Comprehensive descriptions of the key data structure and algorithm used in task 2 and 3

The project's tasks 2 and 3 take us through the Exploratory Data Analysis (EDA) and some Machine learning modeling. Task 2 is mainly focused on EDA while task 3 combine EDA and Machine Learning techniques.

The codebase prominently relies on *Pandas* DataFrames, which serve as versatile structures for data claening, manipulation , some visualizations and basic statistics. Since we are dealing with labeled rows and columns, Pandas DataFrames facilitate various operations such as selecting, filtering, aggregating, and merging data. Leveraging this functionality, the code implements a range of preprocessing tasks, including handling missing values, feature engineering, and data aggregation.

In conjunction with Pandas, *NumPy* arrays play a pivotal role in numerical computations. Its *where* method has been especially used to filter and make imputation of outliers and missing values. Its vectorization is useful when it comes to large computation.

In order order to automate data cleaning process *sklearn Pipeline* has been used to create pipeline.

Matplotlib and especially *seaborn* are mainly used for data visualization. *Seaborn* provides professionnal design when it comes to presentation.

The integration of *Scikit-Learn* expands the code's analytical horizons, notably through the utilization of the *KMeans* clustering algorithm. Using the *MinMaxScaler* of *sklearn* we first normalized the data and get them in *numpy* array format, before running *KMeans* clustering algorithm. It's an essential step before clustering as Euclidean distance is very sensitive to the changes in the differences.

We also standardized the data before *PCA* analysis for dimension reduction. Using *Standard-Scaler* of *sklearn* we standardize in order to avoid bias in the contribution of each variable to the principal components construction.

Apart from *pandas DataFrame*, we also used *list* especially to get the inertias from *Kmeans* clustering results for different values of the number of clusters in order order to apply the *elbow rule*.

Despite its strengths, scalability remains a critical consideration for the codebase. As datasets scale up in size and complexity, the computational demands imposed by Pandas and Scikit-Learn operations escalate accordingly. As the dataset size grows by 10x, 1000x, 100000x, or 1000000x, both the time and memory requirements of the code will increase proportionally, potentially leading to longer execution times and higher memory usage. To address this challenge, scalable frameworks like Dask or Spark may offer viable alternatives, enabling distributed computing and parallel processing to handle big data effectively.

Nevertheless, within the manageable datasets, the code provides a robust toolkit for comprehensive data analysis. Through a combination of *Pandas DataFrames, NumPy arrays, and Scikit-Learn* algorithms, users can explore, preprocess, analyze, and visualize data with ease. The modular nature of the code facilitates extensibility and customization, empowering users to adapt it to their specific analytical needs.

In summary, while the code excels in addressing the analytical requirements of modest-sized datasets, it necessitates careful consideration and potential adaptation for scalability to larger data volumes. By leveraging appropriate tools and frameworks, users can harness the full potential of their data assets while mitigating the computational challenges posed by increasing data size and complexity.