90-Day Data Science / Machine Learning Roadmap for Beginners Targeting ₹10 LPA

This **90-day roadmap** is a comprehensive, strategic guide for absolute beginners with no prior experience, aiming to secure entry-level roles like **Junior Data Scientist**, **Machine Learning Engineer**, or **Advanced Data Analyst** in India with a salary of ₹10 LPA. It assumes **6–8 hours of daily effort** and covers Python programming, essential mathematics, data manipulation, visualization, machine learning, portfolio building, model deployment, and job search strategies. Each day includes detailed tasks, time estimates, explanations, beginner-friendly resources, practical tips, and milestones to ensure clarity and progress. The roadmap builds a robust portfolio with 5+ projects, prepares you for technical and behavioral interviews, and positions you for job applications in the Indian market (e.g., startups, MNCs like TCS, Infosys, ZS Associates, or analytics firms like Fractal Analytics).

Prerequisites and Setup

- **Hardware**: Laptop with 8GB+ RAM (Windows, macOS, or Linux) to handle Python and data processing.
- **Software**: Install Python 3.10+, Anaconda (includes Jupyter Notebook, Pandas, etc.), and VS Code (all free).
- **Accounts**:
    - **GitHub**: For hosting projects and version control.
    - **LinkedIn**: For networking and job applications.
    - **Kaggle**: For datasets and competitions.
    - **HackerRank/LeetCode**: For coding practice.
    - **Naukri/Instahyre/Wellfound**: For job applications.
- **Tools**: Use Notion, Trello, or a notebook to track tasks, errors, and progress. Create a spreadsheet (Excel/Google Sheets) to log job applications (columns: Company, Role, Date Applied, Status).
- **Mindset**: Commit to 6–8 hours daily, embrace errors as learning opportunities, and engage with communities (Reddit's r/datascience, Stack Overflow, LinkedIn groups, Discord data science channels).
- **Indian Job Market Context**: Focus on skills for roles at companies like TCS, Infosys, Accenture, ZS Associates, Fractal Analytics, or startups on platforms like Naukri,

Instahyre, and Wellfound. Common requirements include Python, SQL, ML basics, and project experience.

Month 1: Foundations in Python, Math, EDA, and Visualization (Days 1–30)

Focus: Build a strong foundation in Python, essential mathematics, data manipulation, and visualization. Complete two exploratory data analysis (EDA) projects to start your portfolio.

Week 1: Python Programming Foundations (Days 1–7)

Goal: Master Python basics, set up tools, and understand core programming concepts for data science.

- **Day 1: Environment Setup and Python Introduction**
  - **Tasks** (6 hours):
    - **Setup Tools** (1.5 hours): Install Anaconda (includes Python, Jupyter Notebook, and libraries like Pandas). Install VS Code for code editing. Open Jupyter Notebook, create a new notebook, and run `print("Hello, World!")` to verify setup.
    - **Learn Python Basics** (2.5 hours): Study variables (e.g., `x = 5`), data types (int, float, str, bool), basic operations (+, -, *, /), and `print()`. Watch a beginner-friendly video for context.
    - **Practice** (2 hours): Solve 10 easy problems on HackerRank (e.g., calculate area of a rectangle, swap two variables without a temp variable).
  - **Why It Matters**: Python is the primary language for data science, and a proper setup ensures smooth learning.
  - **Resources**:
    - Anaconda Installation (follow OS-specific instructions).
    - FreeCodeCamp Python Tutorial (watch 0:00–1:00).
    - W3Schools Python Basics (Variables, Data Types).
    - HackerRank Python (select “Easy” problems).
  - **Tips**:
    - Create a GitHub repository named “DataScienceJourney” and save your first notebook as `day1_basics.ipynb`.
    - If installation fails, search errors on Stack Overflow (e.g., “Anaconda install error Windows”).

        - Write new terms (e.g., “variable,” “string”) in a notebook for revision.
        - Dedicate 10 minutes to set up a Notion/Trello board to track daily tasks.
    - **Time Breakdown**:
        - 9:00–10:30 AM: Setup tools.
        - 10:30 AM–1:00 PM: Learn basics, watch video.
        - 2:00–4:00 PM: Practice problems.
    - **Milestone**: Python environment set up, first code executed, 10 problems solved, GitHub repository created.
- **Day 2: Control Structures (Conditionals and Loops)**
    - **Tasks** (6 hours):
        - **Learn Conditionals** (2 hours): Study `if`, `elif`, `else` statements. Understand indentation (Python’s way of defining code blocks). Example: `if x > 0: print("Positive")`.
        - **Learn Loops** (2 hours): Study `for` loops (e.g., `for i in range(5):`) and `while` loops (e.g., `while x < 10:`). Learn `break` and `continue`.
        - **Practice** (2 hours): Solve 10 problems (e.g., check if a number is prime, print first 10 Fibonacci numbers) on HackerRank or LeetCode.
    - **Why It Matters**: Conditionals and loops control program flow, essential for data processing (e.g., filtering rows in a dataset).
    - **Resources**:
        - Automate the Boring Stuff with Python (Chapter 2, free online).
        - HackerRank Python Challenges (select “Easy”).
        - LeetCode Python Problems.
    - **Tips**:
        - Test code snippets in Jupyter Notebook for instant feedback.
        - Use `print()` inside loops to debug outputs (e.g., `print(i)` in a loop).
        - Save solutions as `day2_control_structures.ipynb` and commit to GitHub.
        - Log any errors (e.g., “IndentationError”) and their fixes in your notebook.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Learn conditionals.
        - 11:00 AM–1:00 PM: Learn loops.
        - 2:00–4:00 PM: Practice problems.
    - **Milestone**: Understand `if` statements and loops, solve 10 problems, commit code to GitHub.
- **Day 3: Functions and Modular Coding**
    - **Tasks** (6 hours):

        - **Learn Functions** (2.5 hours): Study defining functions (`def my_function(arg):`), arguments (positional, keyword), return statements, and `lambda` functions (e.g., `square = lambda x: x**2`). Understand scope (local vs. global).
        - **Learn Modularity** (1.5 hours): Explore why functions make code reusable. Study built-in functions like `map()` and `filter()`.
        - **Practice** (2 hours): Write 5 functions (e.g., calculate BMI, factorial, check palindrome) and test in Jupyter.
    - **Why It Matters**: Functions reduce code duplication and are critical for writing reusable data science scripts (e.g., data cleaning functions).
    - **Resources**:
        - Real Python: Functions.
        - Python Crash Course (Chapter 8, or find PDF online).
        - GeeksforGeeks: Python Functions.
    - **Tips**:
        - Start with simple functions, then try `lambda` for concise code.
        - Add comments (e.g., `# Calculates factorial of n`) to explain function purpose.
        - Save as `day3_functions.ipynb` and commit to GitHub.
        - Test functions with different inputs to ensure robustness.
    - **Time Breakdown**:
        - 9:00–11:30 AM: Learn functions.
        - 11:30 AM–1:00 PM: Learn modularity.
        - 2:00–4:00 PM: Practice writing functions.
    - **Milestone**: Write and test 5 functions, understand modularity, commit to GitHub.
- **Day 4: Data Structures**
    - **Tasks** (6 hours):
        - **Learn Data Structures** (3 hours): Study lists (ordered, mutable, e.g., `[1, 2, 3]`), dictionaries (key-value, e.g., `{"name": "Alice", "age": 25}`), sets (unique elements, e.g., `{1, 2, 3}`), and tuples (immutable, e.g., `(1, 2, 3)`). Learn methods: `append()`, `pop()`, `keys()`, `add()`, etc.
        - **Practice** (3 hours): Solve 10 problems per structure (e.g., reverse a list, count word frequency with dictionary, find unique elements with set) on HackerRank or LeetCode.
    - **Why It Matters**: Data structures store and manipulate data in data science (e.g., lists for features, dictionaries for metadata).
    - **Resources**:
        - Python Crash Course (Chapters 5–7).
        - GeeksforGeeks: Data Structures.
        - Real Python: Lists and Dictionaries.

    - **Tips**:
        - Visualize dictionaries as key-value tables.
        - Practice list comprehensions (e.g., `[x**2 for x in range(5)]`) for efficiency.
        - Save as `day4_data_structures.ipynb` and commit to GitHub.
        - Log common methods (e.g., `list.append()`) in your notebook.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn data structures.
        - 1:00–4:00 PM: Practice problems.
    - **Milestone**: Master lists, dictionaries, sets, tuples; solve 40 problems, commit to GitHub.
- **Day 5: File Handling and Error Handling**
    - **Tasks** (6 hours):
        - **Learn File I/O** (2.5 hours): Study reading/writing text files (`open()`, `read()`, `write()`) and CSVs with `pandas.read_csv()`. Use `with` statements for safe file handling (e.g., `with open('file.txt', 'r') as f:`).
        - **Learn Exceptions** (1.5 hours): Study `try`, `except`, `finally` for handling errors (e.g., `FileNotFoundError`, `ValueError`).
        - **Practice** (2 hours): Download a sample CSV (e.g., Iris from Kaggle), read it, handle errors, and write processed data to a new CSV.
    - **Why It Matters**: Data science involves loading datasets from files; error handling ensures robust code.
    - **Resources**:
        - Real Python: File I/O.
        - Kaggle Python Course (File I/O section).
        - Pandas: Reading CSV.
    - **Tips**:
        - Save the sample CSV locally and test reading it.
        - Log errors and fixes (e.g., “FileNotFoundError: Check file path”) in your notebook.
        - Save as `day5_file_handling.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:30 AM: Learn file I/O.
        - 11:30 AM–1:00 PM: Learn exceptions.
        - 2:00–4:00 PM: Practice file handling and error handling.
    - **Milestone**: Read/write files, handle errors, commit to GitHub.
- **Day 6: NumPy Fundamentals**
    - **Tasks** (6 hours):

        - **Learn NumPy** (3 hours): Install NumPy (`pip install numpy`). Study arrays (e.g., `np.array([1, 2, 3])`), indexing/slicing (e.g., `arr[0:2]`), broadcasting, and operations (dot product, reshape).
        - **Practice** (3 hours): Solve 10 problems (e.g., normalize an array, multiply matrices) using NumPy on HackerRank or in Jupyter.
    - **Why It Matters**: NumPy enables fast numerical computations for large datasets, a core data science tool.
    - **Resources**:
        - NumPy Quickstart.
        - DataCamp NumPy Cheat Sheet.
        - GeeksforGeeks: NumPy.
    - **Tips**:
        - Avoid loops; use vectorized operations (e.g., `arr * 2` instead of `for` loop).
        - Test arrays in Jupyter for quick feedback.
        - Save as `day6_numpy.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn NumPy.
        - 1:00–4:00 PM: Practice problems.
    - **Milestone**: Create and manipulate NumPy arrays, solve 10 problems, commit to GitHub.
- **Day 7: Pandas Basics**
    - **Tasks** (6 hours):
        - **Learn Pandas** (3 hours): Install Pandas (`pip install pandas`). Study Series (1D arrays) and DataFrames (2D tables). Practice `head()`, `tail()`, `info()`, filtering (e.g., `df[df['age'] > 30]`), and column operations (e.g., `df['new_col'] = df['col'] * 2`).
        - **Practice** (3 hours): Load a CSV (e.g., Iris), filter rows, compute column stats (mean, max).
    - **Why It Matters**: Pandas is the primary library for data manipulation in data science.
    - **Resources**:
        - Pandas Getting Started.
        - Kaggle Pandas Course.
        - Real Python: Pandas.
    - **Tips**:
        - Download a Pandas cheat sheet for quick reference.
        - Save processed DataFrame as `iris_processed.csv` for practice.
        - Save as `day7_pandas.ipynb` and commit to GitHub.
    - **Time Breakdown**:

        - 9:00–12:00 PM: Learn Pandas.
        - 1:00–4:00 PM: Practice data manipulation.
    - **Milestone**: Load, filter, and analyze a dataset with Pandas, commit to GitHub.

Week 2: Advanced Pandas, Statistics, and First EDA Project (Days 8–14)

Goal: Deepen Pandas skills, learn foundational statistics, and complete an EDA project for your portfolio.

- **Day 8: Advanced Pandas**
    - **Tasks** (6 hours):
        - **Learn Advanced Functions** (3 hours): Study `groupby()` (e.g., `df.groupby('category').mean()`), `merge()` (join datasets), `pivot_table()` (summarize), and `apply()` (custom functions). Handle missing data with `isna()`, `fillna()`, `dropna()`.
        - **Practice** (3 hours): Use a dataset (e.g., Titanic from Kaggle) to group by a column, merge two DataFrames, and handle missing values.
    - **Why It Matters**: Advanced Pandas enables complex data wrangling, a critical skill for analysis.
    - **Resources**:
        - Pandas User Guide (GroupBy, Merging).
        - Kaggle Pandas Course (advanced sections).
        - Towards Data Science: Pandas.
    - **Tips**:
        - Save intermediate DataFrames (e.g., `df_cleaned.csv`) to track changes.
        - Test `merge()` with small datasets to avoid errors.
        - Save as `day8_advanced_pandas.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn advanced Pandas functions.
        - 1:00–4:00 PM: Practice with dataset.
    - **Milestone**: Perform grouping, merging, and missing data handling, commit to GitHub.
- **Day 9: Descriptive Statistics**
    - **Tasks** (6 hours):
        - **Learn Statistics** (3 hours): Study mean, median, mode, variance, standard deviation, interquartile range (IQR), and skewness. Use Pandas to compute (e.g., `df['column'].mean()`).
        - **Practice** (3 hours): Compute stats on a dataset (e.g., Titanic), visualize distributions with histograms using Matplotlib.

