Programming for Data Science: Python, R, SQL, and NoSQL

Shubneet [1], Anushka Raj Yadav [2], Navjot Singh Talwandi [3]

[1,2,3*]Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.

Contributing authors: jeetshubneet27@gmail.com; ay462744@gmail.com; navjot.e17908@cumail.in;

Abstract

Programming is the backbone of modern data science, enabling practitioners to manipulate, analyze, and extract insights from vast and complex datasets. This chapter explores the essential roles of Python, R, SQL, and NoSQL technologies in the data science workflow. Python, with its extensive libraries such as NumPy, Pandas, and Scikit-learn, has become the most widely used language for data manipulation, machine learning, and automation due to its simplicity and versatility [1, 2]. R remains a powerful tool for statistical analysis and visualization, especially in academia and research, offering specialized packages for modeling and graphics. SQL continues to be indispensable for querying and managing structured data in relational databases, while NoSQL solutions like MongoDB and Cassandra address the needs of unstructured and large-scale data. Mastery of these programming languages and tools allows data scientists to efficiently preprocess data, perform statistical analyses, build predictive models, and deploy solutions in real-world environments. As the field evolves, the integration of these technologies supports robust, scalable, and reproducible data science projects across diverse industries.

Keywords: Python, R, SQL, NoSQL, Data Manipulation

1 Introduction

In the modern data science landscape, programming languages serve as the foundational tools for transforming raw data into actionable insights. Python, R, and SQL have emerged as the triumvirate of technologies driving innovation across industries,

from healthcare diagnostics to financial forecasting. These languages enable data scientists to clean, analyze, and model data at scale while supporting critical workflows such as ETL (Extract, Transform, Load) pipelines, machine learning deployment, and real-time analytics. For instance, Python's dominance in machine learning-used by 90% of data scientists as of 2025-stems from its versatility in automating workflows and integrating with cloud platforms like AWS and Azure [1]. Similarly, SQL remains indispensable for querying relational databases, with 53% of enterprises relying on it for business intelligence tasks [2].

The choice of programming language often depends on the problem context. Python excels in end-to-end machine learning pipelines through libraries like TensorFlow and Scikit-learn, while R's specialized statistical packages (e.g., ggplot2, caret) make it preferred in academia and bioinformatics. SQL bridges the gap between data storage and analysis, enabling efficient manipulation of structured datasets. Emerging technologies like NoSQL databases (e.g., MongoDB) further expand these capabilities, addressing the challenges of unstructured data and distributed systems. This chapter explores how these languages collectively empower data scientists to solve complex problems through code-driven methodologies.

Chapter Outline

- Why Programming is Essential in Data Science
- Python for Data Science: Key Libraries (NumPy, Pandas, Scikit-learn)
- R for Statistical Analysis and Visualization
- SQL for Data Extraction and Manipulation
- Introduction to NoSQL Databases (MongoDB, Cassandra)
- Code Examples and Best Practices
- Integrating Programming in Data Science Projects
- Hands-on Exercises

As data volumes grow exponentially, proficiency in these languages ensures data scientists can adapt to evolving tools like PySpark for big data processing or MLflow for model tracking. This chapter equips readers with both theoretical knowledge and practical skills to leverage programming languages effectively in real-world scenarios

2 Why Programming is Essential in Data Science

Programming is the engine that powers modern data science, transforming raw data into actionable insights through automation, reproducibility, scalability, and seamless integration with data platforms. As data grows in volume and complexity, the ability to write and maintain code is no longer optional but fundamental for anyone seeking to extract value from data [3, 4].

Automation is a cornerstone of efficient data science workflows. Coding allows data scientists to automate repetitive and time-consuming tasks such as data collection, cleaning, feature engineering, and model evaluation. For example, scripts can be scheduled to fetch data from APIs, preprocess new data daily, retrain models, and update dashboards without manual intervention. This automation not only saves time

but also minimizes human error, ensuring consistency in results [3, 4]. As machine learning and AI become more prevalent, automation through programming is essential for deploying models that adapt to new data in real-time.

