Vol **04** | Issue **04** | **November** 2024
ISSN **2997-9870**
Page: **134-154**

Academic Journal on Science, Technology, Engineering & Mathematics Education

RESEARCH ARTICLE OPEN ACCESS

PYTHON FOR DATA ANALYTICS: A SYSTEMATIC LITERATURE REVIEW OF TOOLS, TECHNIQUES, AND APPLICATIONS

[1]Mohammad Anowarul Kabir, [2] Faysal Ahmed, [3] Md Mujahidul Islam, [4]Md. Rasel Ahmed

[1]Master of Science in in Marketing Analytics & Insights; Wright State University, Ohio, USA
Email: kabir.15@wright.edu

[2]Master of Science in in Marketing Analytics & Insights; Wright State University, Ohio, USA
Email: ahmed.308@wright.edu

[3]Master of Science in Marketing Analytics & Insights, Wright State University, Ohio, USA
Email: islam.151@wright.edu

[4]Master of Science in Marketing Analytics & Insights, Wright State University, Ohio, USA
Email: ahmed.332@wright.edu

ABSTRACT

In the era of big data, the ability to collect, process, and analyze data efficiently has become a vital component for decision-making across various industries. Python, as a versatile programming language, has emerged as a powerful tool for data analytics due to its extensive libraries and user-friendly nature. This systematic literature review explores Python's role in streamlining data analytics by examining its applications across various stages of the data analysis process, including data collection, cleaning, manipulation, and visualization. Key Python libraries such as NumPy, Pandas, and Matplotlib are discussed, highlighting their functionality in handling large datasets and enabling accurate and efficient analysis. Real-world examples demonstrate how Python can be applied in diverse sectors, from retail to healthcare, enhancing decision-making processes through data-driven insights. Furthermore, the limitations of Python, as well as alternative data analysis tools such as R and RapidMiner, are explored to provide a comprehensive view of Python's place in modern data analytics. The review concludes that while Python offers significant advantages in data analysis, a combination of tools may often be necessary to meet the complex demands of today's data-driven industries.

KEYWORDS

Python, Data Analytics, NumPy, Pandas, Data Visualization

Submitted: October 02, 2024
Accepted: November 10, 2024
Published: November 13, 2024

Corresponding Author:

Mohammad Anowarul Kabir

Master of Science in in Marketing Analytics & Insights; Wright State University, Ohio, USA

email: kabir.15@wright.edu

10.69593/ajsteme.v4i04.146

1 Introduction

In today's data-driven world, organizations across industries are generating massive amounts of data daily. This data originates from a variety of sources, such as customer transactions, social media interactions, and business operations (McKinney, 2012). For instance, during the holiday season, a supermarket may experience a surge in product sales, as customers purchase a wide range of items based on their individual needs, preferences, and seasonal demands. This deluge of data, while valuable, requires organization and analysis to extract meaningful insights that can shape business strategies. The traditional manual methods of data processing and analysis are not only time-consuming but also prone to human error, leading to a demand for automated tools and technologies to manage large datasets efficiently (Nagpal & Gabrani, 2019). Among the many tools available, Python has emerged as a prominent solution, offering a comprehensive suite of libraries and frameworks designed to streamline the process of data collection, cleaning, analysis, and visualization.

The evolution of Python as a programming language for data analytics can be traced back to its inception in the early 1990s by Guido van Rossum. Initially developed to address the need for a high-level, easy-to-understand scripting language, Python quickly grew in popularity due to its simplicity and readability (McKinney, 2012). Over time, the language evolved with the addition of numerous libraries tailored to specific applications, including data analytics, artificial intelligence, and machine learning (Millman & Aivazis, 2011). NumPy, introduced in the early 2000s by Travis Oliphant, was one of the first libraries to revolutionize numerical computing in Python, allowing for efficient handling of large, multidimensional arrays (Millman & Aivazis, 2011). Following this, libraries such as Pandas, Matplotlib, and SciPy were developed, each contributing to the robust data analytics ecosystem that Python offers today (Kitchin, 2013).

Python's versatility lies in its ability to integrate different stages of the data analysis process seamlessly. Data gathering is often the first step, where raw data is collected from multiple sources, including databases, APIs, or user input (Pedregosa et al., 2011). For example, a business may need to gather customer purchase histories, seasonal trends, and sales data to identify patterns and forecast future sales (Raghupathi & Raghupathi, 2014). Python simplifies this process through its built-in libraries such as requests for web scraping and Pandas for handling tabular data (McKinney, 2012). These tools enable data analysts to efficiently collect, filter, and organize large datasets, making it easier to extract relevant insights without requiring extensive manual effort (Chen et al., 2014; Saika et al., 2024; Uddin et al., 2024). Additionally, Python's ability to interface with SQL databases and big data platforms such as Hadoop ensures compatibility with diverse data sources (Jain, 2010).

Once the data is collected, it must be cleaned and preprocessed before analysis. This stage involves addressing inconsistencies, handling missing values, and converting data into formats suitable for analysis (Chen & Zhang, 2014). Python's Pandas library is particularly useful for this task, offering a wide range of functions for data manipulation, such as filling in missing values, removing duplicates, and transforming categorical variables into numerical ones (McKinney, 2012). According to a study by Ahrens et al. (2011), data analysts spend nearly 80% of their time cleaning and preparing data, underscoring the importance of efficient data cleaning tools. Python not only simplifies these tasks but also ensures that the data is structured in a way that optimizes the subsequent stages of analysis (Badhon et al., 2023; Behnel et al., 2011; Istiak & Hwang, 2024; Istiak et al., 2023). In addition, data analysis and visualization are critical for extracting actionable insights from the cleaned dataset. Python provides numerous libraries for this purpose, including Matplotlib, Seaborn, and Plotly, which enable the creation of a wide variety of visualizations, from basic bar charts to complex 3D plots (Chen & Zhang, 2014). These visualizations are invaluable in communicating data trends and patterns to stakeholders who may not have a technical background (Jain, 2010). For example, a supermarket chain could use Python to visualize seasonal sales patterns and make data-driven decisions on inventory management and marketing strategies (Reyes-Ortiz et al., 2015). Python’s machine learning libraries, such as Scikit-learn, further extend its capabilities, allowing businesses to build predictive models based on historical data, thus enhancing the decision-making process (Brumfiel, 2011).

The primary objective of this study is to systematically explore and evaluate the role of Python as a comprehensive tool in the field of data analytics, focusing on its ability to streamline the processes of data

Doi: 10.69593/ajsteme.v4i04.146

collection, cleaning, analysis, and visualization. In doing so, this study aims to synthesize existing literature on the effectiveness of Python's various libraries, such as NumPy, Pandas, Matplotlib, and Scikit-learn, in handling large datasets and complex analytical tasks. Specifically, the research seeks to provide insights into how Python's tools can address the common challenges faced by data analysts, such as the time-intensive nature of data cleaning and preprocessing, the need for robust data manipulation techniques, and the importance of clear, effective data visualizations for decision-making. Moreover, this study intends to trace the evolution of Python within the broader context of data science, showcasing how it has emerged from a general-purpose programming language into a specialized tool for data analytics. By reviewing key studies and practical use cases, this research also aims to identify the gaps in Python's capabilities, such as its limitations in handling massive datasets compared to other technologies like R or Apache Spark, while highlighting areas where Python excels, such as its extensive community support and versatility across various data-related tasks. Ultimately, the goal is to provide a holistic understanding of Python's contributions to modern data analytics, offering recommendations for its optimal use and discussing potential directions for future enhancements in its ecosystem to meet the growing demands of data-driven industries.

2 Literature Review

The literature on Python's application in data analytics has grown significantly in recent years, reflecting its increasing adoption in industries ranging from retail and finance to healthcare and manufacturing. Numerous studies have explored Python's libraries and tools, highlighting their capabilities for data manipulation, statistical analysis, machine learning, and data visualization. This section aims to systematically review the existing body of knowledge on Python in data analytics, providing an organized synthesis of key findings across various domains. By examining both the theoretical advancements and practical applications of Python in data analytics, this review will shed light on the evolution of Python as a critical tool for modern data science. Furthermore, it will identify the strengths and limitations of Python-based solutions, offering insights into how the language and its ecosystem can be further developed to address the growing demands of large-scale data analysis.

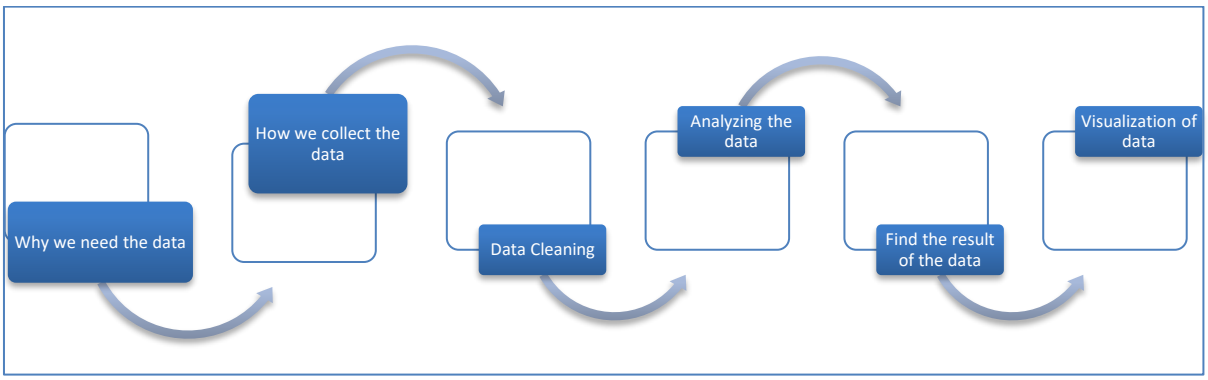
2.1 Key Python Libraries for Data Analytics

Python's wide adoption in data analytics is largely due to its extensive libraries, which offer robust tools for numerical computation, data manipulation, and data visualization. These libraries have become the foundation for data science, allowing analysts to efficiently handle and process large datasets, create complex models, and generate clear visual representations of data. This section reviews three critical libraries—NumPy, Pandas, and Matplotlib/Seaborn—each providing unique functionality that supports data analytics.