    - **Why It Matters**: Statistics provide insights into data and are used in ML evaluation.
    - **Resources**:
        - StatQuest: Descriptive Stats (search “statistics”).
        - Khan Academy: Statistics.
        - Pandas: Descriptive Stats.
    - **Tips**:
        - Use `df.describe()` for quick stats overview.
        - Compare mean vs. median for skewed data.
        - Save as `day9_statistics.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn statistics.
        - 1:00–4:00 PM: Practice and visualize.
    - **Milestone**: Compute and interpret basic statistics, create histograms, commit to GitHub.
- **Day 10: Probability Basics**
    - **Tasks** (6 hours):
        - **Learn Probability** (3 hours): Study probability rules (addition, multiplication), independent/dependent events, normal and binomial distributions, and expected value.
        - **Practice** (3 hours): Solve 5 problems (e.g., probability of rolling two 6s) and simulate distributions with SciPy (`pip install scipy`, e.g., `stats.norm.rvs()`).
    - **Why It Matters**: Probability is foundational for ML algorithms like Naive Bayes and understanding uncertainty.
    - **Resources**:
        - Probability for Data Science (or find PDF online).
        - 3Blue1Brown: Probability.
        - SciPy Stats Docs.
    - **Tips**:
        - Focus on intuition (e.g., normal distribution as a bell curve).
        - Plot distributions with `seaborn.histplot()` for visualization.
        - Save as `day10_probability.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn probability.
        - 1:00–4:00 PM: Practice and simulate.
    - **Milestone**: Understand probability rules, simulate distributions, commit to GitHub.
- **Day 11: Matplotlib Visualization**

    - **Tasks** (6 hours):
        - **Learn Matplotlib** (3 hours): Install Matplotlib (`pip install matplotlib`). Create bar, pie, histogram, scatter plots, and subplots. Customize with titles, labels, and colors.
        - **Practice** (3 hours): Create 5 plots on a dataset (e.g., Titanic or Iris) using Matplotlib.
    - **Why It Matters**: Visualizations communicate insights to stakeholders, a key data science skill.
    - **Resources**:
        - Matplotlib Tutorials.
        - DataCamp Matplotlib Cheat Sheet.
        - Real Python: Matplotlib.
    - **Tips**:
        - Save plots with `plt.savefig('plot.png')` for documentation.
        - Use `plt.tight_layout()` for clean layouts.
        - Save as `day11_matplotlib.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn Matplotlib.
        - 1:00–4:00 PM: Practice visualizations.
    - **Milestone**: Create and customize 5 visualizations, commit to GitHub.
- **Day 12: Seaborn Visualization**
    - **Tasks** (6 hours):
        - **Learn Seaborn** (3 hours): Install Seaborn (`pip install seaborn`). Create `pairplot()`, `boxplot()`, `heatmap()`, and `distplot()` for statistical visualizations.
        - **Practice** (3 hours): Create 5 visualizations on a dataset (e.g., Titanic) using Seaborn.
    - **Why It Matters**: Seaborn simplifies professional, aesthetically pleasing visualizations.
    - **Resources**:
        - Seaborn Official Docs.
        - Kaggle Visualization Course.
        - Towards Data Science: Seaborn.
    - **Tips**:
        - Use Seaborn for quick plots, Matplotlib for customization.
        - Experiment with color palettes (e.g., `sns.set_palette('deep')`).
        - Save as `day12_seaborn.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn Seaborn.

        - 1:00–4:00 PM: Practice visualizations.
    - **Milestone**: Create 5 statistical visualizations with Seaborn, commit to GitHub.
- **Day 13: Plan First EDA Project**
    - **Tasks** (6 hours):
        - **Select Dataset** (1 hour): Choose a beginner-friendly Kaggle dataset (e.g., Titanic, Iris) with 5–10 columns, numerical/categorical data, and clear documentation.
        - **Explore Dataset** (3 hours): Load with Pandas (`pd.read_csv()`), use `df.info()`, `df.head()`, and `df.isna().sum()` to identify missing values, data types, and potential questions (e.g., "What factors predict Titanic survival?").
        - **Plan EDA** (2 hours): List 5–10 questions to answer, create a Jupyter Notebook with sections (Introduction, Data Loading, Cleaning, Analysis, Conclusion).
    - **Why It Matters**: EDA is the first step in data science projects, teaching you to explore and understand data.
    - **Resources**:
        - Kaggle Datasets (search "Titanic" or "Iris").
        - Kaggle Notebooks (search "Titanic EDA" for inspiration).
        - Towards Data Science: EDA.
    - **Tips**:
        - Choose a dataset with 500–5000 rows for manageability.
        - Write questions in Markdown for clarity (e.g., `## EDA Questions`).
        - Save as `day13_eda_planning.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–1:00 PM: Explore dataset.
        - 2:00–4:00 PM: Plan EDA in notebook.
    - **Milestone**: Select dataset, explore structure, outline EDA plan, commit to GitHub.
- **Day 14: Execute First EDA Project**
    - **Tasks** (7 hours):
        - **Clean Data** (2 hours): Handle missing values (`fillna()` or `dropna()`), remove duplicates (`drop_duplicates()`), and address outliers (e.g., IQR method: `Q1 - 1.5*IQR`).
        - **Analyze and Visualize** (3 hours): Compute statistics (mean, median, correlations with `df.corr()`), create visualizations (histograms, boxplots, correlation heatmap) using Matplotlib/Seaborn.

        - **Summarize and Publish** (2 hours): Write 5–7 insights in Markdown (e.g., "Older passengers had lower survival rates"), upload notebook to GitHub with a README (include dataset description, questions, key findings).
    - **Why It Matters**: This project showcases your ability to analyze and visualize data, a core portfolio piece.
    - **Resources**:
        - Kaggle: Data Cleaning.
        - Seaborn Visualization.
        - GitHub Markdown Guide.
    - **Tips**:
        - Use comments to explain cleaning steps (e.g., `# Dropped rows with missing age`).
        - Include at least 3–4 visualizations with captions.
        - Save as `day14_eda_project.ipynb`, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Clean data.
        - 11:00 AM–2:00 PM: Analyze and visualize.
        - 2:00–4:00 PM: Summarize and publish.
    - **Milestone**: Complete and publish first EDA project with visualizations and insights, commit to GitHub.

Week 3: Advanced Python, Data Wrangling, and Second EDA Project (Days 15–21)

Goal: Enhance Python skills, master advanced data manipulation, and complete a second EDA project.

- **Day 15: Advanced Python Concepts**
    - **Tasks** (6 hours):
        - **List Comprehensions and Generators** (2 hours): Study list comprehensions (e.g., `[x**2 for x in range(10)]`) and generators (e.g., `(x**2 for x in range(10))`) for memory efficiency.
        - **Modules and Packages** (2 hours): Create/import custom modules (e.g., `my_utils.py` with a cleaning function), install/use external packages (e.g., `requests` via `pip install requests`).
        - **Practice** (2 hours): Write 5 list comprehensions and a module with 2 functions (e.g., data cleaning, stats calculation).

    - **Why It Matters**: Advanced Python improves code efficiency and prepares you for larger projects.
    - **Resources**:
        - Real Python: List Comprehensions.
        - Automate the Boring Stuff (Chapter 8).
        - GeeksforGeeks: Modules.
    - **Tips**:
        - Test generators with `next()` to understand flow.
        - Save module as `my_utils.py` and import in a notebook (`day15_advanced_python.ipynb`).
        - Commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Learn comprehensions and generators.
        - 11:00 AM–1:00 PM: Learn modules.
        - 2:00–4:00 PM: Practice.
    - **Milestone**: Write efficient Python code, create a reusable module, commit to GitHub.
- **Day 16: Data Cleaning Techniques**
    - **Tasks** (6 hours):
        - **Learn Cleaning** (3 hours): Study handling duplicates (`df.drop_duplicates()`), inconsistent data (e.g., standardizing text case with `str.lower()`), and outliers (e.g., IQR method).
        - **Practice** (3 hours): Clean a messy dataset (e.g., Kaggle's "Dirty Datasets" or Titanic). Fix nulls, duplicates, and outliers.
    - **Why It Matters**: Clean data is critical for accurate analysis and modeling.
    - **Resources**:
        - Kaggle: Data Cleaning Course.
        - Towards Data Science: Data Cleaning.
        - Pandas: Handling Missing Data.
    - **Tips**:
        - Always check `df.info()` and `df.isna().sum()` before cleaning.
        - Save cleaned data as `cleaned_dataset.csv`.
        - Save as `day16_data_cleaning.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn cleaning techniques.
        - 1:00–4:00 PM: Practice cleaning.
    - **Milestone**: Clean a dataset, handle common issues, commit to GitHub.
- **Day 17: Interactive Visualization with Plotly**

    - **Tasks** (6 hours):
        - **Learn Plotly** (3 hours): Install Plotly (`pip install plotly`). Create interactive plots (scatter, bar, heatmap, line) using `plotly.express` and `plotly.graph_objects`.
        - **Practice** (3 hours): Create 5 interactive visualizations on a dataset (e.g., Titanic or Iris).
    - **Why It Matters**: Interactive visualizations stand out in portfolios and dashboards, appealing to employers.
    - **Resources**:
        - Plotly Python Docs.
        - DataCamp Plotly Tutorial.
        - Kaggle: Plotly Notebooks.
    - **Tips**:
        - Export plots as HTML (`fig.write_html('plot.html')`) for sharing.
        - Compare Plotly with Seaborn for style differences.
        - Save as `day17_plotly.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn Plotly.
        - 1:00–4:00 PM: Practice visualizations.
    - **Milestone**: Create 5 interactive visualizations, export one as HTML, commit to GitHub.
- **Day 18: Introduction to SQL for Data Science**
    - **Tasks** (6 hours):
        - **Learn SQL Basics** (3 hours): Study `SELECT`, `WHERE`, `ORDER BY`, `LIMIT`, and filtering using an online SQL editor (e.g., SQLite via Mode Analytics or DB Fiddle).
        - **Practice** (3 hours): Solve 5 SQL problems (e.g., filter rows by condition, sort data) on LeetCode or Mode Analytics.
    - **Why It Matters**: SQL is essential for querying databases, a common requirement in data science roles.
    - **Resources**:
        - Mode Analytics SQL Tutorial.
        - LeetCode SQL Problems (select “Easy”).
        - W3Schools SQL.
    - **Tips**:
        - Use an online editor for quick practice (no local setup needed).
        - Save queries in a text file (`day18_sql.txt`) and commit to GitHub.
        - Practice explaining query logic aloud.
    - **Time Breakdown**:

        - 9:00–12:00 PM: Learn SQL basics.
        - 1:00–4:00 PM: Practice SQL problems.
    - **Milestone**: Write and execute basic SQL queries, commit to GitHub.
- **Day 19: Intermediate SQL**
    - **Tasks** (6 hours):
        - **Learn Joins and Aggregations** (3 hours): Study `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and aggregation functions (`COUNT`, `SUM`, `AVG`, `GROUP BY`).
        - **Practice** (3 hours): Solve 5 problems involving joins and aggregations on StrataScratch or LeetCode.
    - **Why It Matters**: Joins and aggregations are critical for complex data queries in real-world projects.
    - **Resources**:
        - W3Schools: SQL Joins.
        - StrataScratch SQL Questions.
        - Mode Analytics: Advanced SQL.
    - **Tips**:
        - Visualize joins as Venn diagrams to understand relationships.
        - Test queries on small datasets to avoid confusion.
        - Save as `day19_intermediate_sql.txt` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn joins and aggregations.
        - 1:00–4:00 PM: Practice problems.
    - **Milestone**: Master SQL joins and aggregations, solve 5 problems, commit to GitHub.
- **Day 20: Second EDA Project – Planning**
    - **Tasks** (6 hours):
        - **Select Dataset** (1 hour): Choose a different Kaggle dataset (e.g., Wine Quality, Heart Disease) with numerical and categorical columns.
        - **Explore and Plan** (3 hours): Load dataset, use `df.info()`, `df.describe()`, and list 5–10 questions (e.g., “What factors predict heart disease?”).
        - **Setup Notebook** (2 hours): Create a Jupyter Notebook with sections (Introduction, Data Loading, Cleaning, Analysis, Conclusion).
    - **Why It Matters**: A second EDA project reinforces skills and diversifies your portfolio.
    - **Resources**:
        - Kaggle Datasets (search “Wine Quality” or “Heart Disease”).
        - Kaggle Notebooks (search “EDA” for inspiration).
        - Towards Data Science: EDA Planning.

