Exploring The “Pandas” Library In Python

Mohamed Amin
900191168
Omar Moustafa
900222400

CSCE 3101

Summer 2024

Introduction to the Library:

The “Pandas” library in Python was first created in the year 2008 by American software developer, Wes Mckinney. The initial motivation behind its development was to remove the lack of data structures for dealing with and handling tabular data that the Python language had. Therefore, McKinney developed a library with an extensive list of functions sufficient for “analyzing, cleaning, exploring, and manipulating data.” The main inspiration behind the name ‘Pandas’ itself are the terms ‘Panel Data’ and ‘Python Data Analysis’,” hinting at how its main purpose was to provide procedures and data structures for working with time series and numerical tables.

This library aims to solve many problems related to data analysis and manipulation. Most notably, these include input and output tools, data cleansing, and data and time series analysis. Starting with the input and output tools, Pandas contains several functions that work to read and write data from all kinds of file formats. Data cleansing is achieved as Pandas includes tools for handling any missing data points and also reshaping and filtering the data. Both data analysis and time series analysis are accomplished as the Pandas library has countless functions that work to transform the data to make arriving at insightful conclusions about a complex dataset more straightforward and it provides great support for time series data as one can generate date ranges, move window statistics, and many more.

There are several domains where Pandas is extensively used, such as the fields of Data Science, Healthcare, Finance, just to name a few. Pandas is extremely commonly used in the field of data science for cleaning data and preprocessing procedures. It is also heavily utilized in healthcare-related industries to help process medical records and analyze healthcare analytics. In addition, Pandas is a major tool for most projects in the field of finance as it provides clear and engaging financial models, and quantitative analyses. These are just a few of the various domains that implement the usage of the Pandas library to achieve their objectives and to track how they are progressing to achieve said objectives.

Software Systems Built Using the Library:

The implementation of the Pandas libraries in software systems has grown exponentially over the years to the point where nowadays almost every huge project has a database management system and preprocesses the data using Pandas to achieve its desired tasks and objectives. A couple of examples of such software systems and applications that do exactly that are the music recommendation system of the Spotify phone application and the software system of robots.

1. Spotify's music recommendation system

Looking at Spotify from a more general lens, the backend of the Spotify cell-phone application version of Spotify includes several "interdependent services connected by their messaging protocol. And around 80% of those services are written in Python" (Pakhomova, 2022). Now going into the aspects specifically related to Pandas, as Spotify is well-known for offering its users personalized song and artist suggestions, this is accomplished through the use of highly advanced data analysis and machine learning (ML) algorithms. These particular traits are the main indicators behind the true importance of Pandas in organizing and evaluating the enormous volume of data collected by Spotify. The contribution of Pandas to these data analysis and machine learning-related tasks of Spotify includes roles such as data transformation, model training and evaluation, and real-time data processing, which are all of utmost importance for personalized recommendations.

Starting with data transformation, Spotify gathers information about user activities, including songs listened to, playlists made, and user preferences. This data is then aggregated into structured formats using the Pandas library, turning unstructured and raw data into useful datasets. This, therefore, makes it very possible for data scientists to examine existing trends in song popularity and the behavior of the users.

Regarding the major role of exploratory data analysis, which is known by the acronym 'EDA' for short, where the Pandas library provides efficient and effective exploratory data analysis features that allow data scientists to be more than capable of visualizing data collections and distributions, finding patterns and existing relationships, and also making valuable insights and conclusions about user preferences and listening patterns.

Onto real-time data processing, in order to offer real-time recommendations to the user, Spotify employs Pandas to quickly process streaming data which therefore guarantees that the recommendation engine may modify the recommendations instantly and is always current with the most recent user interactions.

As for feature engineering, this is more representative of the machine learning tasks that Pandas is used for by Spotify where it is used to develop and manipulate features for the machine learning models that are in the driving seat of Spotify's recommendation engine. Specifically, Pandas is heavily used to create user profiles, song attributes and descriptions, as well as interaction matrices which are then passed to the recommendation algorithms and engines to proceed with reasonable recommendations to the user which are based on their activity on the application.

Finally, regarding the role of model training and evaluation, Pandas makes handling missing values, dividing datasets, and normalizing data easier when preparing data for model training and it facilitates the assessment of model performance by offering methods for comparing and contrasting prediction outcomes.

2. Robotic software systems

Moving on to the software system of robots, for these systems to function well and properly, regardless of where they are employed, which can range from manufacturing to advanced vehicles to even healthcare, excellent data processing and analysis are necessary. Therefore, these particular systems frequently use Pandas to be able to sensor data, handle operational logs, and also arrive at reasonable decisions. The contribution of Pandas to robotic systems includes roles such as sensor data processing to help with object detection, path planning, data analysis and logging, simulation, and machine learning (ML).