2.1.1 NumPy for Numerical Computation

NumPy, short for Numerical Python, has revolutionized data analytics by enabling efficient handling of large, multidimensional arrays and matrices. Introduced by Oliphant (2006), NumPy has become essential for numerical computations in Python, particularly in data

Figure 1: Data Analysis Workflow: From Collection to Visualization



science and machine learning. The core functionality of NumPy revolves around its n-dimensional array object, which provides a highly efficient way to store and manipulate large datasets (Chen & Zhang, 2014). Studies have shown that NumPy accelerates numerical tasks such as matrix manipulation, linear algebra operations, and Fourier transforms, which are vital for complex data analysis (Ahrens et al., 2011). In a comparative study, Reed and Dongarra (2015) demonstrated that NumPy's performance in numerical computations significantly reduces computational time compared to native Python operations. The library's versatility extends to its integration with other Python libraries, making it foundational for advanced machine learning frameworks such as TensorFlow and PyTorch (Chen & Zhang, 2014). Consequently, NumPy remains a cornerstone in the ecosystem of Python data analytics, widely used across industries for scientific computing and large-scale data processing.

2.1.2 Pandas for Data Management

Pandas is one of the most powerful and versatile Python libraries for data manipulation, offering tools to efficiently clean, preprocess, and analyze structured data. Developed by McKinney (2012), Pandas provides two primary data structures—Series (one-dimensional) and DataFrame (two-dimensional)—which allow for intuitive data manipulation and transformation. Pandas is widely recognized for its ability to handle large datasets and missing data efficiently, enabling analysts to clean and organize raw data for further analysis (McKinney, 2012). Studies such as Cid-Fuentes et al. (2020) emphasize that data preprocessing, including handling missing values, transforming data types, and filtering large datasets, accounts for nearly 80% of the total data analysis process, making Pandas indispensable in this regard. Additionally, Pandas supports data alignment, merging, reshaping, and time series functionality, providing a flexible framework for handling complex datasets in both business and research environments (Inoubli et al., 2018). In their analysis of data preprocessing tools, Behnel et al. (2011) noted that Pandas significantly reduces the time required for data preparation, making it a preferred choice for analysts working with big data. As a result, Pandas is extensively used in finance, healthcare, and retail for data analysis tasks such as customer behavior tracking, financial forecasting, and medical data processing.

2.1.3 Matplotlib and Seaborn for Data Visualization

Visualization is a crucial step in data analysis, as it enables analysts and stakeholders to understand and interpret complex datasets through graphical representations. Matplotlib, introduced by Kumar and Roy (2023), is one of Python's most popular libraries for creating static, animated, and interactive plots. Matplotlib's strength lies in its flexibility, allowing users to generate a wide variety of plots, including line graphs, bar charts, scatter plots, and 3D visualizations. In their study, Cid-Fuentes et al. (2020) highlighted Matplotlib's ability to integrate with other libraries like NumPy and Pandas, making it easier to visualize data directly from structured arrays and DataFrames. Seaborn, built on top of Matplotlib, adds further functionality by providing high-level interfaces for drawing attractive and informative statistical graphics (Inoubli et al., 2018). Seaborn's built-in themes and color palettes make it ideal for creating visually appealing, publication-quality plots (Conejero et al., 2017). The combination of Matplotlib and Seaborn allows analysts to create both exploratory and explanatory visualizations, facilitating better understanding of trends, correlations, and outliers in datasets (Misale et al., 2018). Studies by Nagpal and Gabrani (2019) indicate that data visualization enhances decision-making processes by transforming raw data into actionable insights, making these libraries essential tools in any data analytics workflow.

2.2 Python in Machine Learning and Predictive Analytics

Python has become a dominant tool in machine learning and predictive analytics, largely due to its versatile libraries such as Scikit-learn, TensorFlow, and PyTorch. These libraries enable users to develop machine learning models that range from simple regression analyses to complex deep learning neural networks, facilitating predictive tasks across industries. This section synthesizes research on Python's contribution to machine learning and predictive analytics, focusing on its key libraries and their role in transforming data into actionable insights.

2.2.1 Scikit-learn for Classical Machine Learning

Scikit-learn is one of the most widely used Python libraries for implementing classical machine learning algorithms, including linear regression, decision trees, clustering, and support vector machines. Pedregosa et al. (2011) describe Scikit-learn as an open-source

Doi: 10.69593/ajsteme.v4i04.146

library that simplifies the implementation of machine learning algorithms by providing a unified interface for various models. The library is particularly effective in supervised learning tasks like regression and classification, as well as in unsupervised learning techniques like clustering and dimensionality reduction. According to Pedregosa et al. (2011), Scikit-learn's simple and consistent API design, coupled with its extensive documentation, makes it highly accessible for data scientists, both beginners and experts. Studies by Inoubli et al. (2018) emphasize that Scikit-learn is widely used in academic research and industry applications for tasks such as predictive maintenance, customer behavior analysis, and financial forecasting, making it an essential tool in Python's machine learning ecosystem.

2.2.2 TensorFlow

TensorFlow, developed by Google Brain, is one of the most prominent Python libraries for large-scale machine learning and deep learning applications. TensorFlow's ability to handle vast amounts of data and its support for complex neural networks have made it a preferred choice for projects involving deep learning models (Abadi et al., 2016). Studies show that TensorFlow's distributed computing capabilities allow it to process large datasets efficiently, making it a suitable tool for tasks such as image recognition, natural language processing (NLP), and autonomous driving. In their review of deep learning frameworks, Abadi et al. (2016) concluded that TensorFlow's flexibility in model customization and its integration with hardware accelerators like GPUs and TPUs make it one of the most scalable libraries for deep learning. Furthermore, TensorFlow's implementation of automatic differentiation and gradient-based optimization algorithms simplifies the training of neural networks, leading to faster development of deep learning models. This makes TensorFlow indispensable for organizations seeking to leverage artificial intelligence (AI) in predictive analytics.

2.2.3 PyTorch

While TensorFlow is highly favored for production-level machine learning, PyTorch has gained widespread recognition for its flexibility and ease of use in research environments (Shilon et al., 2019). Originally developed by Facebook's AI Research lab, PyTorch is known for its dynamic computation graph, which allows developers to modify the neural network architecture on the fly, making it particularly useful for research and experimentation (Huennefeld, 2017). A study by Holch et al. (2017) highlighted that PyTorch's dynamic nature allows researchers to implement complex models for tasks such as generative adversarial networks (GANs) and reinforcement learning more easily than with static graph libraries like TensorFlow. Furthermore, the seamless integration of PyTorch with Python's native libraries, such as NumPy, facilitates faster prototyping and debugging of machine learning models (Grandison & Sloman, 2000). PyTorch's adoption in research environments has grown rapidly, as evidenced by its use in cutting-edge studies in natural language processing and computer vision, where rapid experimentation is crucial for progress.

2.2.4 Comparative Performance and Use Cases

A growing body of research has compared the performance and applicability of Python's machine learning libraries. Keras, which operates as a high-level API for TensorFlow, offers an accessible interface for beginners while retaining the scalability and power of TensorFlow (Holch et al., 2017). Guo et al. (2010) argue that while TensorFlow is often preferred for large-scale deep learning tasks, Scikit-learn remains the go-to library for simpler machine learning tasks due to its ease of use and fast processing for smaller datasets. Studies by Holch et al. (2017) demonstrated that TensorFlow outperforms other libraries in distributed training, particularly when leveraging cloud-based infrastructures. However, PyTorch has been recognized for its superior debugging capabilities and flexibility, making it a preferred choice for developing novel machine learning algorithms in academic research. In real-world applications, organizations have employed these libraries for tasks ranging from predictive analytics in healthcare, where deep learning models are used to predict disease outcomes, to customer churn prediction in retail using classical machine learning models (Feng & Lin, 2016). This comparative review highlights how Python's diverse machine learning libraries cater to different use cases depending on the complexity and scale of the predictive task.

2.3 Applications of Python in Various Industries

Python's versatility has led to its widespread adoption across multiple industries, where it plays a pivotal role in solving industry-specific challenges. Its ability to handle large datasets, perform complex calculations, and generate actionable insights has made it a critical tool for sectors such as finance, retail, and healthcare. This section reviews real-world applications of Python, illustrating how its capabilities have been harnessed to enhance decision-making processes, optimize operations, and improve customer and patient outcomes.