    - **Tips**:
        - Choose a dataset with clear documentation and 500–5000 rows.
        - Write questions in Markdown for clarity.
        - Save as `day20_eda2_planning.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–1:00 PM: Explore dataset.
        - 2:00–4:00 PM: Plan EDA in notebook.
    - **Milestone**: Select and explore a new dataset, outline EDA plan, commit to GitHub.
- **Day 21: Second EDA Project – Execution**
    - **Tasks** (7 hours):
        - **Clean and Analyze** (4 hours): Handle missing values, outliers, and inconsistent data. Compute statistics (mean, median, correlations). Create visualizations (boxplots, pairplots, heatmap) using Seaborn/Plotly.
        - **Summarize** (2 hours): Write 5–7 insights in Markdown (e.g., "Age strongly correlates with heart disease").
        - **Publish** (1 hour): Upload notebook to GitHub with a README (include dataset description, questions, key findings).
    - **Why It Matters**: This project demonstrates your ability to handle diverse datasets, strengthening your portfolio.
    - **Resources**:
        - Kaggle: Data Cleaning.
        - Seaborn Visualization.
        - GitHub Markdown Guide.
    - **Tips**:
        - Use comments to explain cleaning steps (e.g., `# Removed outliers using IQR`).
        - Include at least 3–4 visualizations with captions.
        - Save as `day21_eda2_execution.ipynb`, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–1:00 PM: Clean and analyze.
        - 1:00–3:00 PM: Summarize insights.
        - 3:00–4:00 PM: Publish to GitHub.
    - **Milestone**: Complete and publish second EDA project with insights and visualizations, commit to GitHub.

Week 4: Mathematics for ML and Additional Tools (Days 22–30)

Goal: Learn essential mathematics for ML, master tools like Git, and prepare for machine learning.

- **Day 22: Linear Algebra Basics**
  - **Tasks** (6 hours):
    - **Learn Concepts** (3 hours): Study vectors (e.g., `[1, 2, 3]`), matrices (2D arrays), dot product, matrix multiplication, and inverse. Understand their role in ML (e.g., feature matrices).
    - **Practice** (3 hours): Use NumPy to perform 5 operations (e.g., matrix inverse with `np.linalg.inv()`, eigenvalue with `np.linalg.eig()`).
  - **Why It Matters**: Linear algebra is the backbone of ML algorithms like regression and neural networks.
  - **Resources**:
    - 3Blue1Brown: Essence of Linear Algebra.
    - Khan Academy: Linear Algebra.
    - NumPy Linear Algebra.
  - **Tips**:
    - Visualize matrices as tables (rows = samples, columns = features).
    - Test operations in Jupyter for clarity.
    - Save as `day22_linear_algebra.ipynb` and commit to GitHub.
  - **Time Breakdown**:
    - 9:00–12:00 PM: Learn linear algebra.
    - 1:00–4:00 PM: Practice with NumPy.
  - **Milestone**: Understand vectors/matrices, perform 5 NumPy operations, commit to GitHub.
- **Day 23: Calculus Basics**
  - **Tasks** (6 hours):
    - **Learn Concepts** (3 hours): Study derivatives (rate of change, e.g., slope of x^2) and integrals (area under curve). Focus on their role in gradient descent for ML optimization.
    - **Practice** (3 hours): Solve 5 simple problems (e.g., derivative of x^3, integral of $2x$) manually or with SymPy (`pip install sympy`).
  - **Why It Matters**: Calculus drives optimization in ML models (e.g., minimizing loss functions).
  - **Resources**:
    - Khan Academy: Calculus.
    - StatQuest: Gradient Descent.
    - 3Blue1Brown: Calculus.

    - **Tips**:
        - Focus on intuition (e.g., derivative as slope).
        - Relate to ML (e.g., gradient descent updates weights).
        - Save as `day23_calculus.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn calculus.
        - 1:00–4:00 PM: Practice problems.
    - **Milestone**: Understand derivatives/integrals, solve 5 problems, commit to GitHub.
- **Day 24: Probability Distributions**
    - **Tasks** (6 hours):
        - **Learn Distributions** (3 hours): Study normal (bell curve), binomial (discrete events), and Poisson (event frequency) distributions. Understand parameters (mean, variance).
        - **Practice** (3 hours): Simulate distributions with SciPy (e.g., `stats.norm.rvs()`), plot with Seaborn.
    - **Why It Matters**: Distributions model uncertainty in ML (e.g., predicting probabilities).
    - **Resources**:
        - SciPy Stats Docs.
        - StatQuest: Distributions.
        - Towards Data Science: Probability.
    - **Tips**:
        - Use `seaborn.histplot()` for visualization.
        - Note parameters (e.g., mean, std dev for normal).
        - Save as `day24_distributions.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn distributions.
        - 1:00–4:00 PM: Simulate and visualize.
    - **Milestone**: Simulate and visualize 3 distributions, commit to GitHub.
- **Day 25: Git and GitHub**
    - **Tasks** (6 hours):
        - **Learn Git** (3 hours): Install Git, learn commands (`git init`, `git add`, `git commit`, `git push`, `git status`). Create a GitHub repository for your projects.
        - **Practice** (3 hours): Commit Day 14 and Day 21 EDA projects to GitHub, write clear commit messages (e.g., “Add Titanic EDA notebook”).
    - **Why It Matters**: Git enables version control, and GitHub showcases projects to recruiters.

    - **Resources**:
        - GitHub Guides.
        - FreeCodeCamp: Git Tutorial.
        - Atlassian Git Tutorial.
    - **Tips**:
        - Write commit messages in present tense (e.g., “Add feature”).
        - Create a clean GitHub profile with a README (add bio, project links).
        - Save Git commands in `day25_git.txt` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn Git.
        - 1:00–4:00 PM: Practice committing.
    - **Milestone**: Set up Git, push projects to GitHub, create profile README.
- **Day 26: Jupyter Notebook Best Practices**
    - **Tasks** (6 hours):
        - **Learn Best Practices** (2 hours): Study writing clean notebooks with Markdown headings, comments, and modular code (e.g., functions for repetitive tasks). Avoid cluttered cells.
        - **Practice** (4 hours): Refactor Day 14 and Day 21 EDA notebooks with clear sections, comments, and Markdown explanations.
    - **Why It Matters**: Professional notebooks impress recruiters and collaborators.
    - **Resources**:
        - Jupyter Notebook Best Practices.
        - Kaggle Notebooks (review top-rated EDA notebooks).
        - Real Python: Jupyter.
    - **Tips**:
        - Use Markdown for headings (e.g., `# Data Cleaning`).
        - Test notebooks by restarting and running all cells (`Kernel > Restart & Run All`).
        - Save as `day26_jupyter_refactor.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Learn best practices.
        - 11:00 AM–3:00 PM: Refactor notebooks.
    - **Milestone**: Refactor two EDA notebooks, ensure they are clean and reproducible, commit to GitHub.
- **Day 27: Introduction to Scikit-learn**
    - **Tasks** (6 hours):
        - **Learn Scikit-learn** (3 hours): Install scikit-learn (`pip install scikit-learn`). Study ML workflow: `fit()`, `predict()`, `score()`. Explore a simple model (e.g., K-Nearest Neighbors, `KNeighborsClassifier()`).

        - **Practice** (3 hours): Load Iris dataset (`sklearn.datasets.load_iris`), train a KNN model, and evaluate accuracy.
    - **Why It Matters**: Scikit-learn is the primary ML library in Python, used in most data science roles.
    - **Resources**:
        - Scikit-learn Getting Started.
        - Kaggle: Intro to Machine Learning.
        - StatQuest: Scikit-learn.
    - **Tips**:
        - Start with small datasets like Iris (available via `sklearn.datasets`).
        - Save model code in `day27_scikit_learn.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn Scikit-learn.
        - 1:00–4:00 PM: Practice with KNN.
    - **Milestone**: Train and evaluate a simple ML model, commit to GitHub.
- **Day 28: Data Preprocessing for Machine Learning**
    - **Tasks** (6 hours):
        - **Learn Preprocessing** (3 hours): Study scaling numerical features (`StandardScaler`), encoding categorical variables (`OneHotEncoder`, `LabelEncoder`), and handling imbalanced data (e.g., oversampling with `imblearn`, `pip install imblearn`).
        - **Practice** (3 hours): Preprocess a dataset (e.g., Titanic) using `ColumnTransformer` to combine scaling and encoding.
    - **Why It Matters**: Proper preprocessing improves ML model accuracy and prevents errors.
    - **Resources**:
        - Scikit-learn Preprocessing.
        - Kaggle: Feature Engineering.
        - Towards Data Science: Preprocessing.
    - **Tips**:
        - Create a preprocessing pipeline with `Pipeline` for automation.
        - Test on a small subset of data to verify transformations.
        - Save as `day28_preprocessing.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn preprocessing.
        - 1:00–4:00 PM: Practice preprocessing.
    - **Milestone**: Preprocess a dataset with scaling and encoding, commit to GitHub.
- **Day 29: Mini Project – Data Cleaning and Visualization**

    - **Tasks** (7 hours):
        - **Select Dataset** (1 hour): Choose a dataset with missing values and outliers (e.g., Kaggle's "Credit Card Fraud" or "Adult Income").
        - **Clean and Visualize** (4 hours): Clean data (handle nulls, outliers, duplicates), compute statistics, create 3–4 visualizations (e.g., boxplot for outliers, heatmap for correlations) using Seaborn/Plotly.
        - **Document and Publish** (2 hours): Write a Jupyter Notebook with insights, upload to GitHub with a README (include dataset, cleaning steps, visualizations).
    - **Why It Matters**: Reinforces cleaning and visualization, adding a third portfolio project.
    - **Resources**:
        - Kaggle Datasets (search "Credit Card Fraud").
        - Seaborn Visualization.
        - GitHub Markdown Guide.
    - **Tips**:
        - Explain cleaning decisions in Markdown (e.g., "Dropped 5% rows with missing income").
        - Save visualizations as PNGs for README.
        - Save as day29_cleaning_visualization.ipynb, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–2:00 PM: Clean and visualize.
        - 2:00–4:00 PM: Document and publish.
    - **Milestone**: Complete and publish a cleaning and visualization project, commit to GitHub.
- **Day 30: Review and Consolidation**
    - **Tasks** (6 hours):
        - **Review Month 1** (3 hours): Revisit Python, Pandas, NumPy, statistics, probability, and visualization concepts (Days 1–29). Skim notebooks and notes.
        - **Practice** (3 hours): Solve 5 mixed problems (e.g., clean a dataset, create a plot, write a function) from HackerRank or Kaggle.
    - **Why It Matters**: Consolidation ensures retention of foundational skills before moving to ML.
    - **Resources**:
        - HackerRank Python Challenges.

        - Kaggle Python Course.
        - DataCamp Cheat Sheets.
    - **Tips**:
        - Create a cheat sheet of key functions (e.g., Pandas `groupby()`, NumPy `array()`) in a notebook.
        - Update GitHub with revised notebooks if needed.
        - Save as `day30_review.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review concepts.
        - 1:00–4:00 PM: Practice problems.
    - **Milestone**: Solidify Month 1 skills, create cheat sheet, commit to GitHub.

Month 2: Core Machine Learning, Projects, and Evaluation (Days 31–60)

Focus: Learn core ML algorithms, evaluation metrics, and complete two mini-projects (housing price prediction and spam detection) to strengthen your portfolio.

Week 5: Machine Learning Basics and Regression (Days 31–37)

Goal: Understand ML fundamentals and master regression techniques.

- **Day 31: Machine Learning Introduction**
    - **Tasks** (6 hours):
        - **Learn Concepts** (3 hours): Study supervised vs. unsupervised learning, train-test split (e.g., `train_test_split()`), overfitting, and underfitting. Understand ML workflow: data > preprocess > train > evaluate.
        - **Practice** (3 hours): Split a dataset (e.g., Iris) using `train_test_split()`, explore train/test sets with `df.shape`.
    - **Why It Matters**: These concepts form the foundation of all ML projects.
    - **Resources**:
        - Scikit-learn User Guide.
        - Coursera: Machine Learning by Andrew Ng (Week 1, free audit).
        - StatQuest: ML Basics.
    - **Tips**:
        - Understand why train-test split prevents overfitting (e.g., 80-20 split).
        - Use `random_state=42` for reproducibility.