Starting with sensor data processing, numerous sensors, including cameras, temperature sensors, and many more, are installed on robots, and these sensors produce tons of data. To be able to provide reliable analysis and go through reasonable decision-making, the Pandas library is heavily used to process and clean this sensor data where noise is eliminated and values are normalized.

Moving onto data analysis and logging, operational logs that are produced by the robots themselves must be examined for predictive maintenance, defect finding, and performance monitoring. So, Pandas makes it possible to handle and analyze these logs in an efficient manner, where patterns and anomalies that point to possible problems are determined.

As for path planning, such algorithms require robots to thoroughly analyze large amounts of collected data to find the most ideal pathway and steer clear of any potential hazards. So, Pandas is very helpful in dealing with these said path planning tasks as it assists in converting sensor inputs and environmental data into formats that these particular algorithms can then proceed to use.

With regards to simulation, which is often referred to as the "development phase," the robotic systems have to go through a lot of testing and simulation to make sure that they function properly where in such a phase, the Pandas library is utilized to analyze the outputs of these tests and simulations and to compare various scenarios and configurations and optimize the performance of the robot prior to the deployment taking place.

Finally, the more tasks that are more specific and related to machine learning (ML), as robotic systems heavily use models of machine learning to be able to perform tasks that include recognizing objects and predicting future behaviors, Pandas is employed in such instances to ease the processes of feature engineering, data preparation, and the thorough evaluation of the model accuracy.

Overall, Pandas offers highly strong capabilities for preprocessing, analysis, and data manipulation, greatly improving both the development and functionality of these systems. A couple of examples of Pandas proving these capabilities are that it facilitates effective data handling from sensors to decision-making processes in robotic systems and it also supports personalized music suggestions in Spotify's scenario where the robotic systems were chosen to cover and portray the artificial intelligence (AI) aspect behind Pandas and Spotify was chosen to cover and portray the database aspect and usefulness behind the Pandas library. Due to the valuable trait of adaptability of the library, Pandas is a genuinely great resource for a variety of applications, ranging from sophisticated robots to streaming services to many more software systems and applications.

Major Features of the Library:

1. Key functions and capabilities

The main key functions and capabilities of the Pandas library in Python are importing, exploring, cleaning, and also analyzing the data. These are the few standout functions and capabilities that differentiate Pandas from most of the other libraries in Python (Garg, 2023).

1. **<u>Importing Data</u>**

The Pandas library is extremely versatile for data scientists working with various data formats since it offers a wide range of capabilities to import data from many sources. A few of these are the following:

- ‘read_csv()’: Reads the data from a CSV file into a DataFrame
- ‘read_excel()’: Imports the data from an Excel file
- ‘read_sql()’: Executes a SQL query and returns the result as a DataFrame

The above functions along with many similar ones make it possible for data scientists to easily import data from a variety of sources and formats, which promotes efficient and effective data operations. This will be further explored and dived deeper into the following sections of this report.

2. **<u>Data Exploration</u>**

The Pandas library includes a countless list of functions that help data scientists learn about the data that they are dealing with where these functions allow them to do various jobs such as, “viewing it or getting the mean, average, min/max” or many other kinds of valuable information about the dataset (Garg, 2023).

- **<u>Viewing the Data:</u>**
  - ‘head()’: Displays the first five rows of the dataset. Five is the default number of rows but this can be changed by typing in a different number within the parenthesis
  - ‘tail()’: Displays the last five rows of the dataset. Once again, five is the default number of rows but this can be changed by typing in a different number within the parenthesis
  - ‘shape()’: Displays the number of rows and columns of the dataset, namely the dimensions of the dataset

- **Conducting Statistical Analysis:**
    - ‘describe()’: Returns “the basic statistics of each column”
    - ‘info()’: Gets “the information about the various data types used and the non-null count of each column”
    - ‘corr()’: Short for correlation which hints at its purpose of returning “the correlation matrix between all the integer columns in the data frame
    - ‘memory_usage()’: Also as hinted by the name, will tell the user “how much memory is being consumed” by each one of the columns
- **Indexing & Slicing**
    - ‘Iloc[row_number]’ which selects a specific row “based on its index”
    - ‘[column_number]’ which selects a specific column
    - ‘[[‘column1’, ‘column2’]]’ which selects “multiple columns given”

3. **Data Cleansing**

Cleaning the data is an important step that must not be overlooked as it must come before actually analyzing the data so that one has gotten rid of any null or garbage data values that could “hamper the performance” of the data model. Such functions that serve the purpose of getting rid of this garbage are the following:

- ‘isnull()’: Identifies all of “the missing values” in the data frame
- ‘dropna()’: Removes the rows that contain any “missing values in any column”
- ‘fillna(value)’: Fills “the missing values with ‘value’ given” within the parentheses
- ‘[‘col’].astype(new_data_type)’: Converts “the data type of the selected columns to a different data type”

4. **Data Analysis**

As for the fun part, analyzing the data in-depth, where Pandas provides helpful works that allow the user to group, sort, and filter the data to be able to then conduct a thorough analysis and come out with insightful conclusions.

