Data Analytics with Python

Beginner's Plain-English Summary

What it is · Why it matters · How it helps YOU

No maths. No jargon. Just clear, simple explanations.

■ What is this subject about?

This subject is about teaching a computer to **learn from data** and make **smart predictions** — using Python code. Think of it as giving your computer a brain that can study past information and then answer questions like: *'Will this customer leave?'*, *'What will the sales be next month?'*, or *'Is this email spam?'*

■ Part 1	**Predictive Modelling**	Teaching computers to predict things
■■ Part 2	**Advanced Techniques**	Smarter & more powerful methods
■ Part 3	**Evaluation & Tuning**	Checking & improving your model

PART 1

Predictive Modelling

The basics — how computers learn to make predictions

Part 1 is your **foundation**. Before you build anything fancy, you need to understand what machine learning actually is and how the basic tools work. Think of this part as learning to drive before getting on the highway.

1.1 What is Predictive Modelling?

Real-world analogy

Imagine you work at a bank. You want to know: 'Will this person repay their loan?' You look at thousands of past customers — their age, salary, job — and whether they repaid or not. You find patterns. Predictive modelling lets Python do this automatically, on millions of rows, in seconds.

Two main types of learning:

Supervised Learning — you give the computer examples WITH the right answers (e.g. 1000 emails already labelled spam/not-spam). It learns the pattern and can label new emails automatically.

Unsupervised Learning — no right answers given. The computer finds hidden groups on its own (e.g. grouping customers by behaviour without telling it how many groups to make).

Real-world uses:

- Netflix recommending your next show
- Banks deciding whether to approve a loan
- Hospitals predicting which patients need urgent care
- Shops forecasting how much stock to order next week

1.2 The Machine Learning Pipeline

Every ML project follows the **same 9-step recipe**. Skip a step and your model will give wrong results — even if the code runs without errors.

1	**Define the Problem**	What question are you trying to answer?
2	**Collect Data**	Get your historical data (CSV, database, web)
3	**Explore the Data**	Look at distributions, spot weird values, find patterns
4	**Clean & Prepare**	Fix missing values, encode text columns, scale numbers

5	**Choose a Model**	Pick the right algorithm for your problem type
6	**Train**	Feed training data to the model so it learns
7	**Evaluate**	Test on new data — how accurate is it really?
8	**Tune**	Tweak settings to squeeze out better performance
9	**Deploy**	Put it into production so real people can use it

1.3 Linear Regression

■ Simple analogy

You want to predict house prices. You notice that bigger houses cost more. Linear regression draws the best straight line through your data. Once the line is drawn, you just plug in any house size and read off the predicted price. That's it — the computer finds the best line automatically.

When to use it: When your answer is a number (price, temperature, sales).

Helpful for: House price prediction, sales forecasting, predicting exam scores.

1.4 Logistic Regression

✉■ Simple analogy

Despite the confusing name, this is for YES/NO questions. Is this email spam? Will this customer churn? It gives you a probability (e.g. 87% chance of spam) and then makes a decision based on a cutoff (usually 50%).

When to use it: When your answer is a category (Yes/No, Spam/Not spam).

Helpful for: Fraud detection, medical diagnosis, customer churn prediction.

1.5 Decision Trees

■ Simple analogy

A decision tree is like a game of 20 questions. 'Is the customer's salary > 50k?' → Yes → 'Are they older than 30?' → Yes → 'Predict: will buy.' The computer automatically finds the best questions to ask based on your data.

Why it's great for beginners: You can actually *see* why it made a decision — it prints out the rules in plain English.

1.6 – 1.7 Feature Engineering & Data Splitting

■ Feature Engineering analogy

Imagine cooking. You have raw ingredients (your data). Feature engineering is the prep work — chopping, marinating, combining. You might turn a 'date of birth' column into 'age' because the model understands numbers better than dates.

■ Data Splitting analogy

When you study for an exam, you practise on past papers (training data) and sit the real exam on questions you've never seen (test data). Data splitting does the same — you never let the model 'peek' at test data during training, otherwise you'd get false confidence in its accuracy.

■■ PART 2

Advanced Modelling Techniques

More powerful algorithms — still explained simply

Part 2 introduces more powerful tools. Each one solves a specific problem better than basic models. You don't need to master all of them — just understand **what each one is good at** so you can pick the right tool for the job.