        - Save as day31_ml_basics.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn ML concepts.
        - 1:00–4:00 PM: Practice splitting dataset.
    - **Milestone**: Understand ML workflow, split a dataset, commit to GitHub.
- **Day 32: Linear Regression**
    - **Tasks** (6 hours):
        - **Learn Linear Regression** (3 hours): Study predicting continuous variables (e.g., house prices) using a linear equation ($y = mx + b$). Explore scikit-learn implementation (LinearRegression()).
        - **Practice** (3 hours): Implement linear regression on a dataset (e.g., Boston Housing from sklearn.datasets). Visualize predictions vs. actual values with a scatter plot.
    - **Why It Matters**: Linear regression is a fundamental supervised learning algorithm.
    - **Resources**:
        - StatQuest: Linear Regression.
        - Scikit-learn: Linear Regression.
        - Kaggle: Regression.
    - **Tips**:
        - Standardize features with StandardScaler before training.
        - Plot residuals (actual - predicted) to check model fit.
        - Save as day32_linear_regression.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn linear regression.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Train and visualize a linear regression model, commit to GitHub.
- **Day 33: Regression Metrics**
    - **Tasks** (6 hours):
        - **Learn Metrics** (3 hours): Study Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score. Understand their formulas and use cases (e.g., RMSE penalizes large errors).
        - **Practice** (3 hours): Compute metrics for a regression model using scikit-learn (mean_squared_error, r2_score) on the Boston Housing dataset.
    - **Why It Matters**: Metrics quantify model performance, critical for evaluation.
    - **Resources**:
        - Real Python: Regression Metrics.

        - Scikit-learn Metrics.
        - Towards Data Science: Metrics.
    - **Tips**:
        - MAE for absolute errors, RMSE for penalizing large errors.
        - Save metrics in a table (e.g., `pd.DataFrame({'Metric': ['MAE', 'RMSE'], 'Value': [...]})`).
        - Save as `day33_regression_metrics.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn metrics.
        - 1:00–4:00 PM: Practice computing metrics.
    - **Milestone**: Compute and interpret regression metrics, commit to GitHub.
- **Day 34: Polynomial and Regularized Regression**
    - **Tasks** (6 hours):
        - **Learn Polynomial Regression** (2 hours): Study `PolynomialFeatures` to model non-linear relationships (e.g., $y = ax^2 + bx + c$).
        - **Learn Regularization** (2 hours): Study Lasso (L1) and Ridge (L2) regression to prevent overfitting by penalizing large coefficients.
        - **Practice** (2 hours): Implement polynomial regression and Lasso/Ridge on a dataset (e.g., Boston Housing), compare results.
    - **Why It Matters**: These techniques handle complex data and improve model generalization.
    - **Resources**:
        - Scikit-learn: Polynomial Regression.
        - StatQuest: Regularization.
        - Real Python: Polynomial Regression.
    - **Tips**:
        - Use `Pipeline` to combine `PolynomialFeatures` and `LinearRegression`.
        - Experiment with polynomial degrees (2, 3) and regularization parameters (e.g., `alpha=1.0`).
        - Save as `day34_advanced_regression.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Learn polynomial regression.
        - 11:00 AM–1:00 PM: Learn regularization.
        - 2:00–4:00 PM: Practice implementation.
    - **Milestone**: Implement and compare polynomial and regularized regression, commit to GitHub.
- **Day 35: Mini Project – Housing Price Prediction**
    - **Tasks** (7 hours):

        - **Setup** (1 hour): Select Kaggle's House Prices dataset (or Boston Housing for simplicity).
        - **Preprocess and Model** (4 hours): Clean data (handle missing values, encode categoricals with `OneHotEncoder`), train a linear regression model, apply preprocessing pipeline.
        - **Evaluate** (2 hours): Compute MAE, RMSE, R^2; create visualizations (predicted vs. actual, residuals) using Matplotlib/Seaborn.
    - **Why It Matters**: This project showcases regression skills, a common task in data science roles.
    - **Resources**:
        - Kaggle: House Prices.
        - Towards Data Science: Regression Projects.
        - Scikit-learn Pipeline.
    - **Tips**:
        - Save preprocessing pipeline for reuse (`joblib.dump(pipeline, 'pipeline.pkl')`).
        - Document assumptions (e.g., "Filled missing prices with median").
        - Save as `day35_housing_project.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–2:00 PM: Preprocess and model.
        - 2:00–4:00 PM: Evaluate and visualize.
    - **Milestone**: Complete a regression project with preprocessing and evaluation, commit to GitHub.
- **Day 36: Enhance Housing Project**
    - **Tasks** (6 hours):
        - **Visualize** (3 hours): Add visualizations (scatter of predictions, residual plot, feature importance using coefficients) using Seaborn/Plotly.
        - **Document** (3 hours): Write detailed Markdown sections (Problem, Approach, Results, Limitations) and code comments.
    - **Why It Matters**: Professional documentation makes your project stand out to recruiters.
    - **Resources**:
        - Seaborn Visualization.
        - Jupyter Best Practices.
        - Kaggle Notebooks.
    - **Tips**:
        - Use Markdown headings for structure (e.g., `## Data Preprocessing`).
        - Save visualizations as PNGs for README.

        - Save as day36_housing_enhanced.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Add visualizations.
        - 1:00–4:00 PM: Document project.
    - **Milestone**: Enhance project with visuals and documentation, commit to GitHub.
- **Day 37: Finalize Housing Project**
    - **Tasks** (6 hours):
        - **Clean Up** (2 hours): Remove unused code, ensure notebook is organized and reproducible (Kernel > Restart & Run All).
        - **Publish** (3 hours): Create a GitHub repository, write a detailed README (project overview, dataset, findings, visuals), and upload notebook.
        - **Share** (1 hour): Share the GitHub link on LinkedIn with a brief post (e.g., "Built a housing price prediction model using linear regression!").
    - **Why It Matters**: A polished project strengthens your portfolio and demonstrates professionalism.
    - **Resources**:
        - GitHub README Templates.
        - Kaggle: House Prices Notebooks.
    - **Tips**:
        - Test notebook for errors before publishing.
        - Include a screenshot of a visualization in the README.
        - Save as day37_housing_final.ipynb, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Clean up notebook.
        - 11:00 AM–2:00 PM: Publish to GitHub.
        - 2:00–3:00 PM: Share on LinkedIn.
    - **Milestone**: Publish a professional housing price prediction project, share on LinkedIn, commit to GitHub.

Week 6: Classification Techniques and NLP Project (Days 38–44)

Goal: Master classification algorithms and complete a text classification project.

- **Day 38: Logistic Regression**
    - **Tasks** (6 hours):
        - **Learn Logistic Regression** (3 hours): Study binary classification, sigmoid function (maps to probabilities), and scikit-learn implementation (LogisticRegression()).

        - **Practice** (3 hours): Implement on Iris dataset (binary classification, e.g., Setosa vs. non-Setosa), visualize decision boundaries with Matplotlib.
    - **Why It Matters**: Logistic regression is a core classification algorithm, widely used in industry.
    - **Resources**:
        - StatQuest: Logistic Regression.
        - Scikit-learn: Logistic Regression.
        - Real Python: Logistic Regression.
    - **Tips**:
        - Understand log-odds and sigmoid curve.
        - Use `predict_proba()` to explore probabilities.
        - Save as `day38_logistic_regression.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn logistic regression.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Train and visualize a logistic regression model, commit to GitHub.
- **Day 39: K-Nearest Neighbors (KNN)**
    - **Tasks** (6 hours):
        - **Learn KNN** (3 hours): Study distance metrics (Euclidean), parameter tuning (K), and scikit-learn implementation (`KNeighborsClassifier()`).
        - **Practice** (3 hours): Implement KNN on a dataset (e.g., Iris), experiment with K values (3, 5, 7), plot accuracy vs. K.
    - **Why It Matters**: KNN is a simple, effective classifier for small datasets.
    - **Resources**:
        - Real Python: KNN.
        - Scikit-learn: KNN.
        - Kaggle: Classification.
    - **Tips**:
        - Scale features with `StandardScaler` (KNN is distance-based).
        - Plot accuracy curve with `matplotlib`.
        - Save as `day39_knn.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn KNN.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Implement KNN, tune K parameter, commit to GitHub.
- **Day 40: Decision Trees**
    - **Tasks** (6 hours):

        - **Learn Decision Trees** (3 hours): Study entropy, Gini index, tree structure, and scikit-learn implementation (`DecisionTreeClassifier()`). Visualize trees with `plot_tree()`.
        - **Practice** (3 hours): Implement on a dataset (e.g., Titanic), visualize the tree, and interpret splits.
    - **Why It Matters**: Decision trees are interpretable and used in many ML applications.
    - **Resources**:
        - StatQuest: Decision Trees.
        - Scikit-learn: Decision Trees.
        - Towards Data Science: Decision Trees.
    - **Tips**:
        - Limit tree depth (`max_depth=3`) to avoid overfitting.
        - Save tree visualization as PNG.
        - Save as `day40_decision_trees.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn decision trees.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Train and visualize a decision tree, commit to GitHub.
- **Day 41: Random Forests**
    - **Tasks** (6 hours):
        - **Learn Random Forests** (3 hours): Study ensemble learning, bagging, and feature importance. Explore `RandomForestClassifier()` in scikit-learn.
        - **Practice** (3 hours): Implement on a dataset (e.g., Titanic), compute feature importance, visualize with a bar plot.
    - **Why It Matters**: Random forests improve accuracy and robustness over single trees.
    - **Resources**:
        - StatQuest: Random Forests.
        - Scikit-learn: Random Forests.
        - Real Python: Random Forests.
    - **Tips**:
        - Start with `n_estimators=100` and `random_state=42`.
        - Plot feature importance with `plt.bar()`.
        - Save as `day41_random_forests.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn random forests.
        - 1:00–4:00 PM: Practice implementation.

    - **Milestone**: Train a random forest, interpret feature importance, commit to GitHub.
- **Day 42: Classification Metrics**
    - **Tasks** (6 hours):
        - **Learn Metrics** (3 hours): Study accuracy, precision, recall, F1-score, and confusion matrix. Understand use cases (e.g., F1 for imbalanced data).
        - **Practice** (3 hours): Compute metrics for a classification model (e.g., logistic regression on Iris), visualize confusion matrix with `seaborn.heatmap()`.
    - **Why It Matters**: Metrics evaluate classifier performance, especially for imbalanced datasets.
    - **Resources**:
        - Real Python: Classification Metrics.
        - Scikit-learn Metrics.
        - Towards Data Science: Metrics.
    - **Tips**:
        - Focus on F1-score for imbalanced data (e.g., fraud detection).
        - Save confusion matrix as PNG.
        - Save as `day42_classification_metrics.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn metrics.
        - 1:00–4:00 PM: Practice computing metrics.
    - **Milestone**: Compute and visualize classification metrics, commit to GitHub.
- **Day 43: Mini Project – Spam/Ham Detection**
    - **Tasks** (7 hours):
        - **Setup** (1 hour): Select Kaggle's SMS Spam Collection dataset.
        - **Preprocess and Model** (4 hours): Use `TfidfVectorizer` for text preprocessing, train a logistic regression model, apply preprocessing pipeline.
        - **Evaluate** (2 hours): Compute accuracy, precision, recall, F1; visualize confusion matrix and word cloud (use `wordcloud`, `pip install wordcloud`).
    - **Why It Matters**: Introduces NLP and classification, a common data science task.
    - **Resources**:
        - Kaggle: SMS Spam Collection.
        - Towards Data Science: NLP Basics.
        - Scikit-learn: Text Processing.
    - **Tips**:
        - Use `stop_words='english'` in `TfidfVectorizer` to remove common words.
        - Test model with custom text inputs (e.g., "Win a free iPhone!").
        - Save as `day43_spam_project.ipynb` and commit to GitHub.

    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–2:00 PM: Preprocess and model.
        - 2:00–4:00 PM: Evaluate and visualize.
    - **Milestone**: Complete a text classification project with NLP preprocessing, commit to GitHub.
- **Day 44: Finalize Spam/Ham Project**
    - **Tasks** (6 hours):
        - **Enhance** (3 hours): Add visualizations (confusion matrix, word cloud), refine preprocessing pipeline (e.g., test different `TfidfVectorizer` parameters).
        - **Publish** (3 hours): Document process in notebook (Problem, Approach, Results, Limitations), clean code, upload to GitHub with a detailed README.
    - **Why It Matters**: A polished NLP project showcases your ability to handle text data.
    - **Resources**:
        - WordCloud Python Library.
        - GitHub README Templates.
        - Kaggle: NLP Notebooks.
    - **Tips**:
        - Include a section on model limitations (e.g., “Struggles with ambiguous text”).
        - Share GitHub link on LinkedIn with a post (e.g., “Built a spam classifier using NLP!”).
        - Save as `day44_spam_final.ipynb`, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Enhance project.
        - 1:00–4:00 PM: Publish and share.
    - **Milestone**: Publish a professional text classification project, share on LinkedIn, commit to GitHub.

Week 7: Unsupervised Learning and Portfolio Building (Days 45–51)

Goal: Learn unsupervised learning techniques and start building a professional portfolio.