- **For Grouping:**
    - ‘groupby()’: Splits data into groups based on some criteria and applies functions to each group independently.

- **<u>For Sorting:</u>**
    - ‘sort_values()’: Sort the data by values with the default being in an ascending order
    - ‘sort_index()’ Sort the data by indices with the default being in an ascending order
- **<u>For Filtering:</u>**
    - ‘query()’: Allows filtering of DataFrame using SQL-like queries

Therefore, these key functions and capabilities are integral parts of dealing with datasets, big or small, and they assist data scientists and developers in their work making the data import, exploration, cleaning, and analysis processes achieved in efficient fashions.
(Garg, 2023).

2. Advantages

Pandas has several advantages that make it truly stand out when compared to other libraries in Python but the most notable of these advantages are its ease of use, versatility with all sorts of data structures, as well as its integration with the overarching Python ecosystem.

1. <u>Ease of Use</u>

The main characteristics that reflect the ease of use of the Pandas library are its intuitive syntax and the support it provides to its community of users. Regarding the intuitive syntax, the syntax of Pandas is very straightforward and user-friendly which makes it an accessible library to use and deal with for both experienced and inexperienced data scientists and developers. As for the support that it gives its community, it provides extensive documentation and plenty of tutorials that support its large pool of users with more-than-sufficient resources to study, solve problems, and improve their abilities using the library.

2. <u>Versatility of Data Structures</u>

Pandas offers two highly powerful data structures which are the one-dimensional “Series” and the two-dimensional “DataFrame” which lead to data manipulation processes being done in flexible and straightforward manners. These two particular data structures are known to be highly efficient for handling all sorts of data types and formats.

3. **<u>Integration with the Python Ecosystem</u>**

Pandas is known to be extremely compatible with a lot of the other commonly-used libraries in the Python ecosystem such as Matplotlib and NumPy, just to name a couple, which truly enhances the functionality of Pandas and allows for comprehensive and cohesive data-science-related workflows.

Ultimately, Pandas is truly distinguished from most of the other Python libraries by its easy-to-use and understandable syntax, strong and adaptable data structures, and smooth interaction. There is no doubt at all that Pandas is an essential tool for all data scientists and developers because of these benefits thoroughly expressed above, which lead to better productivity and more insightful conclusions.

Comparison with Similar Libraries:

1. Dask

There is no doubt that when it comes to tasks of data manipulation and analysis, Pandas has established itself as probably the most powerful tool in the entire Python ecosystem. However, since datasets have grown larger and computational demands increased, there has been a need for other modules that combine Pandas's tools with large scales. Dask is one of those modules that extends the capabilities of Pandas by incorporating parallel computing and handling larger datasets that exceed in-memory constraints.

Functionality is a key area where Dask differentiates itself from Pandas. While Pandas provides a rich set of features for data manipulation with its DataFrame and Series structures, Dask builds on this foundation to support out-of-core computation. This means Dask can process data that doesn't fit into memory by breaking it down into smaller chunks and computing them in parallel. Additionally, Dask, just like Pandas, integrates very easily with many other libraries such as NumPy and Scikit-learn, enabling distributed computing that scales with the size of the dataset.

In terms of ease of use, Dask offers an API that closely resembles that of Pandas, making it relatively straightforward for existing Pandas users to transition. However, Dask does come with a steeper learning curve than that of Pandas, but in fact, this steep learning curve is only due to the need for understanding distributed computing environments to be able to have full use of Dask's tools, which may not be necessary when working with Pandas.

When considering performance, Dask shines in scenarios involving very large data volumes. Its design focuses on parallel and distributed computing, which allows it to manage and process large datasets more efficiently than Pandas, which operates in memory and may struggle with data that exceeds available RAM. This makes Dask particularly suitable for big data processing tasks where performance and scalability are critical. However, this also clearly means that Dask is not a suitable solution for small-scale problems as using distributed programming with such problems would incur an unnecessary overhead cost that would end up making solving such problems with Dask way more inefficient than with Pandas.