2.1 Random Forest — Wisdom of the Crowd

■■ Simple analogy

One doctor's opinion might be wrong. Ask 300 doctors and take the majority vote — you get a much better answer. Random Forest builds hundreds of decision trees, each trained on slightly different data, then combines their votes. It is one of the most reliable 'out of the box' algorithms in data science.

Why it's helpful: Works well without much tuning. Handles messy data. Tells you which features (columns) matter most.

- ■ Credit risk scoring at banks
- ■ Medical diagnosis from patient data
- ■ Predicting customer churn for telecoms

2.2 Boosting — XGBoost & LightGBM

■■ Simple analogy

Imagine a student who gets a test wrong. Instead of moving on, they spend extra time specifically on the questions they got wrong. Boosting works the same way — each new model focuses on the mistakes the previous model made. XGBoost and LightGBM are the kings of this approach and win most data science competitions.

XGBoost is fast, accurate, and handles missing data automatically. **LightGBM** is even faster on very large datasets. Both are **the most popular choice for real-world tabular data problems.**

2.3 Support Vector Machines (SVM)

—■ Simple analogy

Imagine you have red and blue dots on a piece of paper. SVM draws the widest possible line between the two groups — maximising the gap (called the margin). Points closest to the line are called 'support vectors' and are the trickiest ones to classify correctly.

Best for: Image recognition, text classification, small-to-medium datasets. Not great for very large datasets (slow to train).

2.4 K-Nearest Neighbours (KNN)

■ Simple analogy

You move to a new city and want to know if your neighbourhood is safe. You ask your 7 nearest neighbours. If 5 of them say it's safe, you conclude it's safe. KNN works exactly like this — to classify a new point, it finds the K closest points in the training data and takes a majority vote.

Key rule: Always scale (normalise) your data before using KNN — otherwise features with bigger numbers will dominate the distance calculation.

2.5 Naive Bayes

■ Simple analogy

You receive an email containing the words 'FREE', 'WINNER', and 'CLICK NOW'. Based on past emails you've seen, you know these words appear mostly in spam. Naive Bayes calculates the probability of spam given those words and makes a decision. It's super fast and works amazingly well for text classification.

2.6 Neural Networks & Deep Learning

■ Simple analogy

Your brain has billions of neurons connected together. When you see a cat, signals pass through layers of neurons — edges → shapes → eyes/ears → 'cat!'. Neural networks copy this structure. Each layer learns something more complex than the last. Deep learning = many layers = very complex patterns.

Important beginner tip: For normal tables of data, XGBoost usually beats neural networks. Use neural networks for **images, text, audio, and video** — that's where they truly shine.

2.7 Clustering — Finding Hidden Groups

■■ Simple analogy

A supermarket has 1 million customers but no labels. K-Means groups them automatically: Group A = young, low spend; Group B = families, high spend; Group C = elderly, loyal. No one told the computer about these groups — it found them by itself. Now the marketing team can send different offers to each group.

2.8 Dimensionality Reduction — PCA & t-SNE

■■ Simple analogy

Imagine your dataset has 500 columns. That's overwhelming — even for a computer. PCA compresses those 500 columns into, say, 20 'super-columns' that still capture 95% of the important information. It's like making a ZIP file for your data. t-SNE is used purely for visualisation — it squishes high-dimensional data into a 2D scatter plot you can actually look at.

2.9 Time Series Modelling

■ Simple analogy

Yesterday's temperature affects today's. Last month's sales affect this month's. Time series models understand that **order matters** — you can't shuffle the rows randomly like with normal data. They detect trends (going up over time), seasonality (busy every December), and then predict what comes next.

- ■ Stock price forecasting
- ■ Electricity demand planning
- ■ Weather prediction
- ■ Monthly sales forecasting for retail

■ PART 3

Model Evaluation & Tuning

How to check if your model is actually good — and make it better

Building a model is easy. Building a model you can **trust** is hard. Part 3 teaches you how to measure performance honestly, spot problems, and squeeze the best results out of your model.

3.1 Classification Metrics — Measuring Accuracy

■ Why accuracy alone can be misleading

Imagine a disease affects 1% of people. A lazy model that always says 'Not sick' gets 99% accuracy — but it's completely useless! You need better metrics that care about catching the sick people (Recall) and not wrongly accusing healthy people (Precision).

Metric	Meaning	When to use
Accuracy	% of total predictions that are correct	*Use ONLY when classes are balanced*
Precision	Of all predicted positives, how many are real?	*Use when false alarms are costly (spam filter)*
Recall	Of all real positives, how many did we catch?	*Use when missing cases is costly (disease)*
F1-Score	Balance between Precision and Recall	*Best for imbalanced datasets*
AUC-ROC	Overall ability to separate two classes	*Higher = better; 1.0 is perfect*

3.2 Regression Metrics

When predicting a **number** (price, temperature, sales), you measure how far off your predictions are from the real values.