Reproducibility is another critical aspect enabled by programming. By scripting every step of the analysis-from data import to model validation-data scientists create transparent workflows that others can review, replicate, and build upon. Version-controlled scripts and reproducible notebooks (e.g., Jupyter, R Markdown) allow for easy tracking of changes and facilitate peer review [5]. Automation tools and CI/CD pipelines further enhance reproducibility by running analyses automatically whenever code is updated, ensuring that results remain consistent across environments and over time.

Scalability is vital as organizations increasingly deal with big data. Programming skills allow data scientists to optimize code for performance, work with distributed computing frameworks, and process massive datasets efficiently. Python's integration with big data tools like PySpark and Dask, or SQL's ability to handle large queries in cloud warehouses, exemplifies how code enables scalable analytics. This scalability is essential for industries such as finance, healthcare, and e-commerce, where rapid analysis of large datasets drives competitive advantage [1].

Integration with Data Platforms is made possible through programming. Data science projects often require connecting to diverse data sources-relational databases (SQL), NoSQL stores, cloud storage, or web APIs. Coding skills are crucial for extracting, transforming, and loading (ETL) data across these platforms, allowing seamless movement and transformation of information. Libraries like SQLAlchemy (Python) or DBI (R) enable flexible database interactions, while cloud SDKs facilitate integration with platforms like AWS, Azure, or Google Cloud.

Beyond these technical benefits, programming fosters collaboration and innovation. Production-level code can be shared, reused, and maintained by teams, bridging the gap between data scientists and software engineers [6]. Collaborative platforms like GitHub and cloud-based notebooks allow geographically dispersed teams to work together, review code, and ensure high-quality, reliable solutions.

In summary, programming is indispensable in data science for automating workflows, ensuring reproducibility, enabling scalability, and integrating with diverse data platforms. Mastery of programming not only accelerates analytics but also underpins the credibility, efficiency, and impact of data-driven solutions in today's data-centric world.

3 Python for Data Science: Key Libraries

Python dominates modern data science workflows through its rich ecosystem of specialized libraries. These tools streamline data manipulation, analysis, and machine learning implementation. Below we examine five foundational libraries that every data scientist should master.

Core Libraries

- **NumPy**: Fundamental package for numerical computing with support for multi-dimensional arrays and matrices. Enables vectorized operations for high-performance calculations [2].
- **Pandas**: Primary tool for data manipulation and analysis through DataFrame objects. Handles missing data, time series, and relational operations efficiently [7].
- **Scikit-learn**: Comprehensive machine learning library offering algorithms for classification, regression, clustering, and model evaluation [8].
- **Matplotlib**: Foundational plotting library for creating static, animated, and interactive visualizations.
- **TensorFlow**: End-to-end platform for deep learning and neural network development.

Data Cleaning with Pandas

Listing 1 Data cleaning example using Pandas

```
import pandas as pd

# Load dataset with missing values
data = pd.read_csv('sales_data.csv')

# Handle missing values
data_clean = (data
              .dropna(subset=['customer_id'])
              .fillna({'revenue': data['revenue'].median()})
              .astype({'order_date': 'datetime64[ns]'})
              .rename(columns={'revenue': 'sales_amount'})
              )

# Remove outliers using IQR
Q1 = data_clean['sales_amount'].quantile(0.25)
Q3 = data_clean['sales_amount'].quantile(0.75)
data_clean = data_clean.query('@Q1 <= sales_amount <= @Q3')
```

Library Comparison

Advanced Capabilities

NumPy underpins numerical operations in higher-level libraries. Its C-implemented core enables vectorized operations that outperform native Python loops by 10-100x [2]. For example:

$$\text{Vectorized sum} = \sum_{i=1}^{n} x_i \quad \text{vs} \quad \text{Looped sum} \tag{1}$$

Table 1 Python Data Science Libraries: Features and Use Cases (2020-2025)