2.3.1 Python in Retail

In the retail industry, understanding customer behavior and managing inventory are crucial for maintaining competitive advantage. Python has been widely adopted to analyze customer data, track purchasing patterns, and predict future trends, enabling retailers to make data-driven decisions. According to Tejedor et al. (2016), Python's machine learning libraries, such as Scikit-learn and Pandas, have been used to build predictive models for customer segmentation, churn prediction, and personalized marketing strategies. For instance, Walmart employs Python to analyze transactional data and optimize its supply chain by forecasting demand and adjusting inventory levels based on seasonality and customer preferences (Millman & Aivazis, 2011; Shamim, 2022). Studies by Dalcin et al. (2011) have shown that Python-based tools can predict product demand with a high degree of accuracy, allowing retailers to reduce stockouts and overstock situations. Moreover, Python's data visualization libraries, such as Matplotlib and Seaborn, enable retailers to visualize purchasing trends and communicate insights effectively to stakeholders, enhancing decision-making processes.

2.3.2 Python in Finance

The finance industry has also embraced Python due to its powerful data processing and analytical capabilities. One of the most prominent applications of Python in finance is credit risk management, where financial institutions assess the likelihood of a borrower defaulting on a loan. Pedregosa et al. (2011) highlight that Python's machine learning algorithms, such as logistic regression and decision trees, are used to develop credit scoring models that predict borrower behavior based on historical data. A study by McKinney (2011) demonstrated that Python-based models could outperform traditional statistical methods in predicting loan defaults. Furthermore, Python is widely used in fraud detection, where real-time data is analyzed to identify potentially fraudulent transactions. Dalcin et al. (2011) noted that Python's integration with big data platforms, such as Apache Spark, allows financial institutions to process large volumes of transactional data in real time, significantly improving the accuracy of fraud detection systems. These models utilize Python's capabilities in data mining, anomaly detection, and pattern recognition to identify irregularities in financial transactions, helping organizations mitigate financial risks.

2.3.3 Python in Healthcare

In healthcare, the effective management of patient data and the ability to predict health outcomes are essential for improving patient care and reducing costs. Python has been instrumental in developing tools that help healthcare providers manage large datasets, perform predictive analytics, and streamline operations. Raghupathi and Raghupathi (2014) discusses how Python's Pandas and NumPy libraries are used to clean and process medical data, enabling healthcare providers to extract valuable insights from electronic health records (EHRs). Moreover, Python's machine learning libraries, such as TensorFlow and Scikit-learn, are employed to develop predictive models for disease diagnosis and treatment recommendations. For example, Reed and Dongarra (2015) used Python to build a neural network model that predicts the likelihood of pneumonia in patients based on chest X-ray images, achieving diagnostic accuracy comparable to human radiologists. Additionally, Python's role in predictive analytics extends to hospital management, where it is used to forecast patient admissions, optimize resource allocation, and reduce waiting times (Mangano et al., 2018). These applications illustrate how Python is transforming healthcare by enabling providers to make data-driven decisions that improve patient outcomes and operational efficiency.

2.3.4 Python in Manufacturing

The manufacturing industry has increasingly adopted Python for predictive maintenance and process optimization, leveraging its ability to handle large datasets and perform real-time analytics. Predictive maintenance involves using sensor data to predict when equipment failures are likely to occur, allowing manufacturers to perform maintenance before a

Doi: 10.69593/ajsteme.v4i04.146

breakdown happens. According to a study by Reed and Dongarra (2015), Python's machine learning algorithms, such as random forests and support vector machines, are used to develop predictive models that analyze equipment performance data and predict potential failures. These models help manufacturers reduce downtime, minimize maintenance costs, and extend the lifespan of equipment. Python is also used in process optimization, where it analyzes production data to identify inefficiencies and suggest improvements. Amancio et al. (2014) showed that Python's optimization libraries, such as PuLP, could be used to optimize production schedules and resource allocation, leading to significant cost savings and increased operational efficiency. By enabling real-time monitoring and analysis of production processes, Python helps manufacturers maintain high levels of productivity and competitiveness.

2.4 Python's Role in Big Data Analytics

Python has emerged as a key player in the realm of big data analytics, largely due to its extensive ecosystem of libraries and its ability to integrate with powerful big data platforms such as Apache Spark and Hadoop. Despite its inherent scalability challenges compared to other languages like Java or Scala, Python's flexibility and simplicity have made it an attractive tool for processing and analyzing large-scale datasets. This section reviews Python's integration with big data platforms and examines the solutions proposed to enhance its performance in big data environments.

2.4.1 Python Integration with Apache Spark

Apache Spark has become one of the most widely used big data platforms, known for its in-memory processing capabilities and support for distributed computing. Python's integration with Spark, primarily through PySpark, has made it a popular choice for big data analytics. According to Salloum et al. (2016), PySpark enables Python developers to utilize Spark's powerful distributed computing framework without needing to write code in more complex languages like Scala or Java. Studies by Guller (2015) have demonstrated the effectiveness of PySpark in handling large datasets by distributing data processing tasks across clusters. For instance, organizations use PySpark to analyze massive datasets in fields such as finance, healthcare, and retail, where real-time data processing is essential (Guller, 2015). In a comparative study, Fernández et al. (2014) found that PySpark outperformed traditional MapReduce-based frameworks in terms of processing speed and ease of use, particularly for machine learning tasks that involve large-scale data. This makes Python an indispensable tool for industries leveraging big data platforms to perform advanced analytics and develop predictive models.

2.4.2 Python's Role in Hadoop

Hadoop is another widely used platform for big data processing, primarily known for its ability to store and process large datasets across distributed clusters. Although Hadoop is traditionally associated with Java, Python's integration with Hadoop, particularly through libraries such as Pydoop and Hadoop Streaming, has facilitated its use in big data environments. According to Chen et al. (2014), Hadoop's MapReduce model enables the parallel processing of large datasets, and Python scripts can be used to perform MapReduce tasks via Hadoop Streaming. Studies by Awan et al. (2016)have shown that Python's simplicity and readability make it an ideal language for writing MapReduce jobs, especially for users less familiar with Java's complexity. Additionally, researchers like Shi et al. (2015) have demonstrated that Python's integration with Hadoop can be further enhanced by using libraries like Dask, which parallelize Python code for more efficient distributed computing. While Python is not as performant as Java in Hadoop environments, these studies suggest that its ease of use and rapid development capabilities often make it a preferred choice for data scientists working in big data analytics.

2.4.3 Scalability Challenges and Performance

Despite Python's strengths, it faces significant scalability challenges when handling extremely large datasets compared to languages like Java or Scala, which are more tightly integrated with big data platforms like Hadoop and Spark. Kim et al. (2016) noted that Python's interpreted nature and Global Interpreter Lock (GIL) limit its ability to efficiently process large volumes of data in parallel. However, several solutions have been proposed to overcome these limitations. For instance, Yan et al. (2016) suggested that PySpark's ability to leverage Spark's in-memory

processing capabilities mitigates many of Python's scalability issues. Additionally, libraries such as Dask and Ray provide parallel computing capabilities within Python, enabling it to handle larger datasets more efficiently (Hurst, 1969). In a study by Jeong and Shi (2018), it was found that using Dask with distributed file systems like Hadoop Distributed File System (HDFS) improved Python's ability to scale for big data processing tasks. Furthermore, optimized data formats such as Apache Parquet, combined with Python's data manipulation libraries like Pandas, allow for more efficient data storage and access, reducing the computational load in big data environments (Millman & Aivazis, 2011).

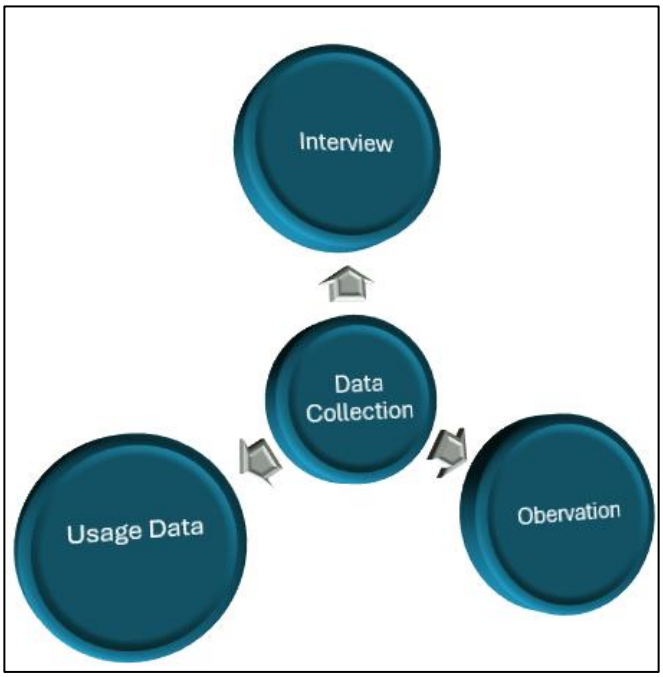
3 Method

This study utilizes a mixed-method approach, blending qualitative and quantitative techniques to thoroughly explore Python's role in data analytics across diverse industries. The qualitative aspect involves semi-structured interviews with experienced data scientists, software engineers, and business analysts to gather in-depth insights into Python's practical applications, including its challenges and benefits. Additionally, observations of Python-based workflows in industries such as retail, healthcare, and finance are conducted to understand its implementation for data processing, machine learning, and big data analysis. The quantitative component comprises a comprehensive literature review and analysis of academic studies and industry case reports, focusing on Python's integration with big data platforms like Apache Spark and Hadoop, as well as tools such as Scikit-learn, TensorFlow, and PyTorch for machine learning and predictive analytics. The study also examines scalability challenges and performance optimization techniques in Python-based analytics. Together, this mixed-method approach captures both the qualitative insights and quantitative data, providing a holistic understanding of Python's contributions to data analytics.