- **Day 45: K-Means Clustering**
    - **Tasks** (6 hours):
        - **Learn K-Means** (3 hours): Study the algorithm (grouping data into K clusters), elbow method, and silhouette score for choosing K.

        - **Practice** (3 hours): Implement K-Means on Iris dataset (sklearn.cluster.KMeans), visualize clusters in 2D with PCA, plot elbow curve.
    - **Why It Matters**: Clustering is used for pattern discovery in unlabeled data (e.g., customer segmentation).
    - **Resources**:
        - StatQuest: K-Means.
        - Scikit-learn: Clustering.
        - Real Python: K-Means.
    - **Tips**:
        - Scale features before clustering (StandardScaler).
        - Try K=2 to 10 for elbow method.
        - Save as day45_kmeans.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn K-Means.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Implement and visualize K-Means clustering, commit to GitHub.
- **Day 46: Hierarchical Clustering**
    - **Tasks** (6 hours):
        - **Learn Hierarchical Clustering** (3 hours): Study agglomerative clustering, linkage methods (e.g., ward), and dendrograms.
        - **Practice** (3 hours): Implement using sklearn.cluster.AgglomerativeClustering, visualize dendrogram with scipy.cluster.hierarchy.dendrogram.
    - **Why It Matters**: Hierarchical clustering provides a tree-based view of data relationships.
    - **Resources**:
        - Real Python: Hierarchical Clustering.
        - Scikit-learn: Hierarchical.
        - SciPy: Dendrogram.
    - **Tips**:
        - Compare with K-Means results.
        - Save dendrogram as PNG.
        - Save as day46_hierarchical.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn hierarchical clustering.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Implement and visualize hierarchical clustering, commit to GitHub.
- **Day 47: Principal Component Analysis (PCA)**
    - **Tasks** (6 hours):

        - **Learn PCA** (3 hours): Study dimensionality reduction, explained variance ratio, and scikit-learn implementation (`PCA()`).
        - **Practice** (3 hours): Apply PCA to a high-dimensional dataset (e.g., Iris), plot explained variance, visualize first two components with `seaborn.scatterplot()`.
    - **Why It Matters**: PCA simplifies high-dimensional data for visualization and modeling.
    - **Resources**:
        - StatQuest: PCA.
        - Scikit-learn: PCA.
        - Towards Data Science: PCA.
    - **Tips**:
        - Standardize data before PCA (`StandardScaler`).
        - Plot components with `scatterplot`.
        - Save as `day47_pca.ipynb` and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn PCA.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Apply PCA, visualize reduced dimensions, commit to GitHub.

- **Day 48: Project – Customer Segmentation**
    - **Tasks** (7 hours):
        - **Setup** (1 hour): Select Kaggle's Mall Customers dataset (or similar, e.g., Wholesale Customers). Download and explore the dataset (df.info(), df.head()) to understand features like age, income, and spending score.
        - **Model and Visualize** (4 hours): Preprocess data (handle missing values, scale with StandardScaler). Apply K-Means clustering (sklearn.cluster.KMeans) with 3–10 clusters, use the elbow method and silhouette score (sklearn.metrics.silhouette_score) to select optimal K. Apply PCA (sklearn.decomposition.PCA) to reduce dimensions to 2D for visualization. Create scatter plots of clusters using seaborn.scatterplot().
        - **Document** (2 hours): Write a Jupyter Notebook with sections (Introduction, Data Preprocessing, Clustering, Visualization, Insights). Summarize findings (e.g., "Cluster 1: High-income, low-spending customers"). Save visualizations as PNGs.
    - **Why It Matters**: Customer segmentation is a common unsupervised learning task in industries like retail and e-commerce, showcasing your ability to derive insights from unlabeled data.

- **Resources**:
  - Kaggle: Mall Customers Dataset.
  - Scikit-learn: K-Means.
  - Towards Data Science: Customer Segmentation.
- **Tips**:
  - Use n_clusters=range(3, 11) for elbow plot to identify optimal K.
  - Add comments explaining preprocessing (e.g., # Scaled features to ensure equal weighting).
  - Save as day48_customer_segmentation.ipynb, create a GitHub repository, and commit with a descriptive message (e.g., "Add customer segmentation project").
- **Time Breakdown**:
  - 9:00–10:00 AM: Select and explore dataset.
  - 10:00 AM–2:00 PM: Preprocess, model, and visualize.
  - 2:00–4:00 PM: Document and save visualizations.
- **Milestone**: Complete a customer segmentation project with K-Means and PCA, commit to GitHub.

- **Day 49: Finalize Customer Segmentation Project**
  - **Tasks** (6 hours):
    - **Enhance Visualizations** (2 hours): Add a 3D scatter plot using plotly.express.scatter_3d() for better cluster visualization. Include an elbow plot and silhouette score plot using Matplotlib.
    - **Polish and Publish** (3 hours): Clean the notebook (remove unused code, ensure reproducibility with Kernel > Restart & Run All). Write a detailed README for the GitHub repository (include dataset description, methodology, key findings, and a visualization screenshot).
    - **Share** (1 hour): Post the project link on LinkedIn with a brief description (e.g., "Built a customer segmentation model using K-Means to identify distinct customer groups!").
  - **Why It Matters**: A polished project with interactive visualizations strengthens your portfolio and demonstrates professionalism to recruiters.
  - **Resources**:
    - Plotly Python Docs.
    - GitHub README Templates.
    - Kaggle: Clustering Notebooks.
  - **Tips**:
    - Test the notebook for errors before publishing.

        - Include a section on limitations (e.g., “K-Means assumes spherical clusters”).
        - Save as day49_customer_segmentation_final.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Enhance visualizations.
        - 11:00 AM–2:00 PM: Polish and publish.
        - 2:00–3:00 PM: Share on LinkedIn.
    - **Milestone**: Publish a professional customer segmentation project, share on LinkedIn, commit to GitHub.
- **Day 50: Portfolio Setup**
    - **Tasks** (6 hours):
        - **Create Portfolio Structure** (2 hours): Set up a GitHub repository named “DataSciencePortfolio” with subfolders for each project (e.g., eda_titanic, housing_prediction, spam_detection, customer_segmentation). Write a main README summarizing all projects with links.
        - **Optimize Projects** (3 hours): Review Day 14, 21, 29, 37, 44, and 49 projects. Ensure each has a clear README, clean code, and visualizations. Update commit messages for clarity.
        - **Share Portfolio** (1 hour): Add portfolio link to LinkedIn profile (About section) and share a post (e.g., “Excited to share my Data Science portfolio with 4 projects!”).
    - **Why It Matters**: A centralized portfolio showcases your work to recruiters and hiring managers, critical for job applications.
    - **Resources**:
        - GitHub Portfolio Examples.
        - Towards Data Science: Portfolio Building.
        - Markdown Guide.
    - **Tips**:
        - Use a consistent README template for each project (Problem, Approach, Results, Visuals).
        - Pin the portfolio repository on your GitHub profile for visibility.
        - Save portfolio README as portfolio_readme.md in the main repository and commit.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Create portfolio structure.
        - 11:00 AM–2:00 PM: Optimize projects.
        - 2:00–3:00 PM: Share portfolio.

    - **Milestone**: Create and share a professional GitHub portfolio with 4 projects, commit to GitHub.
- **Day 51: Introduction to Model Selection**
    - **Tasks** (6 hours):
        - **Learn Model Selection** (3 hours): Study cross-validation (cross_val_score), train-test split, and hyperparameter tuning (e.g., GridSearchCV in scikit-learn).
        - **Practice** (3 hours): Apply 5-fold cross-validation to a logistic regression model on the Iris dataset. Use GridSearchCV to tune hyperparameters (e.g., C for logistic regression).
    - **Why It Matters**: Model selection ensures robust, generalizable models, a key skill for ML engineers.
    - **Resources**:
        - Scikit-learn: Cross-Validation.
        - Scikit-learn: GridSearchCV.
        - StatQuest: Cross-Validation.
    - **Tips**:
        - Use cv=5 for cross-validation to balance speed and accuracy.
        - Save best parameters from GridSearchCV (e.g., grid.best_params_).
        - Save as day51_model_selection.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn model selection.
        - 1:00–4:00 PM: Practice cross-validation and tuning.
    - **Milestone**: Implement cross-validation and hyperparameter tuning, commit to GitHub.

Week 8: Advanced Machine Learning and Ensemble Methods (Days 52–58)

Goal: Learn advanced ML algorithms (SVM, Gradient Boosting) and apply them to a project.

- **Day 52: Support Vector Machines (SVM)**
    - **Tasks** (6 hours):
        - **Learn SVM** (3 hours): Study linear and non-linear SVM (SVC in scikit-learn), kernels (linear, RBF), and the margin concept. Understand regularization parameter C.
        - **Practice** (3 hours): Implement SVM on a dataset (e.g., Iris or Breast Cancer from sklearn.datasets). Visualize decision boundaries for a 2D dataset using Matplotlib.

    - **Why It Matters**: SVM is effective for small to medium datasets and is used in classification tasks like fraud detection.
    - **Resources**:
        - StatQuest: SVM.
        - Scikit-learn: SVM.
        - Real Python: SVM.
    - **Tips**:
        - Scale features with StandardScaler (SVM is sensitive to scale).
        - Experiment with kernel='linear' and kernel='rbf'.
        - Save as day52_svm.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn SVM.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Train and visualize an SVM model, commit to GitHub.
- **Day 53: Gradient Boosting (XGBoost)**
    - **Tasks** (6 hours):
        - **Learn XGBoost** (3 hours): Study boosting concepts, gradient boosting, and XGBoost (pip install xgboost). Explore key parameters (learning_rate, n_estimators).
        - **Practice** (3 hours): Implement XGBoost (XGBClassifier) on a dataset (e.g., Titanic). Compute feature importance and visualize with a bar plot.
    - **Why It Matters**: XGBoost is a powerful algorithm used in many data science competitions and industry applications.
    - **Resources**:
        - XGBoost Documentation.
        - Towards Data Science: XGBoost.
        - StatQuest: Gradient Boosting.
    - **Tips**:
        - Start with default parameters, then tune learning_rate (e.g., 0.1, 0.01).
        - Save feature importance plot as PNG.
        - Save as day53_xgboost.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn XGBoost.
        - 1:00–4:00 PM: Practice implementation.
    - **Milestone**: Train an XGBoost model, visualize feature importance, commit to GitHub.
- **Day 54: Hyperparameter Tuning**
    - **Tasks** (6 hours):

        - **Learn Tuning** (3 hours): Study GridSearchCV and RandomizedSearchCV for tuning hyperparameters (e.g., C for SVM, max_depth for XGBoost).
        - **Practice** (3 hours): Tune an XGBoost model on the Titanic dataset using GridSearchCV (try n_estimators=[50, 100], max_depth=[3, 5]).
    - **Why It Matters**: Tuning optimizes model performance, a critical skill for ML engineers.
    - **Resources**:
        - Scikit-learn: GridSearchCV.
        - Towards Data Science: Hyperparameter Tuning.
        - Kaggle: Hyperparameter Tuning.
    - **Tips**:
        - Use RandomizedSearchCV for faster tuning on large datasets.
        - Save best model with joblib.dump(grid.best_estimator_, 'model.pkl').
        - Save as day54_tuning.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn tuning techniques.
        - 1:00–4:00 PM: Practice tuning XGBoost.
    - **Milestone**: Tune an XGBoost model, save best model, commit to GitHub.
- **Day 55: Ensemble Methods**
    - **Tasks** (6 hours):
        - **Learn Ensembles** (3 hours): Study voting classifiers (VotingClassifier), stacking (StackingClassifier), and bagging. Understand how they combine multiple models.
        - **Practice** (3 hours): Build a voting classifier combining logistic regression, random forest, and SVM on the Titanic dataset.
    - **Why It Matters**: Ensembles improve performance by leveraging multiple models, common in industry.
    - **Resources**:
        - Scikit-learn: Ensemble Methods.
        - Towards Data Science: Ensemble Learning.
        - StatQuest: Ensemble Methods.
    - **Tips**:
        - Use voting='soft' for probability-based voting.
        - Compare ensemble accuracy with individual models.
        - Save as day55_ensembles.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn ensemble methods.
        - 1:00–4:00 PM: Practice voting classifier.