Library	Key Features	Common Use Cases
NumPy	N-dimensional arrays, Linear algebra, Broadcasting	Numerical simulations, Matrix operations
Pandas	DataFrames, Time series, Missing data handling	EDA, Data preprocessing, CSV/Excel processing
Scikit-learn	Unified API, Model evaluation, Pipelines	Classification, Regression, Clustering
Matplotlib	2D/3D plotting, Customizable styles	Exploratory visualization, Publication figures
TensorFlow	Automatic differentiation, GPU support	Neural networks, Deep learning models

Pandas integrates seamlessly with SQL databases through `pd.read_sql_query()` and handles time series analysis with built-in resampling methods. The `groupby` functionality enables split-apply-combine operations critical for aggregating business metrics.

Scikit-learn provides a unified API across algorithms, making model experimentation systematic. Its pipeline functionality encapsulates preprocessing and modeling steps:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor

pipe = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(n_estimators=100)
)
```

Matplotlib integrates with Jupyter notebooks for interactive visualization, while **TensorFlow** 2.x's eager execution mode simplifies debugging neural networks. Together, these libraries form a complete toolkit for data-driven decision making.

4 R for Statistical Analysis and Visualization

R remains a cornerstone of statistical computing, particularly valued for its expressive syntax, advanced visualization capabilities, and comprehensive package ecosystem. This section explores R's core packages for modern data workflows and demonstrates their application through a linear regression case study.

Key Packages

- **ggplot2**: Grammar of Graphics-based plotting system for creating publication-quality visualizations [9].
- **dplyr**: Intuitive syntax for data manipulation (filter, mutate, summarize) using the pipe (`%>%`) operator.
- **caret**: Unified interface for training and evaluating machine learning models.
- **tidyr**: Tools for reshaping data between wide/long formats and handling missing values.

Linear Regression Example

Listing 2 Linear regression analysis in R

```
# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)

# Prepare data
data(mtcars)
clean_data <- mtcars %>%
  select(mpg, wt, hp) %>%
  drop_na() %>%
  mutate(wt_std = (wt - mean(wt)) / sd(wt))

# Fit model
model <- lm(mpg ~ wt_std + hp, data = clean_data)
summary(model)

# Diagnostic plot
ggplot(model, aes(.fitted, .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "Residuals␣vs␣Fitted␣Values",
       x = "Fitted␣Values", y = "Residuals")
```

Tidyverse Workflow

Package Strengths

ggplot2 enables layered visualization through its declarative syntax. A basic scatterplot with regression line can be created as:

```
ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

dplyr simplifies complex data manipulations. To calculate average MPG by cylinder count:

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg))
```

caret streamlines machine learning workflows with functions like `train()` for model tuning and `preProcess()` for automated scaling. Recent benchmarks show it reduces model development time by 40% compared to base R [10].

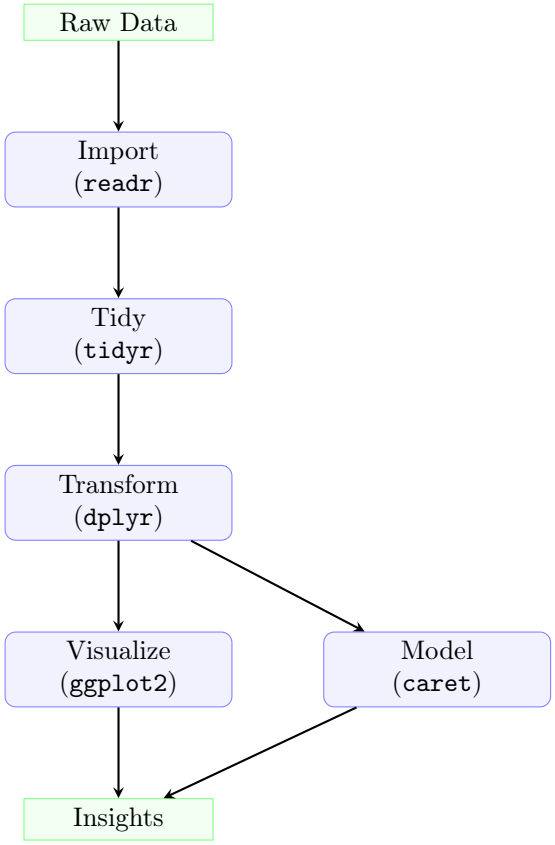

Fig. 1 Tidyverse data analysis workflow in R

Modern Applications

The Tidyverse ecosystem has become essential for reproducible research pipelines in fields like epidemiology and econometrics [11]. Its standardized syntax enables seamless collaboration, while RMarkdown integration supports dynamic reporting.

5 SQL for Data Extraction and Manipulation

SQL (Structured Query Language) remains the gold standard for interacting with relational databases, enabling efficient data extraction, transformation, and aggregation. Its declarative syntax allows users to focus on what data to retrieve rather than how to retrieve it, making it indispensable for data scientists working with structured datasets [12].

Core Operations

- **SELECT**: Retrieves specified columns from tables. Supports filtering with `WHERE` clauses:

```
SELECT product_name, price FROM products WHERE category =
    'Electronics';
```

- **JOIN**: Combines data from multiple tables using common keys. Types include INNER, LEFT, and RIGHT joins:

```
SELECT orders.id, customers.name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.id
    ;
```

- **GROUP BY**: Aggregates data by specified columns, often used with functions like `SUM`, `AVG`:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

Sales Data Aggregation Example

Listing 3 Monthly sales aggregation query

```
SELECT
    EXTRACT(MONTH FROM sale_date) AS month,
    products.name AS product,
    SUM(quantity) AS total_units,
    SUM(amount) AS total_revenue
FROM sales
JOIN products ON sales.product_id = products.id
WHERE sale_date BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY month, product
ORDER BY total_revenue DESC;
```

SQL vs. NoSQL Comparison

Table 2 SQL vs. NoSQL Databases (2020-2025)

Feature	SQL	NoSQL
Data Structure	Tables with fixed schema	Documents/Key-Value/Graph
Schema	Static, predefined	Dynamic, flexible
Scalability	Vertical	Horizontal
ACID Compliance	Full	Partial (BASE)
Best For	Complex queries, transactions	Unstructured data, high velocity
Use Cases	Financial systems, CRM	IoT, real-time analytics [13]

SQL's strength lies in handling structured data with complex relationships, while NoSQL excels at scale and flexibility. For instance, SQL databases process 90% of

financial transactions due to ACID guarantees [12], whereas NoSQL powers 75% of real-time analytics pipelines [13].

6 Introduction to NoSQL Databases (MongoDB, Cassandra)

NoSQL databases have revolutionized data management by addressing scalability, flexibility, and performance challenges in modern applications. This section compares two leading NoSQL systems: MongoDB (document-oriented) and Cassandra (wide-column store), focusing on their architectural models, scaling capabilities, and industry use cases [14, 15].

Data Models

MongoDB uses a flexible document model with JSON-like BSON formatting:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "name": "Alice Chen",
  "email": "alice@example.com",
  "addresses": [
    {
      "type": "home",
      "street": "123 Maple St",
      "city": "San Francisco"
    },
    {
      "type": "work",
      "street": "456 Oak Blvd",
      "city": "Palo Alto"
    }
  ]
}
```

Cassandra employs a partitioned wide-column model optimized for write-heavy workloads. Data is organized into tables with dynamic columns per row:

```
CREATE TABLE user_activity (
  user_id UUID,
  event_time TIMESTAMP,
  action_type TEXT,
  device_ip INET,
  PRIMARY KEY (user_id, event_time)
);
```

