]

plt.pie(sales, labels=products, autopct="%1.1f%%")
plt.title("Sales by Product")
plt.show()
```

4. DATA VISUALIZATION

4.2 Histograms, Boxplots, Heatmaps (Seaborn)

- To use Seaborn, you must first import it:

```python
import seaborn as sns
```

Histogram

- Used to show distribution of data — how often values occur.

Example:

```python
import numpy as np

data = np.random.randn(100)

sns.histplot(data, bins=10, kde=True)
plt.title("Value Distribution")
plt.show()
```

Boxplot

- Used to show summary of data — minimum, maximum, median, and outliers.

Example:

```python
marks = [70, 75, 80, 100, 50, 65, 85, 90]

sns.boxplot(data=marks)
plt.title("Marks Boxplot")
plt.show()
```

Heatmap

- Used to show data as a colored grid — great for comparing multiple values.

Example:

```python
import pandas as pd

data = pd.DataFrame({
    "Math": [90, 80, 70],
    "Science": [85, 88, 75],
    "English": [78, 82, 89]
}, index=["Ravi", "Sneha", "Anjali"])

sns.heatmap(data, annot=True)
plt.title("Student Scores Heatmap")
plt.show()
```

4. DATA VISUALIZATION

4.3 Customizing Graphs

- To make your charts look better or fit your brand/style, you can customize:

Title & Axis Labels

```python
plt.title("Chart Title")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
```

Colors

```python
plt.bar(names, scores, color='green')
```

Gridlines

```python
plt.grid(True)
```

Legends

If you have more than one line or bar:

```python
plt.legend(["Line 1", "Line 2"])
```

Figure Size

```python
plt.figure(figsize=(10, 6))
```

4.4 Real-world Visualization Examples

- Example 1: Sales Trend Over a Week (Line Chart)

```python
days = ["Mon", "Tue", "Wed", "Thu", "Fri"]
sales = [150, 200, 170, 220, 180]

plt.plot(days, sales, marker='o')
plt.title("Weekly Sales")
plt.xlabel("Day")
plt.ylabel("Sales")
plt.grid(True)
plt.show()
```

4. DATA VISUALIZATION

4.4 Real-world Visualization Examples

- Example 2: Product Sales Comparison (Bar Chart)

```python
products = ["Phone", "Laptop", "TV", "Headphones"]
units_sold = [400, 300, 250, 500]

plt.bar(products, units_sold, color='orange')
plt.title("Product Sales")
plt.show()
```

- Example 3: Student Scores Heatmap (Seaborn)

```python
data = pd.DataFrame({
    "Ravi": [85, 75, 90],
    "Sneha": [88, 95, 92],
    "Anjali": [78, 82, 89]
}, index=["Math", "Science", "English"])

sns.heatmap(data, annot=True, cmap="YlGnBu")
plt.title("Subject Scores per Student")
plt.show()
```

5. STATISTICS & PROBABILITY

- Statistics and Probability are the foundation of Data Science.
- They help you understand data, make predictions, and test ideas with confidence.

5.1 Descriptive Statistics

- Descriptive statistics help you summarize and describe the main features of a dataset.
- Common Terms:

Term	Meaning (Simple)
Mean	Average of numbers
Median	Middle value
Mode	Value that appears most often
Range	Biggest - Smallest value
Variance	How spread out the data is
Standard Deviation	Average distance from the mean

Example:

- Let's say student scores are:

```python
scores = [85, 90, 75, 95, 80]
```

- Mean = (85 + 90 + 75 + 95 + 80) / 5 = 85
- Median = 85 (middle value)
- Mode = No repeating value
- Range = 95 - 75 = 20

5.2 Probability Distributions

- Probability shows how likely something is to happen.
- A probability distribution shows all possible outcomes and how likely each one is.

Types of Distributions:

1. Uniform Distribution

- All outcomes are equally likely.

Example: Rolling a dice (each number has 1/6 chance)

2. Normal Distribution

- Bell-shaped curve. Most values are around the mean.

5. STATISTICS & PROBABILITY

- Example: Heights of people, test scores.

3. Binomial Distribution

- Used when there's only 2 outcomes like yes/no, success/failure.
- Example: Tossing a coin 10 times.

Example:

```python
import numpy as np
import matplotlib.pyplot as plt

data = np.random.normal(0, 1, 1000)
plt.hist(data, bins=30)
plt.title("Normal Distribution")
plt.show()
```

5.3 Bayes' Theorem

- Bayes' Theorem helps you update your belief when you get new information.

Formula:

- $P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$

Where:

- P(A|B) = Probability of A given B (updated belief)
- P(B|A) = Probability of B given A (how likely B is if A is true)
- P(A) = Probability of A happening
- P(B) = Probability of B happening

Example:
Suppose:

- 1% of people have a disease (P(Disease) = 0.01)
- Test is 99% accurate
- What is the chance a person really has the disease if they tested positive?

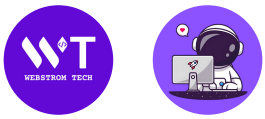
Bayes' Theorem helps you solve this.

5.4 Hypothesis Testing

- It helps you test if a claim about data is true or not using evidence.

Steps in Hypothesis Testing:

1. State the Hypotheses:
    - Null Hypothesis (H_0): Nothing is happening
    - Alternative Hypothesis (H_1): Something is happening



5. STATISTICS & PROBABILITY

2. Choose Significance Level (α):
    - Usually 0.05 (means 5% risk of being wrong)
3. Perform the Test:
    - Use statistical test (like t-test, z-test)
4. Compare p-value with α:
    - If p-value < α → Reject H_0 (significant result)

Example:

- Claim: A new teaching method improves test scores.
- You collect scores from students with old and new methods and compare.
- If p-value is small, you reject the null hypothesis and accept the new method works.

5.5 P-Values and Confidence Intervals

What is a p-value?

- The p-value tells you how likely your results happened by random chance.
- Small p-value (< 0.05) = Result is significant (not by chance)
- Large p-value (> 0.05) = Result is likely due to chance

What is a Confidence Interval?

- It tells you a range of values where you believe the true result lies — with confidence.

Example:

- "Average height is 160 cm ± 5 cm with 95% confidence."
- Means you are 95% sure that the real average height is between 155 cm and 165 cm.

6. EXPLORATORY DATA ANALYSIS (EDA)

- EDA is the process of exploring and understanding your dataset before building any models.
- Think of EDA like looking at your data under a magnifying glass — to spot patterns, missing values, outliers, and relationships between columns.

6.1 Univariate, Bivariate, and Multivariate Analysis

Univariate Analysis

- "Uni" means one – so we analyze one column at a time

Goals:

- Understand the distribution of values
- Find mean, median, mode, min, max
- Detect outliers or skewness

Examples:

```python
df["Age"].describe()
sns.histplot(df["Age"])
sns.boxplot(x=df["Salary"])
```

You might find:

- Most customers are aged 20–30
- Salaries are mostly between ₹20,000–₹50,000

Bivariate Analysis

- "Bi" means two – analyze two columns together

Goals:

- See relationships between variables
- Compare values using graphs

Examples:

```python
sns.scatterplot(x="Age", y="Salary", data=df)
sns.boxplot(x="Gender", y="Spending", data=df)
```

You might learn:

- Older people spend more
- Males and females have different spending patterns

Multivariate Analysis

- Analyze three or more columns together

Goals:

- Understand how multiple factors interact
- Build a deeper picture

Examples:

```python
sns.pairplot(df[["Age", "Salary", "Spending"]])
sns.heatmap(df.corr(), annot=True)
```

6. EXPLORATORY DATA ANALYSIS (EDA)

You might find:

- Salary and spending are highly related
- Age has little effect on salary

6.2 Outlier Detection

- Outliers are data points that are very different from others.
- They can confuse your model if not handled properly.

Methods to detect outliers:

1. Boxplot

```python
sns.boxplot(x=df["Income"])
```

- Any dots outside the box are outliers.

2. Z-score

```python
from scipy import stats
z = np.abs(stats.zscore(df["Income"]))
df[z > 3]  # Outliers
```

3. IQR Method

```python
Q1 = df["Income"].quantile(0.25)
Q3 = df["Income"].quantile(0.75)
IQR = Q3 - Q1

outliers = df[(df["Income"] < (Q1 - 1.5 * IQR)) | (df["Income"] > (Q3 + 1.5 * IQR))]
```

What to do with outliers?

- Keep them if they are important
- Remove them if they are errors
- Transform data (like log scale) to reduce their effect

6.3 Correlation Matrix

Correlation shows how strongly two columns are related.

- Range: -1 to +1
  - +1: Perfect positive correlation
  - -1: Perfect negative correlation
  - 0: No correlation
- How to use it:

```python
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
```

You can see:

- Which features move together (example: Salary and Experience)
- Which features are negatively related (example: Age and Screen Time)

6. EXPLORATORY DATA ANALYSIS (EDA)

6.4 Feature Engineering Techniques

- Feature Engineering means creating new columns or modifying existing ones to help the model learn better.

Common Techniques:

1. Creating New Features

```python
df["Age_Group"] = pd.cut(df["Age"], bins=[0, 18, 30, 45, 60], labels=["Teen", "Young", "Adult", "Senior"])
```

2. Encoding Categorical Variables

```python
df = pd.get_dummies(df, columns=["Gender", "City"])
```

3. Scaling Values

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df["Salary_scaled"] = scaler.fit_transform(df[["Salary"]])
```

4. Date Features

```python
df["Year"] = df["Purchase_Date"].dt.year
df["Month"] = df["Purchase_Date"].dt.month
```

5. Interaction Features

```python
df["Income_per_Age"] = df["Income"] / df["Age"]
```

7. MACHINE LEARNING BASICS

- Machine Learning (ML) means teaching a computer to learn from data and make decisions or predictions — without being told what to do step by step.