Lastly, community support is an aspect where Pandas and Dask diverge. Pandas benefit from a large, well-established community with extensive resources, documentation, and third-party packages. This is due to Pandas being more popular than Dask, and being very beginner-friendly. However, Dask, while still growing, has a smaller but increasingly active community, and as Dask gains more attraction for big data applications, which are exponentially rising, its community is surely to significantly widen.

In summary, while Pandas remains a powerful and versatile tool for data analysis problems that are more towards the smaller scale, Dask offers better advantages for handling larger datasets by leveraging parallel computing. The choice between the two depends significantly on the specific needs of the project, mainly the size of the dataset.

2. Polars:

Another Pandas combatant is Polars, a relatively new entrant in the field, which offers an alternative to the widely-used Pandas library. While Pandas has established itself as a foundational tool in the Python ecosystem, Polars presents several enhancements aimed at improving performance and scalability.

Functionality is a primary area of differentiation between Polars and Pandas. Polars offers a similar DataFrame API to Pandas but with optimizations focused on performance. Polars introduces features such as parallel execution and lazy evaluation, which delay computation until necessary. This approach enhances its ability to handle larger datasets efficiently, which solves some of the limitations of Pandas that rely heavily on in-memory operations.

When it comes to ease of use, Polars aims to have a user experience similar to Pandas to make it easier for users familiar with Pandas to transition to Polars. Both libraries offer DataFrame APIs, but Polars introduces its syntax and conventions, which can create a learning curve for new users. Unfortunately, while Pandas has a wide range of tutorials and examples due to its maturity, Polars is still developing its documentation and user resources.

Performance is where Polars distinguishes itself most significantly from Pandas. Pandas operates in memory, limiting its performance and scalability when dealing with large datasets. In contrast, Polars is designed with performance optimization in mind, utilizing multi-threaded execution and memory-efficient data structures to handle large-scale data processing tasks more effectively. The lazy evaluation model in Polars further enhances its performance by optimizing query execution and reducing unnecessary computations. This results in substantial speed improvements for data manipulation tasks, making Polars an attractive option for users requiring high-performance data processing.

Finally, while Pandas benefits from a large, established community with extensive resources, documentation, and third-party packages, Polars, while growing in popularity, has a smaller community compared to Pandas. However, the Polars community is supported by an active development team and is gaining more attention by the day for its innovative features and performance benefits.

All in all, while Pandas remains a versatile and widely supported tool for data analysis, Polars offers significant advantages in terms of performance. The focus on high-speed data processing and efficient memory management makes Polars a very competitive alternative for large datasets and complex tasks especially when the user is confident enough to use the library without the need or support of a wide community, such as Pandas.

Demonstration of Library Usage:

The list of Pandas functions and object methods is extensive, so the following is a brief description of 15 of the most important but not widely-known Pandas functions.

1. **pd.merge():** The merge function is used to vertically merge data frames based on a common column. It can be used to combine additional features from a supplementary dataset, which is super useful in any database management system.

2. **pd.concat():** This function concatenates data frames along a particular axis. It is usually used for horizontally merging two data frames that have the same columns which is essential for adding multiple data entries without having to do it manually step by step or having to use a for loop.

3. **pd.pivot_table():** The function is used to create a pivot table to aggregate and summarize data to understand trends and patterns. For example, it can be used to get the total number of patients divided by their gender in any patient dataset.

4. **pd.melt():** This is used to unpivot data frames from the wide format to the long format which is extremely useful in applications like beautiful plotting and visualization, and data analysis.

5. **pd.crosstab():** As the name suggests, the function is used to get a cross-tabulation of two or more columns in a data frame, which is of course very useful in examining the relationship between categorical variables in data analysis.

6. **pd.cut():** The cut function is used to discretize continuous values into different bins and could also assign labels to them based on their assigned bin. This is useful whenever we want to get a rough insight into any demographic distribution.

7. **pd.qcut():** Similar to the cut function, the qcut function also discretizes the data but it works with a quantile-based discretization, which could be more useful than simple discretization in financial and medical fields.

8. **pd.DataFrame.apply():** The apply method allows us to apply a function along a specified axis (rows or columns) of a data frame or Series. It is useful for custom operations or aggregations where we need to perform complex calculations or transformations row-wise or column-wise, such as aggregating data or applying custom functions across a data frame.

9. **pd.DataFrame.transform():** The transform method is similar to the apply method as it applies a function to each element in a DataFrame or Series and returns an object with the same shape as the input. It is useful for element-wise transformations, such as scaling or normalizing features, where we need to maintain the original structure of the data while applying a function to each element.