    - **Milestone**: Build and evaluate a voting classifier, commit to GitHub.
- **Day 56: Project – Churn Prediction**
    - **Tasks** (7 hours):
        - **Setup** (1 hour): Select Kaggle's Telco Customer Churn dataset. Explore features (e.g., tenure, contract type).
        - **Preprocess and Model** (4 hours): Clean data (handle missing values, encode categoricals with OneHotEncoder). Train multiple models (logistic regression, random forest, XGBoost) with a preprocessing pipeline.
        - **Evaluate** (2 hours): Compute accuracy, precision, recall, F1; visualize ROC curve (sklearn.metrics.roc_curve) and confusion matrix.
    - **Why It Matters**: Churn prediction is a common business problem, showcasing classification skills.
    - **Resources**:
        - Kaggle: Telco Churn.
        - Scikit-learn: ROC Curve.
        - Towards Data Science: Churn Prediction.
    - **Tips**:
        - Handle class imbalance with imblearn.over_sampling.SMOTE (pip install imblearn).
        - Save ROC curve as PNG.
        - Save as day56_churn_project.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select and explore dataset.
        - 10:00 AM–2:00 PM: Preprocess and model.
        - 2:00–4:00 PM: Evaluate and visualize.
    - **Milestone**: Complete a churn prediction project with multiple models, commit to GitHub.
- **Day 57: Enhance Churn Project**
    - **Tasks** (6 hours):
        - **Improve Models** (3 hours): Tune hyperparameters for XGBoost using GridSearchCV (e.g., learning_rate=[0.01, 0.1], max_depth=[3, 5]). Add feature importance plots.
        - **Document** (3 hours): Write detailed Markdown sections (Problem, Data Cleaning, Modeling, Results, Limitations). Add comments to code for clarity.
    - **Why It Matters**: Tuning and documentation make the project professional and appealing to recruiters.
    - **Resources**:

        - XGBoost Tuning Guide.
        - Jupyter Best Practices.
        - Kaggle: Churn Notebooks.
    - **Tips**:
        - Explain tuning choices (e.g., “Lowered learning_rate to reduce overfitting”).
        - Save visualizations in the notebook and as PNGs.
        - Save as day57_churn_enhanced.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Improve models and visualize.
        - 1:00–4:00 PM: Document project.
    - **Milestone**: Enhance churn project with tuning and documentation, commit to GitHub.
- **Day 58: Finalize Churn Project**
    - **Tasks** (6 hours):
        - **Clean Up** (2 hours): Ensure notebook is reproducible (Kernel > Restart & Run All), remove unused code, and organize sections.
        - **Publish** (3 hours): Create a GitHub repository, write a detailed README (include dataset, methodology, results, visuals), and upload notebook.
        - **Share** (1 hour): Share the project link on LinkedIn with a post (e.g., “Developed a churn prediction model using XGBoost!”).
    - **Why It Matters**: A polished project strengthens your portfolio and demonstrates end-to-end ML skills.
    - **Resources**:
        - GitHub README Templates.
        - Kaggle: Churn Notebooks.
    - **Tips**:
        - Include a screenshot of the ROC curve in the README.
        - Test notebook for errors before publishing.
        - Save as day58_churn_final.ipynb, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Clean up notebook.
        - 11:00 AM–2:00 PM: Publish to GitHub.
        - 2:00–3:00 PM: Share on LinkedIn.
    - **Milestone**: Publish a professional churn prediction project, share on LinkedIn, commit to GitHub.

Week 8.5: Model Deployment Basics (Days 59–60)

Goal: Learn to deploy ML models to showcase projects interactively.

- **Day 59: Introduction to Model Deployment**
  - **Tasks** (6 hours):
    - **Learn Deployment** (3 hours): Study Flask for creating simple web APIs to serve ML models. Understand pickling models (joblib.dump) and basic HTML for frontend.
    - **Practice** (3 hours): Pickle the best model from Day 56 (churn project). Create a simple Flask app (app.py) to load the model and predict on sample input.
  - **Why It Matters**: Deployment demonstrates your ability to make models accessible, a valuable skill for ML roles.
  - **Resources**:
    - Flask Quickstart.
    - Real Python: Flask.
    - Towards Data Science: Model Deployment.
  - **Tips**:
    - Install Flask (pip install flask) and test locally (flask run).
    - Use a simple HTML form to accept input (e.g., tenure, contract type).
    - Save as day59_flask_deployment.py and commit to GitHub.
  - **Time Breakdown**:
    - 9:00–12:00 PM: Learn Flask and deployment.
    - 1:00–4:00 PM: Practice deploying churn model.
  - **Milestone**: Create a Flask app to serve a model, commit to GitHub.
- **Day 60: Deploy Model with Streamlit**
  - **Tasks** (6 hours):
    - **Learn Streamlit** (3 hours): Study Streamlit (pip install streamlit) for creating interactive web apps. Learn to create input fields and display predictions.
    - **Practice** (3 hours): Build a Streamlit app for the churn model. Add input fields (e.g., dropdown for contract type, slider for tenure), display predictions, and visualize feature importance.
  - **Why It Matters**: Streamlit is beginner-friendly and creates professional-looking apps, impressing recruiters.
  - **Resources**:
    - Streamlit Documentation.
    - Towards Data Science: Streamlit.
    - Real Python: Streamlit.

- **Tips**:
  - Run app locally with streamlit run app.py.
  - Save model as churn_model.pkl for reuse.
  - Save as day60_streamlit_app.py, commit to GitHub, and test locally.
- **Time Breakdown**:
  - 9:00–12:00 PM: Learn Streamlit.
  - 1:00–4:00 PM: Build Streamlit app.
- **Milestone**: Deploy churn model with Streamlit, commit to GitHub.

Month 3: Advanced Topics, Final Projects, and Job Preparation (Days 61–90)

Focus: Learn advanced ML topics (time series, neural networks), complete two final projects, and prepare for job applications and interviews in the Indian job market.

Week 9: Time Series and Neural Networks (Days 61–67)

Goal: Learn time series analysis and introductory neural networks.

- **Day 61: Introduction to Time Series**
  - **Tasks** (6 hours):
    - **Learn Time Series** (3 hours): Study time series components (trend, seasonality, noise), stationarity, and moving averages. Explore Pandas for time series (e.g., pd.to_datetime()).
    - **Practice** (3 hours): Load a time series dataset (e.g., Kaggle's Air Passengers). Plot trends and compute rolling averages (df.rolling(window=12).mean()).
  - **Why It Matters**: Time series is critical for forecasting in industries like finance and retail.
  - **Resources**:
    - Kaggle: Time Series.
    - Towards Data Science: Time Series.
    - Pandas: Time Series.
  - **Tips**:
    - Ensure date columns are in datetime format.
    - Save time series plots as PNGs.
    - Save as day61_time_series.ipynb and commit to GitHub.

    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn time series concepts.
        - 1:00–4:00 PM: Practice with dataset.
    - **Milestone**: Analyze and visualize a time series dataset, commit to GitHub.
- **Day 62: ARIMA Models**
    - **Tasks** (6 hours):
        - **Learn ARIMA** (3 hours): Study AutoRegressive Integrated Moving Average models (statsmodels.tsa.arima.model.ARIMA). Understand parameters (p, d, q) and ACF/PACF plots.
        - **Practice** (3 hours): Fit an ARIMA model on the Air Passengers dataset, forecast 12 months, and visualize predictions.
    - **Why It Matters**: ARIMA is a standard forecasting model for time series data.
    - **Resources**:
        - Statsmodels: ARIMA.
        - Towards Data Science: ARIMA.
        - Kaggle: Time Series Notebooks.
    - **Tips**:
        - Use adfuller test to check stationarity (statsmodels.tsa.stattools.adfuller).
        - Start with ARIMA(1,1,1) and adjust based on ACF/PACF.
        - Save as day62_arima.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn ARIMA.
        - 1:00–4:00 PM: Practice forecasting.
    - **Milestone**: Fit and forecast with ARIMA, commit to GitHub.
- **Day 63: Project – Time Series Forecasting**
    - **Tasks** (7 hours):
        - **Setup** (1 hour): Select Kaggle's Store Sales Time Series dataset (or Air Passengers for simplicity).
        - **Model and Forecast** (4 hours): Preprocess (convert dates, handle missing values), fit ARIMA, and forecast future values. Visualize actual vs. predicted values.
        - **Document** (2 hours): Write a Jupyter Notebook with sections (Problem, Preprocessing, Modeling, Results). Include visualizations and insights (e.g., "Sales show strong seasonality").
    - **Why It Matters**: Time series projects demonstrate forecasting skills, valuable for analytics roles.
    - **Resources**:
        - Kaggle: Store Sales.

        - Statsmodels Documentation.
        - Towards Data Science: Time Series.
    - **Tips**:
        - Decompose time series with seasonal_decompose for insights.
        - Save visualizations as PNGs for documentation.
        - Save as day63_time_series_project.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–2:00 PM: Model and forecast.
        - 2:00–4:00 PM: Document project.
    - **Milestone**: Complete a time series forecasting project, commit to GitHub.
- **Day 64: Finalize Time Series Project**
    - **Tasks** (6 hours):
        - **Enhance** (3 hours): Add decomposition plots (statsmodels.tsa.seasonal_decompose) and try a second model (e.g., SARIMA for seasonality).
        - **Publish** (3 hours): Clean notebook, write a detailed README (dataset, methodology, results), and upload to GitHub. Share on LinkedIn (e.g., "Built a sales forecasting model using ARIMA!").
    - **Why It Matters**: A polished project showcases your ability to handle time series data.
    - **Resources**:
        - Statsmodels: SARIMA.
        - GitHub README Templates.
    - **Tips**:
        - Include a section on model limitations (e.g., "ARIMA assumes linear trends").
        - Test notebook reproducibility.
        - Save as day64_time_series_final.ipynb, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Enhance project.
        - 1:00–4:00 PM: Publish and share.
    - **Milestone**: Publish a professional time series project, share on LinkedIn, commit to GitHub.
- **Day 65: Introduction to Neural Networks**
    - **Tasks** (6 hours):

        - **Learn Neural Networks** (3 hours): Study perceptrons, layers (input, hidden, output), activation functions (sigmoid, ReLU), and backpropagation. Explore tensorflow or keras (pip install tensorflow).
        - **Practice** (3 hours): Build a simple neural network (keras.Sequential) for Iris classification. Use 2 hidden layers with 10 neurons each.
    - **Why It Matters**: Neural networks are foundational for deep learning, increasingly required in ML roles.
    - **Resources**:
        - TensorFlow: Keras Basics.
        - StatQuest: Neural Networks.
        - Towards Data Science: Neural Networks.
    - **Tips**:
        - Start with a small network to avoid long training times.
        - Use categorical_crossentropy loss for classification.
        - Save as day65_neural_networks.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn neural networks.
        - 1:00–4:00 PM: Practice building a model.
    - **Milestone**: Build a simple neural network for classification, commit to GitHub.
- **Day 66: Neural Network Optimization**
    - **Tasks** (6 hours):
        - **Learn Optimization** (3 hours): Study optimizers (SGD, Adam), learning rate, and batch size. Explore dropout (keras.layers.Dropout) to prevent overfitting.
        - **Practice** (3 hours): Add dropout to the Iris neural network, experiment with learning rates (e.g., 0.001, 0.01), and plot training/validation loss.
    - **Why It Matters**: Optimization improves neural network performance and generalization.
    - **Resources**:
        - TensorFlow: Keras Optimizers.
        - Real Python: Keras.
        - Towards Data Science: Dropout.
    - **Tips**:
        - Use model.fit(..., validation_split=0.2) to monitor validation loss.
        - Save loss plots as PNGs.
        - Save as day66_nn_optimization.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn optimization techniques.

        - 1:00–4:00 PM: Practice optimizing neural network.
    - **Milestone**: Optimize a neural network, visualize training loss, commit to GitHub.
- **Day 67: Mini Project – Image Classification**
    - **Tasks** (7 hours):
        - **Setup** (1 hour): Select Kaggle's MNIST or CIFAR-10 dataset (simpler subset, e.g., 1000 images per class).
        - **Build and Train** (4 hours): Preprocess images (normalize with X / 255.0). Build a neural network with Keras (2–3 hidden layers, dropout). Train and evaluate accuracy.
        - **Document** (2 hours): Write a Jupyter Notebook with sections (Problem, Preprocessing, Model, Results). Include accuracy plot and sample predictions.
    - **Why It Matters**: Image classification introduces computer vision, a high-demand skill in ML.
    - **Resources**:
        - Kaggle: MNIST.
        - TensorFlow: Image Classification.
        - Towards Data Science: Image Classification.
    - **Tips**:
        - Use a small subset to speed up training.
        - Visualize sample images with matplotlib.pyplot.imshow().
        - Save as day67_image_classification.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–2:00 PM: Build and train model.
        - 2:00–4:00 PM: Document project.
    - **Milestone**: Complete an image classification project, commit to GitHub.

Week 10: Final Project and Deployment (Days 68–74)

Goal: Build a comprehensive final project and deploy it online.

- **Day 68: Plan Final Project – Predictive Maintenance**
    - **Tasks** (6 hours):
        - **Select Dataset** (1 hour): Choose Kaggle's Predictive Maintenance dataset (or similar, e.g., Machine Failure). Explore features (e.g., sensor readings, failure labels).