7.1 Supervised vs Unsupervised Learning

Supervised Learning

- The model learns from labeled data.
- (We give the correct answers during training.)

Example:

- If you give student scores along with their pass/fail status, the model learns to predict pass/fail for new students.

Data example:

Hours Studied	Score	Result
2	50	Fail
5	90	Pass

Common Algorithms:

- Linear Regression
- Logistic Regression
- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)

Unsupervised Learning

- The model learns from data without labels.
- (No correct answers are given — the model finds hidden patterns.)

Example:

- Given only customer purchase data, the model groups similar customers together (like “high-spenders” or “frequent buyers”).

Data example:

Customer	Purchases	Visits
A	1000	10
B	300	2

Common Algorithms:

- K-Means Clustering
- Hierarchical Clustering
- PCA (Principal Component Analysis)

7. MACHINE LEARNING BASICS

7.2 Train/Test Split, Cross-Validation

- Train/Test Split

Before training a model, we divide the data:

- Training Set → Used to teach the model
- Test Set → Used to check how well it learned

```python
from sklearn.model_selection import train_test_split

X = df[["Hours_Studied"]]
y = df["Result"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Usually:

- 80% data → training
- 20% data → testing

Cross-Validation

- Instead of testing once, we test multiple times on different parts of data to get a better accuracy estimate.

```python
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print(scores.mean())
```

- Cross-validation avoids bad luck from one bad test split.

7.3 Model Evaluation Metrics (Accuracy, Precision, Recall, F1-score)

- Once the model gives predictions, we need to check how good it is.
- Let's assume we built a model to predict if an email is spam or not.

Accuracy

- Tells us how many total predictions were correct.

```python
Accuracy = (Correct Predictions) / (Total Predictions)
```

- Example: If 90 out of 100 are right → accuracy = 90%
- But: Accuracy alone can mislead when data is imbalanced.

Precision

- Out of all emails the model predicted as spam, how many were actually spam?

7. MACHINE LEARNING BASICS

Precision

- Out of all emails the model predicted as spam, how many were actually spam?

```python
Precision = TP / (TP + FP)
```

- TP: True Positive (correct spam prediction)
- FP: False Positive (wrongly predicted spam)

Recall

- Out of all actual spam emails, how many did we correctly find?

```python
Recall = TP / (TP + FN)
```

- FN: False Negative (missed spam)

F1-Score

- F1-Score is a balance between Precision and Recall.

```python
F1 = 2 * (Precision * Recall) / (Precision + Recall)
```

- When data is imbalanced, F1 is the best metric to use.

7.4 Overfitting vs Underfitting

Overfitting

- Model learns too much from training data, including the noise.
- Great on training set
- Bad on test set

Think of a student memorizing answers, but failing in real exams.

Underfitting

- Model doesn't learn enough — too simple to understand the pattern.
- Bad on both training and test sets
- Think of a student who didn't study at all — just guesses randomly.

How to Fix?

Problem	Solution
Overfitting	Use simpler models, regularization, more data
Underfitting	Use more complex models, add features

8. COMMON MACHINE LEARNING ALGORITHMS

- These are the most widely used algorithms every beginner in data science should know. Each has its own use case and is used based on the type of data and the problem to solve (prediction, classification, grouping, etc.).

8.1 Linear & Logistic Regression

Linear Regression

- Used when we want to predict a number (like marks, salary, price).
- It draws a straight line through the data points.

Example:

- Predicting marks based on hours studied.

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

Formula:

- y=mx+c

Where:

- y is the output (predicted value)
- x is the input (feature)
- m is the slope (learned weight)
- c is the intercept

Logistic Regression

- Used when we want to predict a category (yes/no, spam/not spam, pass/fail).
- Even though it has “regression” in the name, it’s used for classification.

Example:

- Predicting if a student will pass or fail based on study hours.

```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

8. COMMON MACHINE LEARNING ALGORITHMS

8.2 Decision Trees & Random Forest

Decision Tree

- It splits the data into branches like a tree, based on questions.

Example:

- If Age > 18 → go right
- Else → go left

```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

- ✅ Easy to understand
- ❌ Can overfit on small data

Random Forest

- Random Forest = many decision trees combined.
- It uses voting from multiple trees to give a better result.

```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

- More accurate than a single tree
- Reduces overfitting

8.3 K-Nearest Neighbors (KNN)

- KNN looks at the 'K' closest points to a new data point and votes.
- If most nearby points are "Pass", then new data is also "Pass".

Example:

- Predicting if a new student will pass, based on how nearby students performed.

```python
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
```

- ✅ Simple to understand
- ❌ Slow with large datasets

8. COMMON MACHINE LEARNING ALGORITHMS

8.4 Support Vector Machines (SVM)

- SVM draws a line (or hyperplane) that best separates the data into classes.
- It tries to keep the widest possible margin between the two groups.