MAE (Mean Absolute Error) — average of your mistakes in the same unit as your target. Easy to understand: 'On average, we're off by £500.'

RMSE (Root Mean Squared Error) — similar but punishes big mistakes more. A prediction that's wrong by £10,000 is penalised much more than ten predictions wrong by £1,000.

R^2 — scores from 0 to 1. An R^2 of 0.85 means 'our model explains 85% of the variation in the data.' Closer to 1 = better.

3.3 Cross-Validation

■ Simple analogy

Instead of having ONE exam to judge a student, you give them 5 different exams on different topics and average the scores. Cross-validation does the same — it tests your model on 5 different portions of your data and averages the results. This gives a much fairer picture of real performance.

3.4 Bias vs Variance — The Key Trade-off

■ Simple analogy

Imagine throwing darts at a target. HIGH BIAS = all darts land in the same wrong spot (consistently wrong = underfitting). HIGH VARIANCE = darts scatter all over the place (inconsistent = overfitting). The goal is low bias AND low variance — consistently hitting the bullseye.

Problem	Symptom	Fix
High Bias (Underfitting)	Bad on training data AND test data	*Use a more complex model, add more features*
High Variance (Overfitting)	Great on training data, bad on test data	*Add regularisation, get more data, simplify model*

3.5 Hyperparameter Tuning

■■ Simple analogy

Think of a model as a coffee machine with lots of dials — water temperature, grind size, brew time. Hyperparameter tuning is adjusting those dials until you get the best coffee (model). You can try every combination (Grid Search), try random combinations (Random Search), or use a smart algorithm that learns which dials matter most (Bayesian/Optuna).

3.6 Regularisation

■■ Simple analogy

Regularisation is like penalising a student for writing an unnecessarily long answer. The model is penalised for being too complex. This forces it to focus only on the most important features and prevents overfitting. L1 (Lasso) can remove features entirely. L2 (Ridge) shrinks all features a little.

3.7 Pipelines — Automation & Safety

■ Simple analogy

A pipeline is like a factory assembly line. Raw data goes in one end and predictions come out the other. Each step (clean → scale → encode → model) happens automatically in the right order. The big benefit: it prevents accidental data leakage and makes your code much cleaner and reusable.

3.8 Model Interpretability — SHAP

■ Simple analogy

Your XGBoost model says 'This customer will churn — 91% probability.' Great, but WHY? SHAP opens the black box and tells you: 'The customer's low usage (+0.3), short contract (-0.2), and three complaints (+0.4) all pushed the prediction toward churn.' This is critical in banking, healthcare, and legal situations.

3.9 Handling Imbalanced Data

■■ Simple analogy

You're training a fraud detection model. 999 out of 1000 transactions are normal. The model could just predict 'not fraud' every time and be 99.9% accurate — completely useless! SMOTE creates synthetic examples of the rare class (fraud) so the model actually learns what fraud looks like.

3.10 Model Deployment

■ Simple analogy

You've trained the perfect model on your laptop. Now what? Deployment means saving the model to a file (like joblib) and wrapping it in an API using FastAPI so any app — a website, mobile app, or dashboard — can send data and get predictions back in real time. This is where your data science work becomes a real product.

- A website sends a customer's details → your API returns 'churn risk: 87%'
- A bank's app sends a transaction → your API returns 'fraud: yes/no'
- A hospital system sends test results → your API returns a risk score

■ SUMMARY

The Big Picture

Everything you learned — in one page

■ Python + Data = the ability to turn raw numbers into smart decisions.

■ Part 1	**Predictive Modelling**	You learned HOW computers learn — regression predicts numbers, classification predicts categories, decision trees make transparent decisions.
■ Part 2	**Advanced Techniques**	You learned WHICH algorithm to pick — Random Forest for reliability, XGBoost for competition, Neural Networks for images/text, Clustering for hidden groups.
■ Part 3	**Evaluation & Tuning**	You learned HOW to TRUST your model — use the right metrics, prevent overfitting with regularisation, explain predictions with SHAP, and deploy with FastAPI.

■ Key golden rule: Split data before preprocessing · Pick metrics that match your problem · Always explain your model's decisions

You now have a complete mental map of Data Analytics with Python. ■