        - **Plan** (3 hours): Outline questions (e.g., “Which features predict machine failure?”). Create a Jupyter Notebook with sections (Introduction, EDA, Preprocessing, Modeling, Deployment).
        - **EDA** (2 hours): Perform initial EDA (visualize distributions, check correlations, handle missing values).
    - **Why It Matters**: Predictive maintenance is a real-world ML application in industries like manufacturing.
    - **Resources**:
        - Kaggle: Predictive Maintenance.
        - Towards Data Science: Predictive Maintenance.
        - Kaggle: EDA Guide.
    - **Tips**:
        - Focus on imbalanced data (failures are rare).
        - Save initial EDA as day68_final_project_plan.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Select dataset.
        - 10:00 AM–1:00 PM: Plan and outline.
        - 1:00–3:00 PM: Perform EDA.
    - **Milestone**: Select dataset, outline project, complete initial EDA, commit to GitHub.
- **Day 69: Final Project – Preprocessing and Modeling**
    - **Tasks** (7 hours):
        - **Preprocess** (3 hours): Handle missing values (fillna), encode categoricals (OneHotEncoder), scale features (StandardScaler). Address class imbalance with SMOTE.
        - **Model** (4 hours): Train multiple models (logistic regression, random forest, XGBoost). Use GridSearchCV to tune the best model (e.g., XGBoost).
    - **Why It Matters**: Robust preprocessing and modeling are critical for accurate predictions.
    - **Resources**:
        - Scikit-learn: Preprocessing.
        - Imbalanced-learn: SMOTE.
        - Kaggle: Predictive Maintenance Notebooks.
    - **Tips**:
        - Save preprocessing pipeline with joblib.dump.
        - Compare model performance in a table (e.g., pd.DataFrame({'Model': [...], 'F1': [...]})).
        - Save as day69_final_project_modeling.ipynb and commit to GitHub.

    - **Time Breakdown**:
        - 9:00–12:00 PM: Preprocess data.
        - 12:00–4:00 PM: Train and tune models.
    - **Milestone**: Preprocess data and train models for final project, commit to GitHub.
- **Day 70: Final Project – Evaluation and Visualization**
    - **Tasks** (6 hours):
        - **Evaluate** (3 hours): Compute metrics (accuracy, precision, recall, F1) for all models. Visualize ROC curves and confusion matrices.
        - **Visualize** (3 hours): Create feature importance plots for XGBoost, visualize predictions vs. actuals, and add EDA visualizations (e.g., correlation heatmap).
    - **Why It Matters**: Evaluation and visualization communicate model performance to stakeholders.
    - **Resources**:
        - Scikit-learn: Metrics.
        - Seaborn Visualization.
        - Towards Data Science: Model Evaluation.
    - **Tips**:
        - Save visualizations as PNGs for the README.
        - Include a comparison table of model performance.
        - Save as day70_final_project_evaluation.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Evaluate models.
        - 1:00–4:00 PM: Create visualizations.
    - **Milestone**: Evaluate models and visualize results, commit to GitHub.
- **Day 71: Final Project – Deployment with Streamlit**
    - **Tasks** (6 hours):
        - **Build App** (4 hours): Create a Streamlit app to serve the best model (e.g., XGBoost). Add input fields for features (e.g., sliders for sensor readings) and display predictions with confidence scores.
        - **Test Locally** (2 hours): Test the app locally (streamlit run app.py), ensure it handles inputs correctly, and debug errors.
    - **Why It Matters**: Deployment makes your project interactive and showcases end-to-end skills.
    - **Resources**:
        - Streamlit Documentation.
        - Towards Data Science: Streamlit Deployment.
        - Real Python: Streamlit.

    - **Tips**:
        - Load the pickled model (joblib.load('model.pkl')).
        - Add a title and description to the app for clarity.
        - Save as day71_final_project_streamlit.py and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–1:00 PM: Build Streamlit app.
        - 1:00–3:00 PM: Test locally.
    - **Milestone**: Deploy final project model with Streamlit, commit to GitHub.
- **Day 72: Final Project – Cloud Deployment**
    - **Tasks** (6 hours):
        - **Learn Cloud Deployment** (3 hours): Study Streamlit Community Cloud (free for hosting Streamlit apps). Understand requirements.txt and GitHub integration.
        - **Deploy** (3 hours): Create a requirements.txt for the Streamlit app (pip freeze > requirements.txt). Push the app to a GitHub repository and deploy to Streamlit Community Cloud.
    - **Why It Matters**: Cloud deployment makes your project accessible online, impressing recruiters.
    - **Resources**:
        - Streamlit Community Cloud.
        - Towards Data Science: Streamlit Deployment.
        - Streamlit Deployment Guide.
    - **Tips**:
        - Ensure requirements.txt includes streamlit, scikit-learn, xgboost, etc.
        - Test the deployed app URL to confirm it works.
        - Save as day72_final_project_cloud.py, create a GitHub repository, and commit.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn cloud deployment.
        - 1:00–4:00 PM: Deploy to Streamlit Cloud.
    - **Milestone**: Deploy final project to Streamlit Cloud, commit to GitHub.
- **Day 73: Final Project – Documentation and Sharing**
    - **Tasks** (6 hours):
        - **Document** (3 hours): Write a detailed Jupyter Notebook with sections (Problem, EDA, Preprocessing, Modeling, Deployment, Limitations). Add code comments and visualizations.
        - **Publish and Share** (3 hours): Create a GitHub repository, write a comprehensive README (include dataset, methodology, deployment link,

screenshots). Share on LinkedIn with a post (e.g., “Deployed a predictive maintenance model with Streamlit!”).

- **Why It Matters**: Professional documentation and sharing increase visibility and credibility.
- **Resources**:
  - GitHub README Templates.
  - Jupyter Best Practices.
  - Kaggle: Predictive Maintenance Notebooks.
- **Tips**:
  - Include the Streamlit app URL in the README.
  - Test notebook reproducibility before publishing.
  - Save as day73_final_project_documentation.ipynb, create a GitHub repository, and commit.
- **Time Breakdown**:
  - 9:00–12:00 PM: Document project.
  - 1:00–4:00 PM: Publish and share.
- **Milestone**: Publish final project with documentation, share on LinkedIn, commit to GitHub.

- **Day 74: Portfolio Update**
  - **Tasks** (6 hours):
    - **Update Portfolio** (3 hours): Add the predictive maintenance project to your GitHub portfolio (DataSciencePortfolio). Update the main README with a project summary and link.
    - **Polish Projects** (2 hours): Review all projects (Days 14, 21, 29, 37, 44, 49, 58, 64, 73) for consistency, clear READMEs, and reproducibility.
    - **Share Portfolio** (1 hour): Update LinkedIn profile with the portfolio link and share a post summarizing your journey (e.g., “Completed a 90-day Data Science roadmap with 6 projects!”).
  - **Why It Matters**: A polished portfolio is your primary tool for job applications.
  - **Resources**:
    - GitHub Portfolio Examples.
    - Towards Data Science: Portfolio Building.
  - **Tips**:
    - Highlight the deployed project in your portfolio README.
    - Ensure all project links are accessible.
    - Save updated portfolio README as portfolio_readme.md and commit to GitHub.
  - **Time Breakdown**:

        - 9:00–12:00 PM: Update portfolio.
        - 1:00–3:00 PM: Polish projects.
        - 3:00–4:00 PM: Share portfolio.
    - **Milestone**: Update and share a professional portfolio with 6 projects, commit to GitHub.

Week 11: Job Preparation and Interview Skills (Days 75–81)

Goal: Prepare for job applications, build a resume, and practice technical interviews.

- **Day 75: Resume Building**
    - **Tasks** (6 hours):
        - **Create Resume** (3 hours): Use a clean template (e.g., Overleaf LaTeX or Canva). Include sections: Contact Info, Objective, Skills (Python, Pandas, Scikit-learn, SQL, etc.), Projects (list 4–6 portfolio projects with brief descriptions and GitHub links), Education, Certifications (e.g., Kaggle courses).
        - **Polish** (3 hours): Tailor resume for data science roles in India (e.g., Junior Data Scientist, ML Engineer). Highlight projects and skills relevant to job descriptions (e.g., from Naukri).
    - **Why It Matters**: A tailored resume is critical for passing initial screenings in the Indian job market.
    - **Resources**:
        - Overleaf: Resume Templates.
        - Canva: Resume Templates.
        - Towards Data Science: Data Science Resume.
    - **Tips**:
        - Keep resume to 1 page, use action verbs (e.g., “Developed,” “Deployed”).
        - Include GitHub and LinkedIn links.
        - Save as resume.tex (if using LaTeX) or resume.pdf and store locally (do not commit to GitHub).
    - **Time Breakdown**:
        - 9:00–12:00 PM: Create resume.
        - 1:00–4:00 PM: Polish and tailor.
    - **Milestone**: Create a professional data science resume tailored for ₹10 LPA roles.
- **Day 76: LinkedIn Profile Optimization**
    - **Tasks** (6 hours):
        - **Optimize Profile** (3 hours): Update LinkedIn with a professional photo, headline (e.g., “Aspiring Data Scientist | Python, ML, SQL”), and About

section summarizing your journey and skills. Add projects with GitHub links.
        - **Network** (3 hours): Connect with 20–30 data scientists, recruiters, or alumni in India (e.g., from TCS, ZS Associates, or startups). Comment on 5 data science posts to increase visibility.
    - **Why It Matters**: LinkedIn is a primary platform for networking and job opportunities in India.
    - **Resources**:
        - LinkedIn Learning: Profile Optimization.
        - Towards Data Science: LinkedIn for Data Scientists.
    - **Tips**:
        - Personalize connection requests (e.g., "Inspired by your churn prediction post!").
        - Add portfolio projects to the Projects section.
        - Save a screenshot of your updated profile as linkedin_profile.png (do not commit).
    - **Time Breakdown**:
        - 9:00–12:00 PM: Optimize profile.
        - 1:00–4:00 PM: Network and engage.
    - **Milestone**: Optimize LinkedIn profile and start networking.
- **Day 77: Job Search Strategy**
    - **Tasks** (6 hours):
        - **Research Jobs** (3 hours): Browse Naukri, Instahyre, and Wellfound for Junior Data Scientist, ML Engineer, or Data Analyst roles. Identify 10–15 job descriptions with requirements (e.g., Python, SQL, ML).
        - **Plan Applications** (3 hours): Create a spreadsheet to track applications (columns: Company, Role, Date Applied, Status, Notes). Shortlist 5 roles to apply for this week.
    - **Why It Matters**: A strategic job search targets roles matching your skills, increasing success rates.
    - **Resources**:
        - Naukri.com (search "Junior Data Scientist").
        - Instahyre (filter for entry-level roles).
        - Wellfound (search startups in India).
    - **Tips**:
        - Focus on companies like TCS, Infosys, ZS Associates, Fractal Analytics, or startups.
        - Save job descriptions as PDFs for reference.

        - Save application spreadsheet as job_applications.xlsx (do not commit).
    - **Time Breakdown**:
        - 9:00–12:00 PM: Research jobs.
        - 1:00–4:00 PM: Plan applications.
    - **Milestone**: Identify 10–15 job roles and create an application tracker.
- **Day 78: Cover Letter and Applications**
    - **Tasks** (6 hours):
        - **Write Cover Letter** (3 hours): Create a template cover letter (300–400 words) tailored for data science roles. Highlight your projects, skills, and enthusiasm for the role.
        - **Apply to Jobs** (3 hours): Apply to 5 roles on Naukri, Instahyre, or Wellfound. Customize the cover letter for each and attach your resume.
    - **Why It Matters**: A tailored cover letter sets you apart in competitive job markets.
    - **Resources**:
        - The Muse: Cover Letter Guide.
        - Towards Data Science: Cover Letter for Data Science.
    - **Tips**:
        - Address the cover letter to the hiring manager if possible.
        - Mention specific projects (e.g., “My churn prediction model achieved 85% F1-score”).
        - Save cover letter template as cover_letter.docx (do not commit).
    - **Time Breakdown**:
        - 9:00–12:00 PM: Write cover letter.
        - 1:00–4:00 PM: Submit applications.
    - **Milestone**: Write a cover letter and apply to 5 jobs.
- **Day 79: Coding Interview Preparation**
    - **Tasks** (6 hours):
        - **Learn Coding Concepts** (3 hours): Review Python basics (lists, dictionaries, loops), algorithms (sorting, searching), and data structures (stacks, queues). Study common interview questions (e.g., reverse a list, find duplicates).
        - **Practice** (3 hours): Solve 10 medium-level problems on LeetCode or HackerRank (e.g., “Two Sum,” “Valid Parentheses”).
    - **Why It Matters**: Coding interviews test problem-solving and Python proficiency, common in Indian tech roles.
    - **Resources**:
        - LeetCode: Python Problems.
        - HackerRank: Python Challenges.