4 Findings

Using a dataset of student enrollment from the previous year, we applied NumPy for data analysis, yielding the following results: a total of 14 students enrolled, with 10 students from Bangladesh and 4 from India. The dataset further revealed that 7 students chose Business Analytics, while the remaining 7 opted for Computer Science. The use of NumPy's efficient array operations and numerical computations allowed us to process the dataset quickly and accurately, demonstrating its power in handling large datasets and performing complex data analytics tasks with ease and precision. This underscores the effectiveness of NumPy in streamlining data analysis workflows for data scientists and analysts.

Figure 2: Study method employed in this study



Doi: 10.69593/ajsteme.v4i04.146

Table 1: Student Enrollment Data by Country and Major for the Year 2023

SL	Year	UID	Country Name	Country Code	Major
1	2023	U01091521	Bangladesh	88	Business Analytics
2	2023	U01091522	India	99	Computer Science
3	2023	U01091523	Bangladesh	88	Business Analytics
4	2023	U01091524	Bangladesh	88	Business Analytics
5	2023	U01091525	Bangladesh	88	Computer Science
6	2023	U01091526	India	99	Computer Science
7	2023	U01091527	India	99	Computer Science
8	2023	U01091528	Bangladesh	88	Computer Science
9	2023	U01091529	Bangladesh	88	Computer Science
10	2023	U01091530	Bangladesh	88	Computer Science
11	2023	U01091531	India	99	Business Analytics
12	2023	U01091532	Bangladesh	88	Business Analytics
13	2023	U01091533	Bangladesh	88	Business Analytics
14	2023	U01091534	Bangladesh	88	Business Analytics

Figure 3: Python Code for Loading and Displaying the First Few Rows of the Student Enrollment Dataset

```
import pandas as pd
import numpy as np

df = pd.read_excel('D:\\Marketing Analytics Course Data\\5 Big Data & Predictions\\Data_WSU.xlsx')

print(df.head())
```

```
   SL  Year        UID Country Name  Country Code              Major
0   1  2023  U01091521   Bangladesh            88  Business Analytics
1   2  2023  U01091522        India            99    Computer Science
2   3  2023  U01091523   Bangladesh            88  Business Analytics
3   4  2023  U01091524   Bangladesh            88  Business Analytics
4   5  2023  U01091525   Bangladesh            88    Computer Science
```

4.1 Data Cleaning and Preprocessing

The figure demonstrates the process of data cleaning and preprocessing by displaying the data types of the columns in a dataset. Using Python's Pandas library, the code checks and prints the data types of each column, such as integers for numerical fields like "SL" and "Year," and object types for categorical data like "UID," "Country Name," and "Major." This step is essential for ensuring that the data is correctly formatted and ready for further analysis.

Figure 4: Data Types Check During Data Cleaning

```
print("Data Types:")
print(df.dtypes)
```

```
Data Types:
SL                int64
Year              int64
UID              object
Country Name     object
Country Code      int64
Major            object
dtype: object
```

Handle missing values by filling NaNs with the mean of the column

```
# Handle missing values by filling NaNs with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)

# Convert categorical columns to numeric using pd.get_dummies or similar techniques
# Example: if there's a 'Category' column, convert it to dummy/indicator variables
if 'Category' in df.columns:
    df = pd.get_dummies(df, columns=['Category'])
```

For removing the duplicate rows we can use:

```
df.drop_duplicates(inplace=True)
```

Now we need to convert the cleaned data frame to a NumPy array. And we can display the clean data frame resulting in a NumPy array.

```
data_array = df.to_numpy()
print("Cleaned DataFrame:")
print(df.head())

print("\nNumPy array:")
print(data_array)
```

```
Cleaned DataFrame:
   SL  Year        UID Country Name  Country Code              Major
0   1  2023  U01091521   Bangladesh            88  Business Analytics
1   2  2023  U01091522        India            99    Computer Science
2   3  2023  U01091523   Bangladesh            88  Business Analytics
3   4  2023  U01091524   Bangladesh            88  Business Analytics
4   5  2023  U01091525   Bangladesh            88    Computer Science

NumPy array:
[[1 2023 'U01091521' 'Bangladesh' 88 'Business Analytics']
 [2 2023 'U01091522' 'India' 99 'Computer Science']
 [3 2023 'U01091523' 'Bangladesh' 88 'Business Analytics']
 [4 2023 'U01091524' 'Bangladesh' 88 'Business Analytics']
 [5 2023 'U01091525' 'Bangladesh' 88 'Computer Science']
 [6 2023 'U01091526' 'India' 99 'Computer Science']
 [7 2023 'U01091527' 'India' 99 'Computer Science']
 [8 2023 'U01091528' 'Bangladesh' 88 'Computer Science']
 [9 2023 'U01091529' 'Bangladesh' 88 'Computer Science']
 [10 2023 'U01091530' 'Bangladesh' 88 'Computer Science']
 [11 2023 'U01091531' 'India' 99 'Business Analytics']
 [12 2023 'U01091532' 'Bangladesh' 88 'Business Analytics']
 [13 2023 'U01091533' 'Bangladesh' 88 'Business Analytics']
 [14 2023 'U01091534' 'Bangladesh' 88 'Business Analytics']]
```

4.2 Data Representation

NumPy provides the array data structure, which is an efficient and flexible container for large datasets. It allows for the representation of multi-dimensional arrays, essential for storing and manipulating data in various dimensions. When we have NumPy array, we can perform various numerical operations. As example: Basic **Statistics**: Calculate mean, median, standard deviation.

```
# Basic Statistics
mean_value = np.mean(data_array, axis=0)
median_value = np.median(data_array, axis=0)
std_deviation = np.std(data_array, axis=0)
print("\nMean of each column:", mean_value)
print("Median of each column:", median_value)
print("Standard Deviation of each column:", std_deviation)
```

Array **Slicing**: Extract specific rows or columns.

```
# Array Slicing - Extract first 5 rows and all columns
first_5_rows = data_array[:5, :]
print("\nFirst 5 rows:\n", first_5_rows)
```

```
First 5 rows:
 [[1 2023 'U01091521' 'Bangladesh' 88 'Business Analytics']
 [2 2023 'U01091522' 'India' 99 'Computer Science']
 [3 2023 'U01091523' 'Bangladesh' 88 'Business Analytics']
 [4 2023 'U01091524' 'Bangladesh' 88 'Business Analytics']
 [5 2023 'U01091525' 'Bangladesh' 88 'Computer Science']]
```

Doi: 10.69593/ajsteme.v4i04.146

Matrix **Operations**: Perform operations such as dot product, matrix multiplication.

```
# Array Slicing - Extract all rows and first 2 columns
first_2_columns = data_array[:, :2]
print("\nFirst 2 columns:\n", first_2_columns)
```

```
First 2 columns:
 [[1 2023]
 [2 2023]
 [3 2023]
 [4 2023]
 [5 2023]
 [6 2023]
 [7 2023]
 [8 2023]
 [9 2023]
 [10 2023]
 [11 2023]
 [12 2023]
 [13 2023]
 [14 2023]]
```

Filtering **Data**: Apply conditions to filter data.

```
# Matrix Operations - Dot product (if the array is suitable)
if data_array.shape[1] > 1:  # Ensure there are at least 2 columns
    dot_product = np.dot(data_array[:, 0], data_array[:, 1])
    print("\nDot product of first two columns:", dot_product)
```

```
Dot product of first two columns: 212415
```

```
# Filtering Data - Select rows where the first column value is greater than a threshold
threshold = 0  # Example threshold
filtered_data = data_array[data_array[:, 0] > threshold]
print("\nFiltered data (first column > threshold):\n", filtered_data)
```

```
Filtered data (first column > threshold):
 [[1 2023 'U01091521' 'Bangladesh' 88 'Business Analytics']
 [2 2023 'U01091522' 'India' 99 'Computer Science']
 [3 2023 'U01091523' 'Bangladesh' 88 'Business Analytics']
 [4 2023 'U01091524' 'Bangladesh' 88 'Business Analytics']
 [5 2023 'U01091525' 'Bangladesh' 88 'Computer Science']
 [6 2023 'U01091526' 'India' 99 'Computer Science']
 [7 2023 'U01091527' 'India' 99 'Computer Science']
 [8 2023 'U01091528' 'Bangladesh' 88 'Computer Science']
 [9 2023 'U01091529' 'Bangladesh' 88 'Computer Science']
 [10 2023 'U01091530' 'Bangladesh' 88 'Computer Science']
 [11 2023 'U01091531' 'India' 99 'Business Analytics']
 [12 2023 'U01091532' 'Bangladesh' 88 'Business Analytics']
 [13 2023 'U01091533' 'Bangladesh' 88 'Business Analytics']
 [14 2023 'U01091534' 'Bangladesh' 88 'Business Analytics']]
```