Example:

- Classifying if a message is spam or not spam.

```python
from sklearn.svm import SVC
model = SVC()
model.fit(X_train, y_train)
```

✅ Works well in complex spaces
❌ Can be hard to tune

8.5 K-Means Clustering

- K-Means is an unsupervised learning algorithm.
- It groups data into K clusters based on similarity.

Example:

- Grouping customers based on purchase behavior.

```python
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(data)
```

✅ Easy to use
❌ You must choose the value of K
❌ Sensitive to outliers

8.6 Principal Component Analysis (PCA)

- PCA is a dimensionality reduction technique.
- It reduces many columns (features) into fewer important components, keeping the most useful information.

Why use PCA?

- To make models faster
- To remove noise
- To visualize high-dimensional data in 2D or 3D

```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
new_data = pca.fit_transform(original_data)
```

✅ Makes big data easier to handle
✅ Helps in visualization
❌ May lose some information

9. SCIKIT-LEARN ESSENTIALS

- Scikit-learn (or sklearn) is one of the most popular Python libraries for Machine Learning.
- It gives you tools for data preprocessing, training models, evaluating them, and improving them — all in one place.

9.1 Preprocessing Pipelines
What is Preprocessing?

- Before training a model, we must prepare the data.

This includes:

- Handling missing values
- Scaling numbers
- Encoding text (like “Male”, “Female” → 0, 1)

What is a Pipeline?

- A Pipeline is a step-by-step process where you:

1. Clean the data
2. Scale or encode it
3. Train the model

- Instead of writing multiple steps, you bundle them into one line.

Example:

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("model", LogisticRegression())
])

pipe.fit(X_train, y_train)
```

Now pipe handles both scaling + model training in one go.

9.2 Model Training & Evaluation

- Train a Model

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X_train, y_train)
```

- Make Predictions

```python
predictions = model.predict(X_test)
```

9. SCIKIT-LEARN ESSENTIALS

- Evaluate the Model

```python
from sklearn.metrics import accuracy_score, confusion_matrix

print("Accuracy:", accuracy_score(y_test, predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
```

You can also use:

- precision_score()
- recall_score()
- f1_score()

9.3 Hyperparameter Tuning

What are Hyperparameters?

- Hyperparameters are settings you choose before training a model.

Example:

- Number of trees in a Random Forest
- Value of K in KNN
- Learning rate in Gradient Boosting

- Choosing the right hyperparameters can make a huge difference in model performance.

Example:

```python
model = RandomForestClassifier(n_estimators=100, max_depth=5)
```

- Here, n_estimators and max_depth are hyperparameters.

9.4 Grid Search & Randomized Search

Grid Search

- Tests all possible combinations of hyperparameters.

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5]
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=3)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
```

✅ Finds the best
❌ Can be slow if combinations are many

9. SCIKIT-LEARN ESSENTIALS

Randomized Search

- Tests random combinations of parameters, not all.

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': randint(3, 10)
}

rand = RandomizedSearchCV(RandomForestClassifier(),
param_distributions=param_dist, n_iter=10, cv=3)
rand.fit(X_train, y_train)

print("Best Parameters:", rand.best_params_)
```

✅ Faster
✅ Good for large parameter spaces
❌ May miss the best if unlucky

10. SQL FOR DATA SCIENCE

- SQL (Structured Query Language) is used to talk to databases.
- As a Data Scientist, you use SQL to get data, filter it, summarize it, and prepare it for analysis.
- Let's break it down step-by-step.

10.1 Basic SELECT Statements

- The SELECT statement is used to get data from a table.

Syntax:

```sql
SELECT column1, column2
FROM table_name;
```

Example:

```sql
SELECT name, age
FROM students;
```

To get all columns, use *:

```sql
SELECT * FROM students;
```

10.2 Filtering & Sorting

- Filtering Rows (WHERE clause)
- You use WHERE to choose only specific rows.

```sql
SELECT name, age
FROM students
WHERE age > 18;
```

You can also use:

- =, !=, <, >, <=, >=
- AND, OR, NOT
- IN, BETWEEN, LIKE

Example:

```sql
SELECT * FROM employees
WHERE department = 'Sales' AND salary > 50000;
```

10. SQL FOR DATA SCIENCE

- Sorting Rows (ORDER BY)

```sql
SELECT name, salary
FROM employees
ORDER BY salary DESC;
```

- ASC = ascending (default)
- DESC = descending

10.3 Aggregation Functions

- These functions summarize your data.

Function	What it does
COUNT()	Counts rows
SUM()	Adds up values
AVG()	Average
MAX()	Largest value
MIN()	Smallest value

Example:

```sql
SELECT COUNT(*) FROM students;
SELECT AVG(salary) FROM employees;
```

10.4 JOINS, GROUP BY, HAVING

JOINS

- Used to combine data from two or more tables.

```sql
SELECT employees.name, departments.name
FROM employees
JOIN departments ON employees.dept_id = departments.id;
```

Types of Joins:

- INNER JOIN: only matching rows
- LEFT JOIN: all from left + matches from right
- RIGHT JOIN: all from right + matches from left
- FULL JOIN: all from both sides

10. SQL FOR DATA SCIENCE

GROUP BY

- Used to group rows that have the same value in a column.

```sql
SELECT department, COUNT(*) AS total
FROM employees
GROUP BY department;
```

HAVING

- Used to filter grouped data (like WHERE but for groups).

```sql
SELECT department, COUNT(*) AS total
FROM employees
GROUP BY department
HAVING COUNT(*) > 10;
```

10.5 Subqueries and Window Functions

Subqueries

- A query inside another query.

