Data Analysis With Python

This report summarizes the code for data analysis with Python. The code performs the following tasks:

Importing libraries: The code imports pandas, numpy, matplotlib, seaborn, and cudf as libraries for data manipulation and visualization.

Exploring a DataFrame:

The code loads a CSV file containing data about Uber rides in Boston, and inspects its shape, columns, and data types. The data has 693,071 rows and 57 columns, with various features related to the ride, location, time, and weather.

Data cleaning:

The code checks for missing values, duplicates, and outliers in the data, and applies appropriate methods to handle them. The code drops some columns that are redundant or relevant, such as id, product\_id, icon, and visibility.1. The code also converts some columns to categorical or datetime data types, and creates new columns for day of week and time of day.

Data Visualization

The code uses matplotlib and seaborn to create various plots to explore the relationships between the features and the target variable (price). The code plots histograms, boxplots, scatterplots, bar charts, and heatmaps to show the distribution, correlation, and trends of the data. The code also uses groupby and pivot\_table methods to aggregate and summarize the data by different categories, such as cab\_type, name, source, destination, day, and hour.

Some of the plots are shown below:Number of Trips vs Hours: This plot shows the distribution of trips over time. The plot reveals that the number of trips peaks at 8 am and 5 pm, which are the typical rush hours. The plot also shows that the number of trips is lowest at 4 am, which is the least busy time of the day.

Cab Type Comparison: This plot shows the comparison of cab types. The plot reveals that UberX is the most popular cab type, followed by UberPool and UberXL. The plot also shows that Black SUV and Lux Black XL are the least popular cab, which are also the most types expensive ones.

Geographic Distribution of Trips: This plot shows the geographic distribution of trips. The plot reveals that the most common sources and destinations are Back Bay, Beacon Hill, Boston University, Fenway, Financial District, Haymarket Square, North End, North Station, South Station, and Theater District. The plot also shows that the trips are clustered around the center of the city, where most of the attractions and businesses are located.

Weather Impact on Trips: This plot shows the impact of weather on trips. The plot reveals that there is a positive correlation between temperature and distance, meaning that people tend to take longer trips when the weather is warmer. The plot also shows that there is a lot of variation in the distance, which could be influenced by other factors such as traffic, surge multiplier, and cab type.

Time Series Analysis of Trips: This plot shows the time series analysis of trips. The plot reveals that there is a seasonal pattern in the distance, meaning that the distance varies depending on the month and the day of the week. The plot also shows that there is a weekly cycle in the distance, meaning that the distance is higher on weekdays than on weekends.

Weather Condition Analysis: This plot shows the weather condition analysis of trips. The plot reveals that there is a difference in the distance depending on the weather icon, which represents the summary of the weather condition. The plot also shows that the distance is higher when the weather icon is clear-day, partly cloudy-day, or rain, and lower when the weather icon is cloudy, fog, or snow.

[Weather Condition Analysis]!

Distribution of Distance: This plot shows the distribution of distance. The plot reveals that the distance is skewed to the right, meaning that most of the trips are short, and only a few are long. The plot also shows that the mean distance is 2.19 miles, and the standard deviation is 1.13 miles.

![Distribution of Distance]

Data Encoding

The last step of the data analysis process is to encode the categorical variables, so that they can be used for machine learning models. The code uses sklearn’s LabelEncoder to transform the categorical variables into numerical values. The code encodes the following columns: id, datetime, timezone, destination, product\_id, and short\_summary. The code also drops the original columns after encoding.

Data Modeling

The final step of the data analysis process is to apply different machine learning models to predict the price of the rides. The code uses sklearn’s train\_test\_split function to split the data into train and test sets, with 80% of the data for training and 20% of the data for testing. The code also uses sklearn’s RFE function to select the best features for each model.

The code applies the following models:

Linear Regression

Decision Tree Regressor

AdaBoost Regression

GradientBoostingRegresso

Random Forest Regressor

Support Vector Regressor

The code evaluates the performance of each model using the following metrics:

Mean Absolute Error (MAE)

Mean Squared Error (MSE)

Root Mean Squared Error (RMSE)

R-squared Score (R2)Formularbeginn

Pipeline Definition:

Defines a scikit-learn pipeline (Pipeline class) containing multiple regression models: Linear Regression, Decision Tree, Random Forest, Gradient Boosting, AdaBoost, and Support Vector Regressor (SVR).

Model Training and Testing:

Iterates through each step in the pipeline.

For classifiers, it prints the accurate score.

For regressors, it prints the R-squared score.

Accessing Individual Models:

Accesses individual models from the pipeline using pipeline.named\_steps['model\_name'].

Hyperparameter Tuning:

Defines hyperparameter grids for each model in the param\_grid dictionary.

Uses GridSearchCV to perform hyperparameter tuning for each model in the pipeline.

Prints the best hyperparameters and the corresponding negative mean squared error.

Updating Pipeline with Best Hyperparameters:

Updates the pipeline with the best hyperparameters obtained from hyperparameter tuning.

Retraining and Testing with Tuned Hyperparameters:

Iterates through each step in the pipeline.

For classifiers, it prints the accuracy score with tuned hyperparameters.

For regressors, it prints the R-squared score with tuned hyperparameters.

In summary, the code aims to create a pipeline with multiple regression models, tune hyperparameters for each model, and evaluate their performance on a given dataset. It demonstrates the use of pipelines and hyperparameter tuning in scikit-learn for regression tasks.