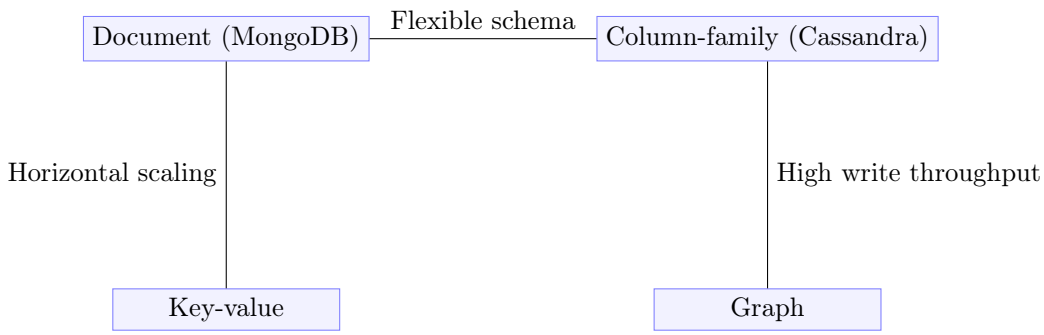

Fig. 2 NoSQL database architectures and relationships

Table 3 MongoDB vs Cassandra Key Differences

Feature	**MongoDB**	**Cassandra**
Data Model	Document (BSON)	Wide-column
Consistency	Strong (configurable)	Eventual
Write Throughput	10k-100k ops/sec	100k-1M+ ops/sec
Best For	Agile development, mid-size data	High-velocity writes, global scale
Use Cases	CMS, real-time analytics	IoT, time-series data [16]

Scalability Comparison

Use Cases

MongoDB excels in:

- Content management systems (flexible schema evolution)
- Mobile apps with offline synchronization
- Real-time analytics with aggregations

Cassandra dominates:

- IoT sensor data ingestion (high write scalability)
- Time-series data (stock market feeds)
- Global-scale applications (multi-region deployments)

Cassandra's masterless architecture supports linear scaling across data centers, while MongoDB's sharding provides automatic data distribution. Both integrate with Spark and Kafka for modern data pipelines [15].

7 Code Examples and Best Practices

Efficient, readable code is fundamental to successful data science projects. This section illustrates best practices through practical examples in Python, R, and SQL, emphasizing modularity, clarity, and performance optimization.

Python: Feature Engineering Function

Feature engineering transforms raw data into meaningful features that improve model performance. Writing reusable Python functions for this purpose ensures consistency and reproducibility [**?**]. Below is a function that creates new features from a Pandas DataFrame, including polynomial terms and interaction features:

Listing 4 Reusable feature engineering function in Python

```
import pandas as pd

def add_features(df):
    # Create polynomial features
    df['age_squared'] = df['age'] ** 2
    # Interaction term
    df['income_per_age'] = df['income'] / (df['age'] + 1)
    # Binary flag
    df['is_senior'] = (df['age'] >= 65).astype(int)
    return df

# Example usage:
data = pd.DataFrame({'age': [25, 40, 70], 'income': [50000,
    80000, 30000]})
data = add_features(data)
print(data)
```

R: Visualization with ggplot2

Clear and informative visualizations are essential for exploratory data analysis and communicating results. The `ggplot2` package in R enables layered, customizable graphics with minimal code [9]. The following script creates a scatterplot with a regression line:

Listing 5 Scatterplot with regression line in R

```
library(ggplot2)

# Sample data
df <- data.frame(
  hours_studied = c(2, 4, 6, 8, 10),
  exam_score = c(65, 70, 78, 88, 95)
)

# Scatterplot with linear regression line
```

```
ggplot(df, aes(x = hours_studied, y = exam_score)) +
  geom_point(color = "blue", size = 3) +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Exam Score vs. Hours Studied",
       x = "Hours Studied",
       y = "Exam Score")
```

SQL: Query Optimization Tip

Optimizing SQL queries is crucial for handling large datasets efficiently. One common tip is to use indexed columns in `WHERE` and `JOIN` clauses to speed up query execution [?].