Example:

```sql
SELECT name
FROM students
WHERE marks > (SELECT AVG(marks) FROM students);
```

Window Functions

- Used to perform calculations across a set of rows without grouping.

Common Window Functions:

- ROW_NUMBER()
- RANK()
- DENSE_RANK()
- SUM() OVER()

Example:

```sql
SELECT name, department, salary,
RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rank_in_dept
FROM employees;
```

- Here, employees are ranked within each department.

11. WORKING WITH REAL DATASETS

- In Data Science, we often work with real-world datasets that come in many formats — like CSV, Excel, JSON, or even from websites and APIs.
- This chapter teaches you how to get, load, and explore real data in Python.

11.1 Loading CSV/Excel/JSON

Loading CSV (Comma-Separated Values):

- CSV files are the most common format used for datasets.

```python
import pandas as pd

df = pd.read_csv("data.csv")
print(df.head())
```

You can also:

- Set a different separator: pd.read_csv("file.txt", sep="\t")
- Skip rows: skiprows=1
- Rename columns after loading

Loading Excel Files:

- Excel files can have multiple sheets.

```python
df = pd.read_excel("data.xlsx")
df2 = pd.read_excel("data.xlsx", sheet_name="Sheet2")
```

- Make sure to install openpyxl:

```bash
pip install openpyxl
```

Loading JSON Files:

- JSON (JavaScript Object Notation) is used for structured data, often from web APIs.

```python
import json

with open("data.json") as file:
    data = json.load(file)

# If it's a table-like structure
df = pd.json_normalize(data)
```

11. WORKING WITH REAL DATASETS

- If you get JSON from a URL:

```python
import requests

response = requests.get("https://api.example.com/data")
data = response.json()
df = pd.json_normalize(data)
```

11.2 Web Scraping Basics

- Web scraping means collecting data from websites.
- Always check the website's robots.txt file or terms of use before scraping.

Using BeautifulSoup:

```python
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
page = requests.get(url)
soup = BeautifulSoup(page.content, "html.parser")

titles = soup.find_all("h2")
for title in titles:
    print(title.text)
```

You can scrape:

- Titles, prices, headlines, reviews, etc.
- Tables and lists

Make sure to install:

```bash
pip install beautifulsoup4
```

11.3 APIs and JSON Handling

- An API (Application Programming Interface) lets you ask for data from websites in a structured way, usually as JSON.
- Example: Using a Public API

```python
import requests

response = requests.get("https://api.agify.io/?name=rudra")
data = response.json()
print(data)
```

Output:

```json
{'name': 'rudra', 'age': 19, 'count': 45}
```

11. WORKING WITH REAL DATASETS

You can get:

- Weather data
- Stock prices
- Sports scores
- News headlines

Handling JSON in Python:

```python
import json

json_data = '{"name": "Ravi", "age": 22}'
parsed = json.loads(json_data)
print(parsed["name"])
```

You can also convert Python to JSON:

```python
my_data = {"name": "Sneha", "score": 95}
json_string = json.dumps(my_data)
```

11.4 Open Datasets Resources

- Here are some websites where you can download free datasets for learning:

Platform	What it offers
Kaggle	Real-world datasets and ML competitions (kaggle.com)
UCI ML Repository	Classic datasets for learning (archive.ics.uci.edu)
Google Dataset Search	A search engine for datasets (datasetsearch.research.google.com)
Data.gov	Government open datasets
Awesome Public Datasets (GitHub)	A giant list of datasets in one place
FiveThirtyEight	Journalistic data used in real articles (fivethirtyeight.com)

12. TIME SERIES ANALYSIS

- Time Series data means data collected over time — daily, monthly, yearly, etc.

Examples:

- Stock prices
- Weather data
- Website visits
- Electricity usage

12.1 DateTime Handling in Pandas

- Pandas makes it easy to work with date and time.

Convert to DateTime:

```python
import pandas as pd

df["date"] = pd.to_datetime(df["date"])
```

Now you can:

- Sort by date
- Filter by month/year
- Group by time

- Extract parts of the date

```python
df["year"] = df["date"].dt.year
df["month"] = df["date"].dt.month
df["weekday"] = df["date"].dt.day_name()
```

- Set date as index

```python
df.set_index("date", inplace=True)
```

- Now you can resample, plot trends, and aggregate by time easily.