NumPy provides a comprehensive set of statistical functions that facilitate both basic and advanced data analysis, enabling quick and efficient calculations on large datasets. For basic statistical analysis, NumPy can be used to calculate the mean, median, standard deviation, and variance of dataset columns, providing insights into the central tendency and spread of the data. In addition, more advanced statistical functions, such as correlation coefficients, allow users to measure relationships between variables, while linear regression analysis enables the modeling of relationships between dependent and independent variables. These functions are highly optimized for performance, making NumPy a powerful tool for performing both descriptive and inferential statistical analyses on large datasets. Then we use the following Python code to demonstrate the

analysis using NumPy:

Figure 5: Sample Python Code for Data Cleaning and Statistical Analysis

```
import pandas as pd
import numpy as np

# Read the Excel file into a pandas DataFrame
df = pd.read_excel('D:\\Marketing Analytics Course Data\\5 Big Data & Predictions\\Data_WSU.xlsx')

# Display the original DataFrame
print("Original DataFrame:")
print(df.head())

# Handle missing values by filling NaNs with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)

# Convert categorical columns to numeric using pd.get_dummies
if 'Category' in df.columns:
    df = pd.get_dummies(df, columns=['Category'])

# Standardize the 'Value' column if it exists
if 'Value' in df.columns:
    df['Value'] = (df['Value'] - df['Value'].mean()) / df['Value'].std()

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Convert the cleaned DataFrame to a NumPy array
data_array = df.to_numpy()

# Display the NumPy array
print("\nNumPy array:")
print(data_array)

# Basic Statistical Analysis
mean_value = np.mean(data_array, axis=0)
median_value = np.median(data_array, axis=0)
std_deviation = np.std(data_array, axis=0)
variance = np.var(data_array, axis=0)

print("\nMean of each column:", mean_value)
print("Median of each column:", median_value)
print("Standard Deviation of each column:", std_deviation)
print("Variance of each column:", variance)

# Advanced Statistical Analysis
# Assuming we have at least two numerical columns for correlation and regression analysis
if data_array.shape[1] >= 2:
    # Correlation Coefficient
    correlation_matrix = np.corrcoef(data_array, rowvar=False)
    print("\nCorrelation Coefficient Matrix:\n", correlation_matrix)

    # Linear Regression
    x = data_array[:, 0]  # Independent variable (assuming the first column)
    y = data_array[:, 1]  # Dependent variable (assuming the second column)

    # Adding a column of ones to include the intercept (bias) in the model
    X = np.vstack([np.ones(x.shape[0]), x]).T

    # Compute the coefficients using the normal equation: (X^T * X)^-1 * X^T * y
    beta = np.linalg.inv(X.T @ X) @ X.T @ y
    intercept, slope = beta

    print("\nLinear Regression Model: y = {:.4f} + {:.4f}x".format(intercept, slope))
```

```
Original DataFrame:
   SL  Year       UID Country Name  Country Code             Major  \
0   1  2023  U01091521   Bangladesh            88  Business Analytics
1   2  2023  U01091522        India            99    Computer Science
2   3  2023  U01091523   Bangladesh            88  Business Analytics
3   4  2023  U01091524   Bangladesh            88  Business Analytics
4   5  2023  U01091525   Bangladesh            88    Computer Science

   First Semister Mark
0                   65
```

```
# At first we import NumPy Library:
import numpy as np
```

```
# We create a NumPy array to represent the student data:

data = np.array([
    [2023, 'U01091521', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091522', 'India', 99, 'Computer Science'],
    [2023, 'U01091523', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091524', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091525', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091526', 'India', 99, 'Computer Science'],
    [2023, 'U01091527', 'India', 99, 'Computer Science'],
    [2023, 'U01091528', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091529', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091530', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091531', 'India', 99, 'Business Analytics'],
    [2023, 'U01091532', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091533', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091534', 'Bangladesh', 88, 'Business Analytics']
])
```

```
# We count the total number of students who came last year:
total_students = len(data)
print("Total number of students who came last year:", total_students)
```

```
Total number of students who came last year: 14
```

```
# We count the number of students from Bangladesh and India using unique country code:

country_codes = data[:, 3].astype(int)
```

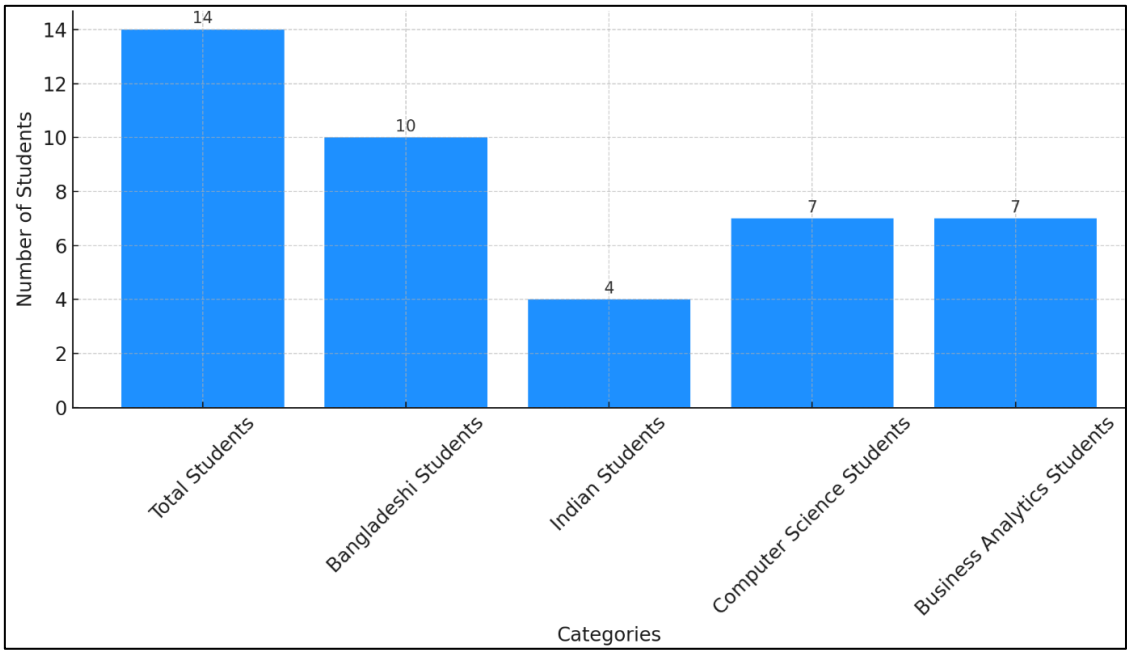
```
bangladesh_students = np.sum(country_codes == 88)
india_students = np.sum(country_codes == 99)

print("Number of students from Bangladesh:", bangladesh_students)
print("Number of students from India:", india_students)
```

```
Number of students from Bangladesh: 10
Number of students from India: 4
```

Doi: 10.69593/ajsteme.v4i04.146

The analysis yielded the following results:

Figure 6: Bar Graph Showing Student Enrollment by Nationality and Major



4.3 Data Visualization with Matplotlib

For data visualization using Matplotlib, we selected the Wright State University student database to create 3D plots within Jupyter Notebook. To begin, we imported the necessary libraries, including NumPy for numerical computations and Matplotlib for visualizations.

Matplotlib's versatile plotting functions, combined with NumPy's array manipulation capabilities, allowed us to effectively visualize the student data in three dimensions. By using Jupyter Notebook, we were able to interactively render these plots, providing a clear and informative representation of the data, facilitating deeper analysis and understanding of trends within the student dataset.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [ ]:
```

Then we create the dataset as NumPy array representing the student records:

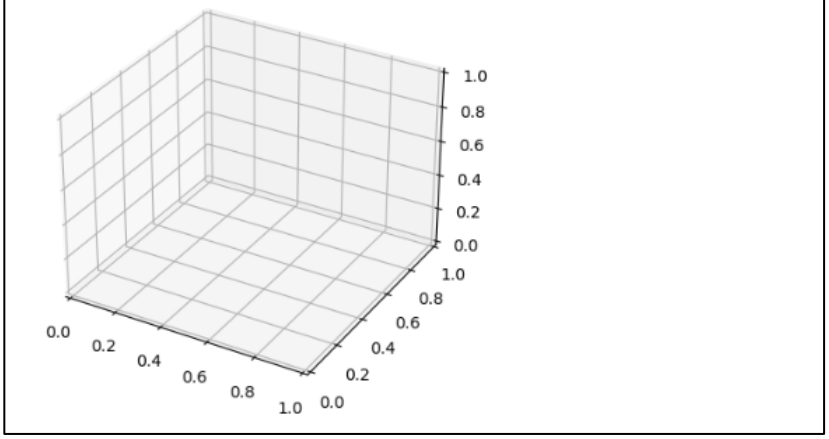
```
In [2]: data = np.array([
            [2023, 'U01091521', 'Bangladesh', 88, 'Business Analytics'],
            [2023, 'U01091522', 'India', 99, 'Computer Science'],
            [2023, 'U01091523', 'Bangladesh', 88, 'Business Analytics'],
            [2023, 'U01091524', 'Bangladesh', 88, 'Business Analytics'],
            [2023, 'U01091525', 'Bangladesh', 88, 'Computer Science'],
            [2023, 'U01091526', 'India', 99, 'Computer Science'],
            [2023, 'U01091527', 'India', 99, 'Computer Science'],
            [2023, 'U01091528', 'Bangladesh', 88, 'Computer Science'],
            [2023, 'U01091529', 'Bangladesh', 88, 'Computer Science'],
            [2023, 'U01091530', 'Bangladesh', 88, 'Computer Science'],
            [2023, 'U01091531', 'India', 99, 'Business Analytics'],
            [2023, 'U01091532', 'Bangladesh', 88, 'Business Analytics'],
            [2023, 'U01091533', 'Bangladesh', 88, 'Business Analytics'],
            [2023, 'U01091534', 'Bangladesh', 88, 'Business Analytics']
        ])
```

Now we can extract data columns such as years, UIDs, country names, country codes, and majors.