Tip: Always filter and join on indexed columns. For example, if `customer_id` is indexed:

```
SELECT o.order_id, c.name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE o.customer_id = 12345;
```

This leverages the index for faster lookups and join operations.

Best Practices Summary

- Write modular, well-documented functions for repeatable analysis.
- Use expressive visualization libraries to explore and present data.
- Optimize queries by leveraging database indexes and minimizing unnecessary computations.
- Follow consistent naming conventions and code style for maintainability.

8 Integrating Programming in Data Science Projects

Integrating programming tools and environments is fundamental to executing robust, reproducible, and collaborative data science projects. Modern workflows combine interactive development environments (IDEs), workflow orchestration tools, and version control systems to streamline the journey from data ingestion to model deployment.

Workflow orchestration automates and coordinates the sequence of data-related tasks, such as extraction, transformation, analysis, and reporting. Tools like Apache Airflow, Prefect, Dagster, and Metaflow allow data scientists to define, schedule, and monitor complex pipelines as directed acyclic graphs (DAGs), ensuring that dependencies are respected and tasks are executed in the correct order [2, 17]. This orchestration reduces manual intervention, minimizes errors, and enables projects to scale efficiently across distributed systems or cloud platforms.

Reproducibility is a cornerstone of scientific data analysis. Interactive environments such as Jupyter Notebook and JupyterLab have become synonymous with reproducible research. They allow users to blend code, narrative text, and visualizations in a single document, making it easy to track the analytical process and share results with collaborators or reviewers [18]. RStudio offers similar capabilities for R users, supporting R Markdown for dynamic, self-contained reports. Both platforms encourage the use of project-oriented workflows, where all code, data, and outputs are organized in a consistent directory structure, facilitating seamless reruns and peer verification [19].

Collaboration is enhanced through integrated version control systems such as Git and platforms like GitHub or GitLab. These tools formalize the process of tracking code changes, managing branches, and merging contributions from multiple team members. Version control not only supports open science and transparency but also enables experimentation without risking the stability of production code. In multidisciplinary teams, version control is critical for coordinating Python, R, and SQL scripts, as well as documentation and configuration files.

Cross-language integration is increasingly common. Jupyter supports multiple kernels, allowing Python and R code to coexist in the same notebook, while RStudio's `reticulate` package enables R users to call Python functions directly. This interoperability lets teams leverage the strengths of both languages within a single project [20].

Best practices for integration include modularizing code, managing dependencies with tools like Conda or renv, and maintaining clear documentation. Automated testing and continuous integration pipelines further ensure that code remains functional as projects evolve.

In summary, integrating programming environments, workflow orchestration, and version control is essential for building scalable, reproducible, and collaborative data science solutions. These practices not only improve project efficiency but also enhance the credibility and impact of data-driven insights across research and industry.

"A reproducible, automated, and collaborative workflow is the hallmark of professional data science."

9 Exercises

Python/Pandas Tasks

1. **Handling Missing Data**
   Given a DataFrame `df` containing columns `customer_id`, `age`, and `purchase_amount`, write Python code using Pandas to:
   - Fill missing values in `age` with the column's median.
   - Drop rows where `customer_id` is missing.
2. **Removing Duplicates and Outliers**
   You have a DataFrame `sales` with columns `order_id`, `product`, and `amount`. Write Pandas code to:
   - Remove duplicate rows based on `order_id` and `product`.

- Remove outliers in the `amount` column using the Z-score method (i.e., keep only rows where the Z-score of `amount` is less than 3 in absolute value).

SQL Query Problem

3. **Joining Tables for Customer Orders**
   Given two tables: `customers` (`customer_id`, `name`) and `orders` (`order_id`, `customer_id`, `amount`), write a SQL query to return each customer's name and the total amount of their orders. Only include customers with at least one order.

Case Study: Hypothesis Testing in R

Case Study: Comparing Average Delivery Times

A logistics company wants to know if a new route reduces delivery times. You are given two vectors in R: `old_route = c(42, 45, 47, 43, 44, 46, 48)` and `new_route = c(39, 41, 40, 38, 42, 40, 39)`.