Resampling:

- Used to convert data from daily to monthly, weekly to yearly, etc.

```python
# Monthly average
monthly_avg = df["sales"].resample("M").mean()
```

12. TIME SERIES ANALYSIS

12.2 Trend, Seasonality, Noise

- A Time Series has 3 key components:

Component	Meaning
Trend	Overall long-term increase or decrease
Seasonality	Repeating pattern (weekly, monthly, yearly)
Noise	Random ups and downs (not predictable)

Visualizing Components:

```python
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df["sales"], model="additive", period=12)
result.plot()
```

- trend: Shows overall direction
- seasonal: Repeats at regular interval
- resid: Noise or leftover data

12.3 Moving Averages

- A Moving Average smooths the data by averaging values over a window.
- Helps to remove noise and highlight trends.

Simple Moving Average (SMA):

```python
df["sma_7"] = df["sales"].rolling(window=7).mean()
```

- This shows the 7-day average of sales.

Exponential Moving Average (EMA):

- Gives more weight to recent data.

```python
df["ema_7"] = df["sales"].ewm(span=7, adjust=False).mean()
```

- EMA reacts faster to recent changes.

12.4 ARIMA Basics

ARIMA stands for:

- AR – Auto Regression (use past values)
- I – Integrated (make data stationary by differencing)
- MA – Moving Average (use past errors)

12. TIME SERIES ANALYSIS

ARIMA Model

- Used to forecast future values in a time series.

Steps:

1. Make the data stationary (no trend or seasonality)
2. Find best (p, d, q) values:
    - p = lag observations (AR part)
    - d = differencing needed
    - q = lagged forecast errors (MA part)
3. Train ARIMA model

Example:

```python
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(df["sales"], order=(2,1,2))  # (p, d, q)
model_fit = model.fit()
forecast = model_fit.forecast(steps=10)
print(forecast)
```

- This gives the next 10 predicted values.

13. DEEP LEARNING INTRODUCTION

- Deep Learning is a part of Machine Learning that uses artificial neural networks — computer systems inspired by how the human brain works.

It's used in tasks like:

- Image recognition
- Voice assistants
- Language translation
- Chatbots (like me!)

13.1 Neural Network Basics

- What is a Neural Network?
- A neural network is made up of layers of nodes (neurons).

Layer Type	Purpose
Input Layer	Takes in data (e.g., images, numbers)
Hidden Layers	Do calculations to find patterns
Output Layer	Gives the prediction (e.g., Yes/No)

- Each neuron is connected to others and has a weight — just like how brain neurons pass signals.

How it works:

1. Input data (like an image)
2. Passes through layers of neurons
3. Each neuron:
    - Applies a function to inputs
    - Sends output to next layer
4. Final output is produced (like "Dog" or "Not Dog")

13.2 Activation Functions

- Activation functions decide whether a neuron should be "fired" (activated) or not.
- They add non-linearity — so the model can learn complex patterns.

Common Activation Functions:

Function	Purpose	Graph Shape
ReLU (Rectified Linear Unit)	Most common, fast	0 if input < 0, else same value
Sigmoid	Gives output between 0 and 1	Used in binary classification
Tanh	Output between -1 and 1	Like Sigmoid but centered

13. DEEP LEARNING INTRODUCTION

Example (ReLU):

```python
def relu(x):
    return max(0, x)
```

- Most deep learning models use ReLU in hidden layers.

13.3 Loss Functions

- A loss function tells how far the prediction is from the truth.
- The model tries to minimize this loss while learning.

Common Loss Functions:

Name	Used for
Mean Squared Error (MSE)	Regression problems
Binary Crossentropy	Binary classification (yes/no)
Categorical Crossentropy	Multiclass problems

Example:

```python
loss = actual - predicted
squared_loss = loss ** 2
```

- The model adjusts its weights to reduce the loss using an algorithm like gradient descent.

13.4 Introduction to TensorFlow & Keras

What is TensorFlow?

- TensorFlow is an open-source deep learning library developed by Google.
- It's used to build, train, and deploy models — especially for big tasks like image or speech recognition.

What is Keras?

- Keras is a simple front-end to TensorFlow.
- It makes building models faster and easier, like a shortcut.

13. DEEP LEARNING INTRODUCTION

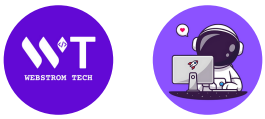
Example: Simple Neural Network in Keras:

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(16, activation='relu', input_shape=(10,)),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

- Dense() → creates a fully connected layer
- relu, sigmoid → activation functions
- compile() → sets optimizer & loss
- fit() → trains the model



14. PROJECTS AND PRACTICE IDEAS

- Practicing projects is the best way to learn and grow in Data Science and Machine Learning.
- This chapter helps you understand how to build a full ML project, where to get real-world datasets, and how to prepare for serious projects like capstones or Kaggle challenges.

14.1 End-to-End ML Project Structure

- A full ML project usually follows these 8 steps:

Step 1: Define the Problem

- Understand the goal.

Example: "Can we predict house prices?"