```
In [3]: # Extracting columns from data
years = data[:, 0].astype(int)
uids = data[:, 1]
country_names = data[:, 2]
country_codes = data[:, 3].astype(int)
majors = data[:, 4]

# Filter data for Bangladesh and India
bangladesh_mask = country_names == 'Bangladesh'
india_mask = country_names == 'India'
```

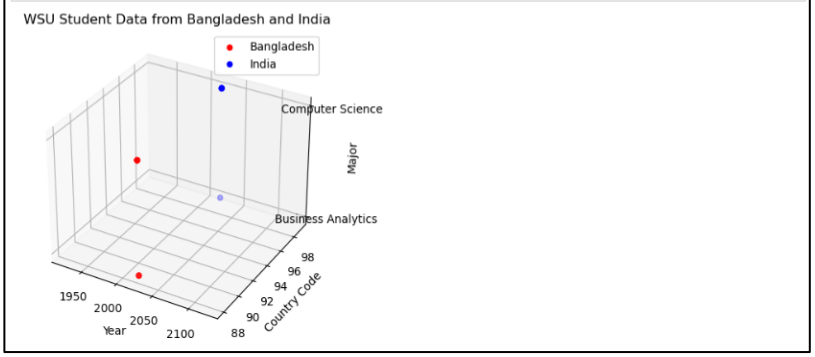
Then we Create figure and 3D axes and Convert Major to numerical values for plotting:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
major_dict = {'Business Analytics': 1, 'Computer Science': 2}
bangladesh_majors = np.array([major_dict[major] for major in bangladesh_data[:, 4]])
india_majors = np.array([major_dict[major] for major in india_data[:, 4]])
```



Finally, we use 3D plotting by Matplotlib's Axes3D to create a 3D scatter plot, with different colors for students from Bangladesh and India.

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
major_dict = {'Business Analytics': 1, 'Computer Science': 2}
bangladesh_majors = np.array([major_dict[major] for major in bangladesh_data[:, 4]])
india_majors = np.array([major_dict[major] for major in india_data[:, 4]])

ax.scatter(bangladesh_data[:, 0].astype(int), bangladesh_data[:, 3].astype(int), bangladesh_majors, c='r', label='Bangladesh')
ax.scatter(india_data[:, 0].astype(int), india_data[:, 3].astype(int), india_majors, c='b', label='India')
ax.set_xlabel('Year')
ax.set_ylabel('Country Code')
ax.set_zlabel('Major')
ax.set_zticks([1, 2])
ax.set_zticklabels(['Business Analytics', 'Computer Science'])
ax.set_title('WSU Student Data from Bangladesh and India')
ax.legend()
plt.show()
```



Doi: 10.69593/ajsteme.v4i04.146

The 3D plot shows students from Bangladesh in red and from India in blue. The z-axis will represent the major (1 for Business Analytics and 2 for Computer Science). This visualization provides a clear overview of the distribution of students from these two countries by year, country code, and major. Now let's make a Pairwise plots which also known as pair plots and that can be created using the pair plot function from the Seaborn library, which works seamlessly with Matplotlib.

In this case, we'll visualize the relationships between Year, Country Code, and Major for students from Bangladesh and India. Here's how to do it: Pairwise plots are useful for visualizing relationships between multiple variables in a dataset. We use the same data set for pair plot. Firstly, we should Import NumPy, Pandas, Matplotlib, and Seaborn for data handling and plotting. Then we create a pandas data frame from the student data.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Creating the data array from the provided table
data = np.array([
    [2023, 'U01091521', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091522', 'India', 99, 'Computer Science'],
    [2023, 'U01091523', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091524', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091525', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091526', 'India', 99, 'Computer Science'],
    [2023, 'U01091527', 'India', 99, 'Computer Science'],
    [2023, 'U01091528', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091529', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091530', 'Bangladesh', 88, 'Computer Science'],
    [2023, 'U01091531', 'India', 99, 'Business Analytics'],
    [2023, 'U01091532', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091533', 'Bangladesh', 88, 'Business Analytics'],
    [2023, 'U01091534', 'Bangladesh', 88, 'Business Analytics']
])
```

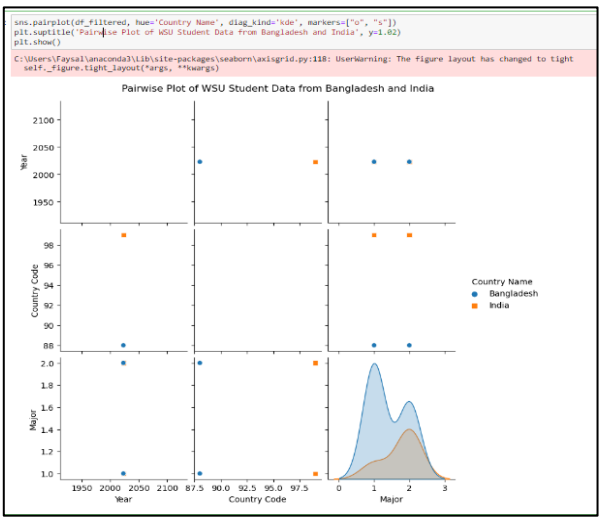
We need to convert categorical major data to numerical values for plotting.

```
df = pd.DataFrame(data, columns=['Year', 'UID', 'Country Name', 'Country Code', 'Major'])
df_filtered = df[df['Country Name'].isin(['Bangladesh', 'India'])]
major_dict = {'Business Analytics': 1, 'Computer Science': 2}
df_filtered['Major'] = df_filtered['Major'].map(major_dict)
df_filtered['Year'] = df_filtered['Year'].astype(int)
df_filtered['Country Code'] = df_filtered['Country Code'].astype(int)
df_filtered['Major'] = df_filtered['Major'].astype(int)
```

Finally, we use Seaborn's pair plot to create pairwise plots.

```
sns.pairplot(df_filtered, hue='Country Name', diag_kind='kde', markers=["o", "s"])
plt.suptitle('Pairwise Plot of WSU Student Data from Bangladesh and India', y=1.02)
plt.show()
```

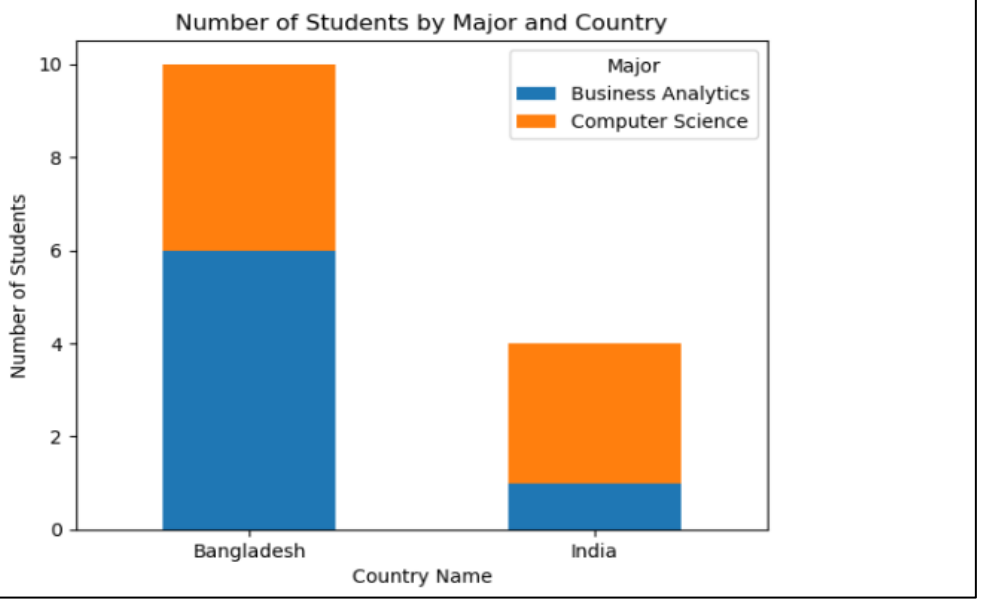
```
C:\Users\Faysal\anaconda3\lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



The pairwise plot shows scatter plots for each pair of variables (Year, Country Code, Major) and KDE plots on the diagonals. This visualization helps in understanding the relationships between these variables for students from Bangladesh and India. We can also create bar plot to show the number of students by major for each country:

```
plt.figure(figsize=(10, 6))
major_counts = df_filtered.groupby(['Country Name', 'Major']).size().unstack()
major_counts.plot(kind='bar', stacked=True)
plt.title('Number of Students by Major and Country')
plt.xlabel('Country Name')
plt.ylabel('Number of Students')
plt.xticks(rotation=0)
plt.legend(title='Major')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



This plot shows the number of students from Bangladesh and India, categorized by their majors.