        - Cracking the Coding Interview.
    - **Tips**:
        - Practice explaining your code aloud.
        - Save solutions as day79_coding_practice.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review coding concepts.
        - 1:00–4:00 PM: Solve problems.
    - **Milestone**: Solve 10 coding problems, commit to GitHub.
- **Day 80: SQL Interview Preparation**
    - **Tasks** (6 hours):
        - **Review SQL** (3 hours): Revisit SELECT, JOIN, GROUP BY, subqueries, and window functions (e.g., ROW_NUMBER()). Study common interview questions (e.g., find top N salaries).
        - **Practice** (3 hours): Solve 10 SQL problems on StrataScratch or LeetCode (e.g., “Combine Two Tables,” “Second Highest Salary”).
    - **Why It Matters**: SQL is a core requirement for data science and analyst roles in India.
    - **Resources**:
        - StrataScratch SQL Questions.
        - LeetCode: SQL Problems.
        - Mode Analytics: SQL Tutorial.
    - **Tips**:
        - Practice writing queries without auto-complete to simulate interviews.
        - Save queries as day80_sql_practice.txt and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review SQL concepts.
        - 1:00–4:00 PM: Solve SQL problems.
    - **Milestone**: Solve 10 SQL problems, commit to GitHub.
- **Day 81: ML Interview Preparation**
    - **Tasks** (6 hours):
        - **Learn ML Concepts** (3 hours): Review supervised/unsupervised learning, overfitting, bias-variance tradeoff, and metrics (F1, RMSE). Study common questions (e.g., “Explain gradient descent”).
        - **Practice** (3 hours): Prepare answers for 5 ML questions (e.g., “How does random forest work?”). Practice explaining a project (e.g., churn prediction) in 2–3 minutes.
    - **Why It Matters**: ML interviews test theoretical and practical knowledge, critical for data science roles.

    - **Resources**:
        - Towards Data Science: ML Interview Questions.
        - Springboard: Data Science Interview.
        - StatQuest: ML Concepts.
    - **Tips**:
        - Use STAR method (Situation, Task, Action, Result) for project explanations.
        - Save answers as day81_ml_interview.md and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review ML concepts.
        - 1:00–4:00 PM: Practice interview answers.
    - **Milestone**: Prepare for 5 ML interview questions, commit to GitHub.

Week 12: Mock Interviews and Final Preparations (Days 82–88)

Goal: Conduct mock interviews, refine skills, and apply to more jobs.

- **Day 82: Mock Coding Interview**
    - **Tasks** (6 hours):
        - **Prepare** (2 hours): Review coding concepts (arrays, dictionaries, recursion). Select 5 medium-level problems from LeetCode (e.g., “Group Anagrams”).
        - **Mock Interview** (4 hours): Simulate a 1-hour coding interview (solve 2 problems, explain aloud). Record yourself or practice with a friend. Review solutions and optimize code.
    - **Why It Matters**: Mock interviews build confidence and identify weaknesses.
    - **Resources**:
        - LeetCode: Medium Problems.
        - Interviewing.io (for mock interview tips).
        - Cracking the Coding Interview.
    - **Tips**:
        - Time yourself (30 minutes per problem).
        - Save solutions as day82_mock_coding.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Review concepts and select problems.
        - 11:00 AM–3:00 PM: Conduct mock interview.
    - **Milestone**: Complete a mock coding interview, commit solutions to GitHub.
- **Day 83: Mock SQL Interview**
    - **Tasks** (6 hours):

        - **Prepare** (2 hours): Review joins, aggregations, and window functions. Select 5 SQL problems from StrataScratch (e.g., “Customer Order Frequency”).
        - **Mock Interview** (4 hours): Simulate a 1-hour SQL interview (solve 2–3 problems, explain logic). Use an online editor like DB Fiddle.
    - **Why It Matters**: SQL interviews are common for data science roles in India.
    - **Resources**:
        - StrataScratch SQL Questions.
        - DB Fiddle (for SQL practice).
        - Mode Analytics: SQL Tutorial.
    - **Tips**:
        - Explain query logic step-by-step as if to an interviewer.
        - Save queries as day83_mock_sql.txt and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Review SQL concepts.
        - 11:00 AM–3:00 PM: Conduct mock interview.
    - **Milestone**: Complete a mock SQL interview, commit solutions to GitHub.
- **Day 84: Mock ML Interview**
    - **Tasks** (6 hours):
        - **Prepare** (2 hours): Review ML concepts (e.g., regularization, ensemble methods). Prepare explanations for 5 projects (e.g., churn, predictive maintenance).
        - **Mock Interview** (4 hours): Simulate a 1-hour ML interview. Answer 3 theoretical questions (e.g., “What is overfitting?”) and explain 1 project in detail.
    - **Why It Matters**: ML interviews test your ability to explain technical concepts and projects.
    - **Resources**:
        - Towards Data Science: ML Interview Questions.
        - Springboard: Data Science Interview.
    - **Tips**:
        - Practice explaining projects using STAR method.
        - Save answers as day84_mock_ml.md and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–11:00 AM: Review ML concepts.
        - 11:00 AM–3:00 PM: Conduct mock interview.
    - **Milestone**: Complete a mock ML interview, commit answers to GitHub.
- **Day 85: Behavioral Interview Preparation**

    - **Tasks** (6 hours):
        - **Learn Behavioral Questions** (3 hours): Study common questions (e.g., “Tell me about a challenge you faced,” “Why data science?”). Use STAR method for answers.
        - **Practice** (3 hours): Write and practice answers for 5 questions. Record yourself or practice with a friend to improve delivery.
    - **Why It Matters**: Behavioral interviews assess communication and cultural fit, critical in Indian companies.
    - **Resources**:
        - The Muse: Behavioral Interview Questions.
        - Glassdoor: Interview Questions (search “Data Scientist”).
    - **Tips**:
        - Tailor answers to data science (e.g., “Overcame a challenge by debugging a model”).
        - Save answers as day85_behavioral.md and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Learn behavioral questions.
        - 1:00–4:00 PM: Practice answers.
    - **Milestone**: Prepare and practice 5 behavioral interview answers, commit to GitHub.
- **Day 86: Apply to More Jobs**
    - **Tasks** (6 hours):
        - **Research** (3 hours): Identify 10–15 new job postings on Naukri, Instahyre, or Wellfound. Focus on companies like Accenture, Fractal Analytics, or startups.
        - **Apply** (3 hours): Apply to 5–10 roles, customizing cover letters and updating the application tracker.
    - **Why It Matters**: Consistent applications increase your chances of landing interviews.
    - **Resources**:
        - Naukri.com.
        - Instahyre.
        - Wellfound.
    - **Tips**:
        - Prioritize roles matching your skills (e.g., Python, ML, SQL).
        - Update tracker with application details.
        - Save updated tracker as job_applications.xlsx (do not commit).
    - **Time Breakdown**:

        - 9:00–12:00 PM: Research jobs.
        - 1:00–4:00 PM: Submit applications.
    - **Milestone**: Apply to 5–10 additional jobs, update tracker.
- **Day 87: Kaggle Competition**
    - **Tasks** (7 hours):
        - **Join Competition** (1 hour): Select a beginner-friendly Kaggle competition (e.g., Titanic or House Prices).
        - **Build Model** (4 hours): Preprocess data, train a model (e.g., random forest or XGBoost), and submit predictions.
        - **Document** (2 hours): Write a Jupyter Notebook summarizing your approach and results. Submit to Kaggle and note your leaderboard rank.
    - **Why It Matters**: Kaggle competitions demonstrate competitive skills and are recognized by employers.
    - **Resources**:
        - Kaggle Competitions.
        - Kaggle: Getting Started.
        - Towards Data Science: Kaggle Guide.
    - **Tips**:
        - Start with a simple model to submit quickly.
        - Save as day87_kaggle_competition.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–10:00 AM: Join competition.
        - 10:00 AM–2:00 PM: Build and submit model.
        - 2:00–4:00 PM: Document and submit to Kaggle.
    - **Milestone**: Submit to a Kaggle competition, commit notebook to GitHub.
- **Day 88: Portfolio Finalization**
    - **Tasks** (6 hours):
        - **Review Portfolio** (3 hours): Ensure all projects (Days 14, 21, 29, 37, 44, 49, 58, 64, 73, 87) have clean code, detailed READMEs, and visualizations. Update the main portfolio README.
        - **Add Kaggle** (2 hours): Add the Kaggle competition notebook to the portfolio with a brief description.
        - **Share** (1 hour): Share the final portfolio on LinkedIn and data science communities (e.g., Reddit's r/datascience).
    - **Why It Matters**: A finalized portfolio maximizes your appeal to recruiters.
    - **Resources**:
        - GitHub Portfolio Examples.
        - Reddit: r/datascience.

    - **Tips**:
        - Highlight the deployed project and Kaggle rank in the portfolio README.
        - Save updated portfolio README as portfolio_readme.md and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review portfolio.
        - 1:00–3:00 PM: Add Kaggle project.
        - 3:00–4:00 PM: Share portfolio.
    - **Milestone**: Finalize and share portfolio with 7 projects and Kaggle submission, commit to GitHub.

Week 13: Final Review and Interviews (Days 89–90)

Goal: Consolidate skills, conduct final mock interviews, and follow up on applications.

- **Day 89: Final Skill Review**
    - **Tasks** (6 hours):
        - **Review Concepts** (3 hours): Skim notes and notebooks from Days 1–88, focusing on Python, Pandas, Scikit-learn, SQL, and ML concepts (e.g., regression, classification, clustering).
        - **Practice** (3 hours): Solve 5 mixed problems (2 coding, 2 SQL, 1 ML) from LeetCode, StrataScratch, or Kaggle.
    - **Why It Matters**: A final review ensures retention and readiness for interviews.
    - **Resources**:
        - LeetCode.
        - StrataScratch.
        - Kaggle: Learn.
    - **Tips**:
        - Create a cheat sheet of key concepts (e.g., SQL joins, ML metrics).
        - Save solutions as day89_final_review.ipynb and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–12:00 PM: Review concepts.
        - 1:00–4:00 PM: Solve mixed problems.
    - **Milestone**: Review all skills, solve 5 problems, commit to GitHub.
- **Day 90: Final Mock Interviews and Follow-Ups**
    - **Tasks** (7 hours):
        - **Mock Interviews** (4 hours): Conduct a 1-hour coding interview (2 problems), 1-hour SQL interview (2 problems), and 1-hour ML interview (explain 1 project, answer 2 questions).

        - **Follow Up** (2 hours): Check application statuses on Naukri/Instahyre, send follow-up emails to 2–3 recruiters (e.g., “Thank you for considering my application for the Junior Data Scientist role”).
        - **Reflect** (1 hour): Write a reflection on your 90-day journey, noting strengths and areas for improvement. Plan next steps (e.g., apply to more jobs, join a bootcamp).
    - **Why It Matters**: Final practice and follow-ups prepare you for real interviews and maintain momentum.
    - **Resources**:
        - LeetCode.
        - StrataScratch.
        - The Muse: Follow-Up Email.
    - **Tips**:
        - Record mock interviews to review delivery.
        - Save reflection as day90_reflection.md and commit to GitHub.
    - **Time Breakdown**:
        - 9:00–1:00 PM: Conduct mock interviews.
        - 1:00–3:00 PM: Follow up on applications.
        - 3:00–4:00 PM: Write reflection.
    - **Milestone**: Complete final mock interviews, follow up on applications, commit reflection to GitHub.

Post-90 Days: Next Steps

- **Continue Applying**: Aim for 5–10 applications per week on Naukri, Instahyre, and Wellfound.
- **Engage in Communities**: Join data science meetups (e.g., Bangalore Data Science Meetup) and contribute to Kaggle discussions.
- **Upskill**: Explore advanced topics (e.g., deep learning with TensorFlow, big data with PySpark) or take certifications (e.g., Google Data Analytics, AWS Certified ML).
- **Track Progress**: Update your application tracker and follow up on interviews weekly.
- **Stay Persistent**: Landing a ₹10 LPA job may take 100+ applications. Refine your approach based on feedback.

Expected Outcomes by Day 90

- **Skills**: Proficient in Python, Pandas, NumPy, Scikit-learn, SQL, Matplotlib, Seaborn, Plotly, Streamlit, and basic neural networks.
- **Portfolio**: 7+ projects (2 EDA, housing prediction, spam detection, customer segmentation, churn prediction, time series, predictive maintenance, Kaggle competition).
- **Applications**: 15–25 job applications submitted, with follow-ups.
- **Interview Readiness**: Prepared for coding, SQL, ML, and behavioral interviews.
- **Network**: 50+ LinkedIn connections with data professionals in India.

Indian Job Market Tips

- **Target Companies**: TCS, Infosys, Accenture, ZS Associates, Fractal Analytics, Mu Sigma, and startups on Wellfound.
- **Salary Expectation**: ₹10 LPA is achievable for entry-level roles with strong projects and skills in Python, SQL, and ML.
- **Common Requirements**: Job descriptions often emphasize Python, SQL, ML algorithms, and communication skills. Tailor your resume to highlight these.
- **Networking**: Engage with recruiters on LinkedIn and attend virtual data science webinars in India.