Step 2: Collect the Data
Get data from:

- CSV/Excel files
- APIs
- Web scraping
- Open datasets

Step 3: Explore the Data (EDA)
Use charts and statistics to:

- Spot trends
- Find missing values
- Understand distributions

Step 4: Preprocess the Data

- Handle missing values
- Convert text to numbers
- Scale/normalize features
- Create new useful features

Step 5: Split the Data
Split into:

- Training set (80%)
- Test set (20%)

Step 6: Train Models
Try different algorithms:

- Logistic Regression
- Random Forest
- SVM, etc.

14. PROJECTS AND PRACTICE IDEAS

Step 7: Evaluate Models

- Check accuracy, precision, recall, and F1-score.
- Use cross-validation.

Step 8: Improve and Deploy

- Tune hyperparameters
- Try ensemble models
- Deploy using tools like Flask, Streamlit, or cloud platforms

14.2 Kaggle Competitions

Kaggle is a popular platform for:

- ML competitions
- Datasets
- Notebooks (code examples)
- Learning resources

Beginner Competitions:

- Titanic: Predict survival
- House Prices: Predict house cost
- Digit Recognizer: Handwriting recognition

Each competition has:

- A dataset
- A leaderboard
- Public notebooks (to learn from others)

How to Start on Kaggle:

1. Sign up at kaggle.com
2. Go to "Competitions" → Select a beginner-level one
3. Download the dataset
4. Build a notebook using what you've learned
5. Submit your predictions to see your rank

14.3 Real-world Dataset Sources

- Here are some top places to find real datasets:

14. PROJECTS AND PRACTICE IDEAS

14.3 Real-world Dataset Sources

- Here are some top places to find real datasets:

Source	Description
Kaggle Datasets	Real data shared by community
UCI ML Repository	Classic datasets for learning
Google Dataset Search	Search engine for public datasets
data.gov	Indian and US government datasets
Awesome Public Datasets (GitHub)	A list of dataset links
FiveThirtyEight	Journalistic datasets
World Bank Data	Global economic and development data

14.4 Capstone Project Tips

- A capstone is a final project that combines everything you've learned.

Capstone Project Ideas:

- Predict customer churn for a telecom company
- Forecast sales for a store
- Sentiment analysis of movie or product reviews
- Classify whether a news article is real or fake
- Predict heart disease based on medical info

Tips for Success:

1. Pick a topic you care about
- You'll stay motivated.
2. Use a real-world dataset
- It makes your project more impressive.
3. Document everything clearly
- Write what you're doing and why.
4. Visualize your results
- Use graphs, confusion matrix, feature importance, etc.
5. Put it on GitHub
- Share your code and project for jobs or portfolio.
6. Try deployment
- Use tools like Streamlit, Flask, or Gradio to turn your model into an app.

15. TOOLS & ENVIRONMENT FOR DATA SCIENCE

- To work efficiently in Data Science, you need the right tools and setup. This chapter helps you understand the most used environments, tools, and shortcuts to boost your productivity.

15.1 Jupyter Notebook Tips

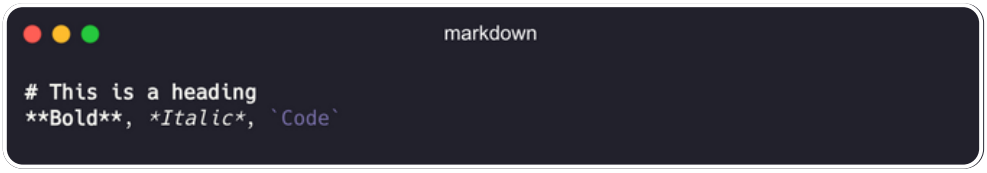
- What is Jupyter Notebook?

Jupyter Notebook is a web-based tool that lets you:

- Write code
- See outputs immediately
- Add notes and charts
- Great for testing and exploring data

Basic Tips:

- Use Shift + Enter to run a cell
- Use # to add comments
- Use Markdown to write titles and notes



Useful Magic Commands:

Command	What it does
%time	Check how long a line takes to run
%matplotlib inline	Show plots inside notebook
!ls, !pwd	Run terminal commands in a cell

Auto-complete:

- Press Tab to auto-complete function names or see options.

Keyboard Shortcuts:

Shortcut	Action
A	Insert cell above
B	Insert cell below
M	Change cell to Markdown
Y	Change cell to Code
DD (Double D)	Delete current cell

15. TOOLS & ENVIRONMENT FOR DATA SCIENCE

15.2 Git & GitHub for Data Science

What is Git?

- Git is a version control tool that helps you track changes in your code and projects.

What is GitHub?

- GitHub is a website to store and share your code using Git.

Perfect for:

- Saving progress
- Showing your projects to others
- Working with a team

Basic Git Commands:

```bash
git init          # Start tracking a folder
git add .         # Add all files
git commit -m "message"   # Save with a message
git push          # Upload to GitHub
```

Steps to Use GitHub:

1. Create account on github.com
2. Create a new repository
3. Connect local folder using:

```bash
git remote add origin <your-repo-url>
git push -u origin main
```

What to Upload:

- Jupyter notebooks
- CSV/Excel datasets
- ReadMe file to explain project
- Screenshots of results

15.3 Virtual Environments (venv, conda)

Why use virtual environments?