10. **pd.DataFrame.groupby().agg():** The groupby().agg() chain method allows you to group data by one or more columns and then apply aggregation functions to each group. It is useful for summarizing data, such as calculating the mean, sum, or other statistics for each group, which helps in understanding patterns and insights across different subsets of any data.

11. **pd.DataFrame.rolling():** this method provides a moving window calculation, allowing us to apply aggregation functions like mean, sum, or standard deviation over a fixed-size window that slides over the data. It is useful for smoothing time series data, analyzing trends, and calculating statistics over a specified number of periods or observations.

12. **pd.DataFrame.expanding():** The expanding method calculates cumulative statistics over a growing window of data. Unlike rolling windows, which have a fixed size, expanding windows include all data up to the current point. It is useful for computing cumulative metrics, such as cumulative sums or means, where you want to account for all

previous observations up to the current data point. This is widely used in financial data analysis to compute things like cumulative returns for instance.

13. **pd.DataFrame.ewm():** The ewm method, short for exponentially weighted mean, calculates just that giving more weight to recent observations while still considering all historical data. It is useful for smoothing time series data, such as calculating exponentially weighted moving averages or volatility, where recent values are more relevant than older ones. This is the case when conducting stock price analysis.

14. **pd.Series.value_counts():** the value_counts method counts the occurrences of unique values in a Series and returns a Series of counts, sorted in descending order. It is useful for summarizing categorical data, understanding the distribution of values, and identifying the most frequent categories or values in a dataset. It is a more Pythonic way of using the standard count function but with dataframes.

15. **pd.DataFrame.pct_change():** This function computes the percentage change between the current and a prior element in a data frame. It is useful for analyzing relative changes over time, such as financial returns or growth rates, by providing insights into the proportional changes between consecutive data points, which is a necessary step in financial returns and stock price analyses.

To test some of these functions in a real project, we used them with the famous Titanic dataset to make a small machine-learning model that predicts whether a passenger survived the Titanic or not based on certain features in the dataset such as age, fare, and income, and other manually-created features such as the family size and the education level. The ML model pipeline consists of two parts: the preprocessing, and the prediction part. The preprocessing part in the pipeline is another pipeline itself with multiple transformers to handle any errors in the dataset. First, we used simple imputers to change the numerical NAN value to its corresponding column’s mean and the categorical NAN values to the most frequent value in their columns. After that we used a standard scalar for standardizing the numerical values, and a one hot encoder for encoding the categorical ones so the model can learn them easily. Finally for the prediction part, we used a RandomForestClassifier with 100 decision trees and fixed seed as we thought it would be the most suitable for our classification task. The accuracy score we got was not that satisfying so we repeated the process using GradientBoostingClassifier and we got a much higher accuracy score. This was logical since the GradientBoostingClassifier is a more powerful ensemble method than the RandomForestClassifier as it focuses on correcting errors from previous iterations.

For a better understanding, the code with comments can be accessed through the following link:
https://colab.research.google.com/drive/1Uvw7wcNTx0YUEf1wfvlOuXZNS5RSxAeX?usp=sharing

Contribution Listing:

- Omar:
    - Introduction to the Library
    - Software Systems Built Using the Library
    - Major Features of the Library

- Mohamed:
    - Comparison with Similar Libraries
    - Demonstration of Library Usage

References

10 essential pandas functions every data scientist should know. KDnuggets. (n.d.). https://www.kdnuggets.com/10-essential-pandas-functions-every-data-scientist-should-know

Dask. *Dask.* https://www.dask.org/

https://www.kaggle.com/c/titanic/data

Pakhomova, K. (2022, January 13). *30 top apps made with python*. DEV Community. https://dev.to/kateryna_pakhomova/30-top-apps-made-with-python-2ah4

Polars. *Polars*. https://pola.rs/

User guide#. User Guide - pandas 2.2.2 documentation. (n.d.). https://pandas.pydata.org/docs/user_guide/index.html#user-guide

W3schools.com. W3Schools Online Web Tutorials. (n.d.). https://www.w3schools.com/python/pandas/pandas_intro.asp#:~:text=Pandas%20is%20a%20Python%20library,by%20Wes%20McKinney%20in%202008

YouTube. (2019, September 3). *Jovan Veljanoski - modern data science with VAEX: A new approach to DataFrames and pipelines.* https://www.youtube.com/watch?v=NMJmR_HXjrQ

YouTube. (2021, June 23). *How to deploy DASK | deployment workshop | dask summit 2021.* https://www.youtube.com/watch?v=faFncNrXmIY