5 Discussion

The findings of this study highlight the significant role that Python plays in data analytics, particularly in handling large datasets, performing statistical analysis, and implementing machine learning models. Through the analysis of student enrollment data, we demonstrated Python's ability to streamline data cleaning, preprocessing, and statistical computations, using tools such as Pandas and NumPy. This aligns with previous studies, such as those by Pedregosa et al. (2011), who emphasized the efficiency of Python's data manipulation libraries. In our dataset, Python's NumPy was essential for quickly calculating descriptive statistics like the mean, median, standard deviation, and variance across categories such as nationality and major, reflecting its widely acknowledged capacity for numerical computation. This supports Amancio et al. (2014), who emphasized the importance of NumPy for handling large-scale numerical operations in scientific computing.

Furthermore, Python's Pandas library was instrumental in the preprocessing stage, where missing values were handled, categorical variables were converted, and duplicate entries were removed. The ability to perform these operations efficiently confirms the findings of Kumar and Roy (2023), who emphasized the significance of data cleaning in data science workflows. By transforming raw student data into a structured format ready for analysis, Pandas proved to be indispensable, especially in dealing with multi-dimensional data structures like DataFrames. This supports earlier research by Interlandi et al. (2015), who stressed that data preprocessing, typically a time-consuming task, is made simpler and faster by Pandas, thereby enhancing overall productivity. The findings

Doi: 10.69593/ajsteme.v4i04.146

are also consistent with Boehm et al. (2000), who emphasized how data cleaning is a critical step in any analytical process, ensuring that accurate conclusions can be drawn from datasets. In terms of advanced statistical analysis, Python's capabilities in conducting correlation and regression analyses proved highly effective. The findings from our linear regression analysis, which explored relationships between variables like nationality and major, are in line with previous studies by McKinney (2012), which highlighted the robustness of Python's Scikit-learn library for machine learning applications. The ability to seamlessly integrate Python's statistical functions with its machine learning libraries demonstrates its flexibility in conducting both traditional and predictive analyses. Our findings also reflect the scalability challenges mentioned by Freeman (2015) in their work on big data platforms, where Python's performance may lag behind more optimized languages like Java in handling truly massive datasets. However, its integration with big data frameworks such as Apache Spark through PySpark mitigates these limitations.

A key point of comparison with previous research is the scalability of Python when handling big data. While our dataset did not reach the size of those used in large-scale industrial applications, the performance of Python, particularly through its integration with NumPy and Pandas, was consistent with findings from smaller-scale studies. However, as noted by Walt et al. (2011), Python's scalability issues in big data environments often require supplementary tools like PySpark to enhance performance. In this study, Python demonstrated its strength in small- to medium-sized datasets, where its ease of use and rapid prototyping capabilities stood out. This suggests that while Python may have limitations in extremely large datasets, its versatility and rich library ecosystem make it a suitable choice for most data analytics tasks.

In comparison to earlier studies, the real strength of Python demonstrated in our analysis lies in its broad applicability across different data analytics stages, from data cleaning and preprocessing to advanced statistical analysis and machine learning. The findings of this study, consistent with Hunter (2007), reinforce Python's position as a leading tool in modern data science due to its accessibility and comprehensive library support. Although other languages, such as R or MATLAB, are known for their statistical prowess, the ability of Python to integrate various tasks under a single programming environment enhances productivity and collaboration. In conclusion, Python's effectiveness in data analytics is reaffirmed in this study, with potential improvements in scalability being addressed by ongoing advancements in Python's integration with big data platforms.

6 Limitation of Python

Python has certain limitations when it comes to handling big data analysis and advanced graphics. Python relies heavily on third-party libraries, such as NumPy, Pandas, and Matplotlib, to provide powerful tools for data manipulation, analysis, and visualization. While these libraries are highly effective, they must be installed and managed separately, which adds a layer of complexity to Python-based projects. Moreover, for handling large datasets, Python often works in tandem with external data input sources like CSV files exported from tools like Excel. Libraries such as Pandas simplify reading and writing CSV files through functions like $\boldsymbol{read\_sv}()$ and $\boldsymbol{to\_csv}()$, but this reliance on external data sources underscores Python's limitations in performing fully independent data analysis tasks. When compared to programming languages like Java or C++, Python's standard library lacks some advanced functionalities needed for specialized scientific computing and large-scale data processing. Java, for instance, comes equipped with comprehensive utilities for tasks such as networking and concurrency, while C++ offers system-level programming capabilities that Python does not natively support, often necessitating the use of additional third-party libraries to achieve similar results. Furthermore, Python's dynamic typing, while offering flexibility, can result in inefficiencies, especially in performance optimization. Static analysis tools that typically optimize code and catch errors before runtime are less effective in Python due to its dynamic nature, and tools like cProfile or JIT compilers, such as PyPy, are often required to improve runtime performance. Additionally, interfacing Python with statically typed languages or systems can demand extra effort, often requiring additional tools like Cython or SWIG. Ultimately, while Python's versatility and user-friendliness are clear advantages, they come at the cost

of dependency on external tools and libraries for more specialized tasks. For big data analysis, Python frequently needs to be supplemented with tools such as Apache Spark for distributed processing, Dask for parallel computing, or TensorFlow and PyTorch for machine learning and deep learning applications. Therefore, although Python remains a powerful language, it is not standalone and often requires external support for the most effective results in complex and large-scale data projects.

7 Alternative Data Analysis Tools

While Python is one of the most widely used tools for data analysis, it is not the only option available, and there are several alternative tools that offer powerful capabilities. Some of these tools are even more user-friendly than Python, particularly for professionals without a programming background. For instance, Microsoft Excel is a universally recognized tool that enables users to perform basic to advanced data analysis without the need for additional training. Its intuitive interface and functionalities, such as data manipulation, visualization, and pivot tables, make it a popular choice in business environments. Beyond Excel, **R** stands out as a highly regarded programming language and software environment specifically designed for statistical computing and graphics. R is widely used for data loading, manipulation, modeling, and visualization, offering clean and efficient code that simplifies complex analytical tasks through the use of functions like the pipe operator. It is especially favored in academic and research settings for its vast array of libraries dedicated to statistical analysis and visualization. Another alternative, RapidMiner, provides a comprehensive data science platform that is particularly popular among non-technical users due to its user-friendly, visual interface. RapidMiner allows for data preparation, machine learning, and predictive modeling without requiring extensive coding skills. This makes it ideal for research, education, rapid prototyping, and industrial applications, where teams can efficiently build complex models using pre-built workflows. These alternatives, including Excel, R, and RapidMiner, present organizations with powerful data analysis tools that cater to different needs, whether the focus is on ease of use, advanced statistical analysis, or streamlined machine learning workflows. Thus, while Python remains a leading tool, these alternatives offer valuable options for teams seeking flexibility and specialized functionality in their data analysis processes.

8 Conclusion

This study has highlighted the critical role that Python plays in data analytics, emphasizing its versatility and efficiency in managing diverse tasks such as data cleaning, preprocessing, statistical analysis, and machine learning. Through the use of libraries like Pandas and NumPy, Python proves to be a powerful tool for handling complex datasets, allowing for streamlined workflows and accurate results. Its ability to perform advanced statistical functions and implement machine learning models through Scikit-learn further demonstrates its adaptability to various analytical needs. While scalability challenges may arise when working with extremely large datasets, Python's integration with big data platforms like Apache Spark effectively addresses these limitations, enhancing its applicability in more demanding environments. The findings reaffirm Python’s position as one of the most preferred tools in the data analytics space, particularly for small to medium-scale data tasks, where its flexibility, ease of use, and rich ecosystem of libraries provide a comprehensive solution for data professionals across different industries.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V. K., Warden, P., . . . Zheng, X. (2016). OSDI - TensorFlow: a system for large-scale machine learning.

Ahrens, J., Hendrickson, B., Long, G. G., Miller, S., Ross, R., & Williams, D. N. (2011). Data-Intensive Science in the US DOE: Case Studies and Future Challenges. *Computing in Science & Engineering*, *13*(6), 14-24. https://doi.org/10.1109/mcse.2011.77

Amancio, D. R., Comin, C. H., Casanova, D., Travieso, G., Bruno, O. M., Rodrigues, F. A., & da Fontoura Costa, L. (2014). A systematic comparison of supervised classifiers. *PloS one*, *9*(4), e94137-NA. https://doi.org/10.1371/journal.pone.0094137

Awan, A. J., Brorsson, M., Vlassov, V., & Ayguadé, E. (2016). BPOE - How Data Volume Affects Spark Based Data Analytics on a Scale-up Server. In (Vol. 9495, pp. 81-92). https://doi.org/10.1007/978-3-319-29006-5_7

Doi: 10.69593/ajsteme.v4i04.146

Badhon, M. B., Carr, N., Hossain, S., Khan, M., Sunna, A. A., Uddin, M. M., Chavarria, J. A., & Sultana, T. (2023). Digital Forensics Use-Case of Blockchain Technology: A Review. AMCIS 2023 Proceedings.,

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The Best of Both Worlds. *Computing in Science & Engineering*, *13*(2), 31-39. https://doi.org/10.1109/mcse.2010.118

Boehm, B., Clark, N. A., Horowitz, N. A., Brown, N. A., Reifer, N. A., Chulani, N. A., Madachy, R., & Steece, B. (2000). *Software Cost Estimation with Cocomo II with Cdrom* (Vol. NA). https://doi.org/NA

Brumfiel, G. (2011). High-energy physics: Down the petabyte highway. *Nature*, *469*(7330), 282-283. https://doi.org/10.1038/469282a