- Each project may need different versions of libraries.
- A virtual environment keeps them separate and avoids errors.

Using venv (built-in in Python)

15. TOOLS & ENVIRONMENT FOR DATA SCIENCE

Using venv (built-in in Python)

```bash
python -m venv myenv
source myenv/bin/activate   # On Mac/Linux
myenv\Scripts\activate      # On Windows
```

Install packages in it:

```bash
pip install pandas
```

Deactivate:

```bash
deactivate
```

Using conda (from Anaconda)

- Conda is a package manager and environment tool, widely used in data science.

```bash
conda create --name myenv pandas numpy
conda activate myenv
```

You can install tools like Jupyter, Scikit-learn, TensorFlow easily with:

```bash
conda install jupyter scikit-learn
```

15.4 VS Code & Notebook Shortcuts

- *VS Code for Data Science*

VS Code (Visual Studio Code) is a popular code editor that supports:

- Python
- Jupyter Notebooks
- GitHub integration
- Extensions for data science

Useful Extensions:

- Python (official by Microsoft)
- Jupyter
- GitLens
- Pylance (code suggestions)
- Material Theme (for better look)

15. TOOLS & ENVIRONMENT FOR DATA SCIENCE

VS Code Shortcuts:

Shortcut	Action
Ctrl + Shift + P	Open Command Palette
Ctrl + /	Comment/Uncomment a line
Alt + Shift + ↓/↑	Copy line up/down
Ctrl + B	Toggle sidebar
Ctrl + J	Toggle terminal
Shift + Enter (in .ipynb)	Run cell

Run Jupyter Notebooks in VS Code:

1. Open .ipynb file
2. Use Run Cell button or Shift + Enter
3. Outputs will appear right below the code

16. RESOURCES FOR LEARNING DATA SCIENCE

- Whether you're just starting or want to go deeper, the right resources can speed up your learning and make it easier to stay motivated.
- This chapter lists trusted blogs, YouTube channels, courses, cheat sheets, and books that even a beginner can understand.

16.1 Blogs, YouTube Channels, and Courses

Top Blogs to Follow:

Blog Name	What It Offers
Towards Data Science (Medium)	Beginner to advanced articles with real examples
Analytics Vidhya	Indian-focused content, competitions, career tips
DataCamp Blog	Concepts explained simply with visuals
KDnuggets	Latest trends, tutorials, interviews
Built In	Real-world use cases and business insights

Best YouTube Channels:

Channel Name	Why It's Great
Krish Naik	Indian data scientist, hands-on projects and ML concepts
Ken Jee	Career advice + practical projects
StatQuest with Josh Starmer	Simple explanations of complex math concepts
freeCodeCamp.org	Full free courses (Python, ML, Deep Learning)
Tech with Tim	Python coding and deep learning basics

Recommended Online Courses (Free & Paid):

Platform	Course	Good For
Coursera	IBM Data Science, Andrew Ng ML	Full career path
Udemy	Python for Data Science, ML A-Z	Hands-on coding
edX	Harvard's Data Science Series	Solid theoretical base
DataCamp	Skill tracks for DS, ML, NLP	Bite-sized practice
Kaggle Learn	Free mini-courses (Pandas, EDA)	Interactive + Free

16. RESOURCES FOR LEARNING DATA SCIENCE

16.2 Cheat Sheets & PDF Summaries

- Cheat sheets are quick reference guides that summarize important syntax and concepts.

Best Cheat Sheets for Beginners:

Topic	Where to Find
Python Basics	Real Python, GitHub cheat repos
Pandas & NumPy	DataCamp, Towards Data Science
Matplotlib & Seaborn	Official docs and GitHub visual guides
Scikit-Learn	Scikit-learn official cheatsheet
Machine Learning	Stanford ML summary PDFs, Kaggle cheatbooks
Statistics	Simplified PDF guides from Khan Academy
SQL	Mode Analytics, W3Schools printable sheets

You can also make your own custom cheat sheets using tools like:

- Notion
- Canva
- Google Docs

17.3 Books to Read

- These books cover core concepts, real projects, and mathematical understanding — all in simple language.

Beginner-Friendly Books:

Book Title	Why It's Useful
Python for Data Analysis – Wes McKinney	Best for learning Pandas & data wrangling
Data Science from Scratch – Joel Grus	Build basic algorithms by hand
Hands-On ML with Scikit-Learn & TensorFlow – Aurélien Géron	Covers real projects & deep learning
Think Stats – Allen Downey	Statistics explained with code
Storytelling with Data – Cole Knaflic	Great for learning data visualization
Naked Statistics – Charles Wheelan	Statistics in plain English

Was this post helpful ?

Follow Our 2nd Account

Follow For More

Tap Here