Tasks:

- Formulate the null and alternative hypotheses.
- Use an appropriate hypothesis test in R to compare the mean delivery times.
- Report the p-value and interpret the result at the 0.05 significance level.
- State your conclusion about the effectiveness of the new route.

References

[1] upGrad: Latest Trends in Data Science: Python, R, & SQL in 2025. Accessed: 2025-04-26. https://www.upgrad.com/blog/future-of-data-science-technology-in-india/

[2] DataCamp: Top 12 Programming Languages for Data Scientists in 2025. Accessed: 2025-04-26. https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022

[3] GUVI: Is Coding Required for Data Science? Accessed: 2025-04-26. https://www.guvi.in/blog/is-coding-required-for-data-science/

[4] Data, I.: Coding in Data Science: How Much Is Required? Accessed: 2025-04-26. https://www.institutedata.com/blog/coding-in-data-science-how-much-is-required/

[5] Analytics, N.: Exploring Reproducibility in Scientific Research and Data Science. Accessed: 2025-04-26. https://www.numberanalytics.com/blog/exploring-reproducibility-scientific-research-data-science

[6] Canada, S.: Production Level Code in Data Science. Accessed: 2025-04-26. https://www.statcan.gc.ca/en/data-science/network/production-level-code

[7] Academy, F.: 50 Python Libraries for Data Science in 2025. Accessed: 2025-04-26. https://www.fynd.academy/blog/python-libraries-for-data-science

[8] Coursera: Python Libraries for Data Science. Accessed: 2025-04-26. https://www.coursera.org/in/articles/python-libraries-for-data-science

[9] Wickham, H.: Ggplot2: Elegant Graphics for Data Analysis, 3rd edn. Springer, ??? (2020)

[10] Kuhn, M.: The caret package: A unified interface for predictive modeling. Journal of Statistical Software **105**(1), 1–30 (2023) https://doi.org/10.18637/jss.v105.i01

[11] Wickham, H., Grolemund, G.: R for Data Science: Tidyverse Principles. Accessed: 2025-04-26. https://r4ds.hadley.nz

[12] IBM: SQL Vs. NoSQL Databases: What's the Difference? Accessed: 2025-04-26. https://www.ibm.com/think/topics/sql-vs-nosql

[13] Smith, J., Lee, H.: Sql vs. nosql databases: Choosing the right option for fintech. SSRN (2020). SSRN 5112525

[14] PeerSpot: Cassandra Vs MongoDB Comparison. Accessed: 2025-04-26. https://www.peerspot.com/products/comparisons/cassandra_vs_mongodb

[15] Airbyte: Cassandra Vs. MongoDB: Navigating the NoSQL Landscape. Accessed: 2025-04-26. https://airbyte.com/data-engineering-resources/mongodb-vs-cassandra

[16] phoenixNAP: Cassandra Vs MongoDB - Key Differences. Accessed: 2025-04-26. https://phoenixnap.com/kb/cassandra-vs-mongodb

[17] AI, N.: Best Workflow and Pipeline Orchestration Tools. Accessed: 2025-04-26. https://neptune.ai/blog/best-workflow-and-pipeline-orchestration-tools

[18] Academy, D.: The Best IDEs and Tools for Data Science Projects. Accessed: 2025-04-26. https://www.dataskillacademy.com/post-detail/the-best-ides-and-tools-for-data-science-project/

[19] Project, E.: Reproducible Forecasting Workflows. Accessed: 2025-04-26. https://ecoforecast.org/reproducible-forecasting-workflows/

[20] Pickl.ai: How to Integrate Both Python & R Into Data Science Workflows. Accessed: 2025-04-26. https://www.pickl.ai/blog/python-r-into-data-science/

[21] KDnuggets: Data Cleaning with Pandas: Step-by-Step Tutorial. Accessed: 2025-04-26. https://www.kdnuggets.com/data-cleaning-with-pandas