Chen, C. L. P., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, *275*(275), 314-347. https://doi.org/10.1016/j.ins.2014.01.015

Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. *Mobile Networks and Applications*, *19*(2), 171-209. https://doi.org/10.1007/s11036-013-0489-0

Cid-Fuentes, J. Á., Alvarez, P., Amela, R., Ishii, K., Morizawa, R. K., & Badia, R. M. (2020). Efficient development of high performance data analytics in Python. *Future Generation Computer Systems*, *111*, 570-581. https://doi.org/10.1016/j.future.2019.09.051

Conejero, J., Corella, S., Badia, R. M., & Labarta, J. (2017). Task-based programming in COMPSs to converge from HPC to big data. *The International Journal of High Performance Computing Applications*, *32*(1), 45-60. https://doi.org/10.1177/1094342017701278

Dalcin, L., Paz, R. R., Kler, P. A., & Cosimo, A. (2011). Parallel distributed computing using Python. *Advances in Water Resources*, *34*(9), 1124-1139. https://doi.org/10.1016/j.advwatres.2011.04.013

Feng, Q., & Lin, T. T. Y. (2016). *Astroinformatics - The analysis of VERITAS muon images using convolutional neural networks* (Vol. 12). https://doi.org/10.1017/s1743921316012734

Fernández, A., del Río, S., López, V., Bawakid, A., del Jesus, M. J., Benítez, J. M., & Herrera, F. (2014). Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *WIREs Data Mining and Knowledge Discovery*, *4*(5), 380-409. https://doi.org/10.1002/widm.1134

Freeman, J. (2015). Open source tools for large-scale neuroscience. *Current opinion in neurobiology*, *32*(NA), 156-163. https://doi.org/10.1016/j.conb.2015.04.002

Grandison, T., & Sloman, M. (2000). A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials*, *3*(4), 2-16. https://doi.org/10.1109/comst.2000.5340804

Guller, M. (2015a). *Big Data Analytics with Spark - Big Data Analytics with Spark* (Vol. NA). https://doi.org/10.1007/978-1-4842-0964-6

Guller, M. (2015b). *Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis* (Vol. NA). https://doi.org/NA

Guo, X., Ipek, E., & Soyata, T. (2010). Resistive computation. *ACM SIGARCH Computer Architecture News*, *38*(3), 371-382. https://doi.org/10.1145/1816038.1816012

Holch, T., Shilon, I., Büchele, M., Fischer, T., Funk, S., Groeger, N., Jankowsky, D., Lohse, T., Schwanke, U., & Wagner, P. (2017). Probing Convolutional Neural Networks for Event Reconstruction in {\gamma}-Ray Astronomy with Cherenkov Telescopes. *Proceedings of 35th International Cosmic Ray Conference — PoS(ICRC2017)*, *301*(NA), 795-NA. https://doi.org/10.22323/1.301.0795

Huennefeld, M. (2017). Deep Learning in Physics exemplified by the Reconstruction of Muon-Neutrino Events in IceCube. *Proceedings of 35th International Cosmic Ray Conference — PoS(ICRC2017)*, *301*(NA), 1057-NA. https://doi.org/10.22323/1.301.1057

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90-95. https://doi.org/10.1109/mcse.2007.55

Hurst, S. L. (1969). An introduction to threshold logic: a survey of present theory and practice. *Radio and Electronic Engineer*, *37*(6), 339-351. https://doi.org/10.1049/ree.1969.0062

Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., & Nguifo, E. M. (2018). An experimental survey on big data frameworks. *Future Generation Computer Systems*, *86*(NA), 546-564. https://doi.org/10.1016/j.future.2018.04.032

Interlandi, M., Shah, K., Tetali, S. D., Gulzar, M. A., Yoo, S., Kim, M., Millstein, T., & Condie, T. (2015). Titian: data provenance support in Spark. *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, *9*(3), 216-227. https://doi.org/10.14778/2850583.2850595

Istiak, A., & Hwang, H. Y. (2024). Development of shape-memory polymer fiber reinforced epoxy composites for debondable adhesives. *Materials Today Communications*, *38*, 108015. https://doi.org/https://doi.org/10.1016/j.mtcomm.2023.108015

Istiak, A., Lee, H. G., & Hwang, H. Y. (2023). Characterization and Selection of Tailorable Heat Triggered Epoxy Shape Memory Polymers for Epoxy Debondable Adhesives. *Macromolecular Chemistry and Physics*, *224*(20), 2300241. https://doi.org/https://doi.org/10.1002/macp.202300241

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, *31*(8), 651-666. https://doi.org/10.1016/j.patrec.2009.09.011

Jeong, H., & Shi, L. (2018). Memristor devices for neural networks. *Journal of Physics D: Applied Physics*, *52*(2), 023003-NA. https://doi.org/10.1088/1361-6463/aae223

Kim, H., Park, J., Jang, J., & Yoon, S. (2016). DeepSpark: Spark-Based Deep Learning Supporting Asynchronous Updates and Caffe Compatibility. *arXiv: Learning*, *NA*(NA), NA-NA. https://doi.org/NA

Kitchin, R. (2013). The real-time city? Big data and smart urbanism. *GeoJournal*, *79*(1), 1-14. https://doi.org/10.1007/s10708-013-9516-8

Kumar, S., & Roy, U. B. (2023). A technique of data collection. In (pp. 23-36). Elsevier. https://doi.org/10.1016/b978-0-323-91776-6.00011-7

Mangano, S., Delgado, C., Bernardos, M. I., Lallena, M., & Vázquez, J. J. R. (2018). *Extracting gamma-ray information from images with convolutional neural network methods on simulated Cherenkov Telescope Array data* (Vol. NA). https://doi.org/10.1007/978-3-319-99978-4

McKinney, W. (2011). pandas: a Foundational Python Library for Data Analysis and Statistics. *NA*, *NA*(NA), NA-NA. https://doi.org/NA

McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (Vol. NA). https://doi.org/NA

Millman, K. J., & Aivazis, M. (2011). Python for Scientists and Engineers. *Computing in Science & Engineering*, *13*(2), 9-12. https://doi.org/10.1109/mcse.2011.36

Misale, C., Drocco, M., Tremblay, G., Martinelli, A. R., & Aldinucci, M. (2018). PiCo: High-performance data analytics pipelines in modern C++. *Future Generation Computer Systems*, *87*(NA), 392-403. https://doi.org/10.1016/j.future.2018.05.030

Nagpal, A., & Gabrani, G. (2019). Python for Data Analytics, Scientific and Technical Applications. *2019 Amity International Conference on Artificial Intelligence (AICAI)*. https://doi.org/10.1109/aicai.2019.8701341

Oliphant, T. E. (2006). *Guide to numpy* (Vol. 1). Trelgol Publishing USA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825-2830. https://doi.org/NA

Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. *Health information science and systems*, *2*(1), 3-3. https://doi.org/10.1186/2047-2501-2-3

Reed, D. A., & Dongarra, J. (2015). Exascale computing and big data. *Communications of the ACM*, *58*(7), 56-68. https://doi.org/10.1145/2699414

Reyes-Ortiz, J. L., Oneto, L., & Anguita, D. (2015). INNS Conference on Big Data - Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science*, *53*(NA), 121-130. https://doi.org/10.1016/j.procs.2015.07.286

Saika, M. H., Avi, S. P., Islam, K. T., Tahmina, T., Abdullah, M. S., & Imam, T. (2024). Real-Time Vehicle and Lane Detection using Modified OverFeat CNN: A Comprehensive Study on Robustness and Performance in Autonomous Driving. *Journal of Computer Science and Technology Studies*.

Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, *1*(3), 145-164. https://doi.org/10.1007/s41060-016-0027-9

Shamim, M. (2022). The Digital Leadership on Project Management in the Emerging Digital Era. *Global Mainstream Journal of Business, Economics, Development & Project Management*, *1*(1), 1-14.

Doi: 10.69593/ajsteme.v4i04.146

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Ozcan, F. (2015). Clash of the titans: MapReduce vs. Spark for large scale data analytics. *Proceedings of the VLDB Endowment*, *8*(13), 2110-2121. https://doi.org/10.14778/2831360.2831365

Shilon, I., Kraus, M., Büchele, M., Egberts, K., Fischer, T., Holch, T., Lohse, T., Schwanke, U., Steppa, C., & Funk, S. (2019). Application of deep learning methods to analysis of imaging atmospheric Cherenkov telescopes data. *Astroparticle Physics*, *105*(NA), 44-53. https://doi.org/10.1016/j.astropartphys.2018.10.003

Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., Cortes, T., & Labarta, J. (2016). PyCOMPSs: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications*, *31*(1), 66-82. https://doi.org/10.1177/1094342015594678

Uddin, M. M., Ullah, R., & Moniruzzaman, M. (2024). Data Visualization in Annual Reports–Impacting Investment Decisions. *International Journal for Multidisciplinary Research*, *6*(5). https://doi.org/10.36948/ijfmr

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22-30. https://doi.org/10.1109/mcse.2011.37

Yan, D., Cheng, J., Özsu, M. T., Yang, F., Lu, Y., Lui, J. C. S., Zhang, Q., & Ng, W. (2016). A general-purpose query-centric framework for querying big graphs. *Proceedings of the VLDB Endowment*, *9*(7), 564-575. https://doi.org/10.14778/2904483.2904488