Madalyn Li
DATA 515
Winter 2022

EDA-assistant Written Report

Background

The goal of this project is to help data scientists or data analysts perform easy and quick exploratory data analysis in Python. With the current process for EDA in Python involving importing many packages and writing multiple lines of code, the EDA-assistant package makes this process simpler for the end user with just a single import and two lines of code to produce a PDF report containing all standard EDA summary statistics and graphs. Specifically, the EDA PDF report produced contains tables for dataset and variable summary statistics calculations, bar graphs for visualizing data distribution, a correlation matrix heat map plot, and a scatter pair plot.

Functional Specification:

Target Users

The target users of this package are data scientists or data analysts who are seeking a more simplified and efficient way to perform basic EDA techniques and tasks on a dataset. In addition, another target user is an inexperienced Python user who is still learning and struggling with how to utilize the many functions within the packages of Pandas, Matplotlib, or Seaborn to manipulate or visualize datasets. Beforehand, users should already know how to install and import packages in Python and know how to run command lines. In addition, users should already be able to read through and understand the basic summary statistics tables and graphs provided by the EDA report.

Data Sources:

The EDA-assistant package utilizes a total of three data sources, all from Kaggle, for testing purposes and demonstration purposes. The file names, source links, short description, and purpose within the package are all listed below.

1. Iris Flower Dataset
    - File Name: IRIS.csv
    - Source: Kaggle Iris Flower Dataset
    - Description: This dataset contains a set of 150 records under 5 attributes - Petal Length, Petal Width, Sepal Length, Sepal width and Class (Species)

    - Purpose: This file is used for the demonstration of the package
2. Wine Quality Dataset
    - File Name: WineQT.csv
    - Source: Kaggle Wine Quality Dataset
    - Description: This dataset contains wine quality measurements based on physicochemical tests from various samples of wine
    - Purpose: This file is used for the demonstration of the package
3. Cereal Dataset
    - File Name: cereal.csv
    - Source: Kaggle 80 Cereals Dataset
    - Description: This dataset contains nutrition measurements from various cereal brands
    - Purpose: This file is used for the test code in the package

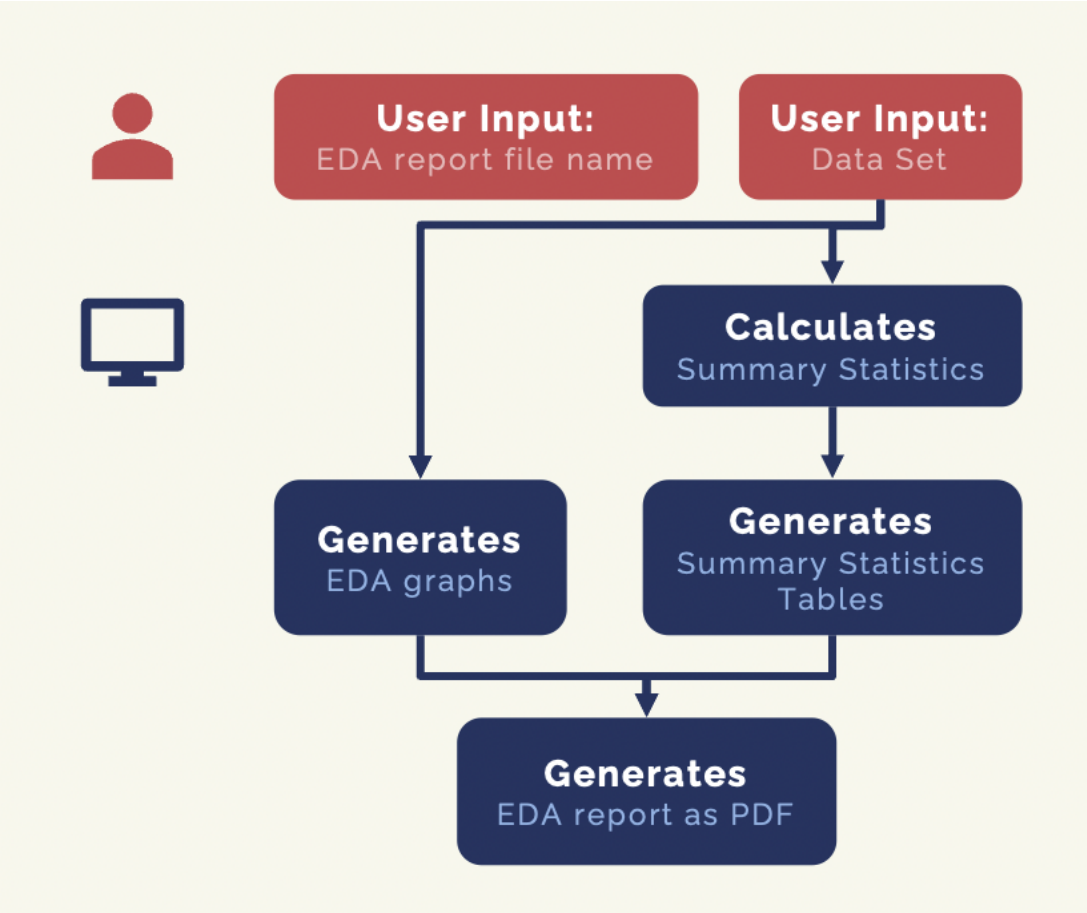
Use Case:

The user will be able to pip install the EDA-assistant Python package and import the package and the eda_assistant module. The user will have a dataset file in the form of a .csv that they will want to perform exploratory data analysis on. The user will be able to create an instance of the EDA class with the dataset file and use this class to perform a task such as creating the EDA report. A summary and more details of this use case are provided below:

Use Case Title: Create EDA Report

- Objective:
    - Create an EDA report PDF file containing dataset summary statistics and graphs.
- Interactions:
    - USER: Install EDA-assistant library and import eda_assistant module from the eda_assistant package.
    - USER: Create an instance of the EDA class with the input of the dataset file path and name to perform exploratory data analysis on.
    - USER: Run create_eda_report on the EDA object which takes in the input of the file name the user wants to save the PDF report as.
    - PROGRAM: Calculates summary statistics for the dataset and variables and combines this into a table format.
    - PROGRAM: Generates univariate and bivariate graphs.
    - PROGRAM: Combines tables and graphs into a PDF file and saves this to the save file name previously inputted by the user.
    - PROGRAM: Opens the file so the user can immediately review the information.

- Figure: The figure below shows the high-level flow for this use case



Additional Note: I had originally planned to add more functionalities to the EDA class for more use cases, however, given the time constraints of the course and project I was unable to do so. I wanted to emphasize this because I wanted it to be clear as to why I decided to create an EDA class for the user to instantiate with only a single use case or functionality for now. I explain this in more detail in the subsequent sections below.

Component Specification:

Assumptions and Dependencies:

Before diving into the details of the software components, I wanted to highlight and emphasize some important assumptions and dependencies of this Python library. Given time constraints and my intermediate skill levels in Python and coding, I had to adapt and compromise some features that may

not always make the end results of the EDA report 100% accurate and functional. These assumptions and dependencies are detailed below:

- Dataset file to create an EDA class must be in a .csv file format
- The variable types in the dataset are determined with Panda's dtype function, which may not always identify the correct variable type 100% of the time
- The categorical bar plots in the EDA report will not be plotted unless the number of unique variables in the categorical column is less than or equal to 10. This is because as the number of bars surpass 10, the bar plot becomes more compressed and thus harder to read
- The scatter pair plot in the EDA report will not be plotted unless the number of numeric variables in the dataset is less than or equal to 10. This is because as the number of variables surpass 10, the processing time for the plot takes much longer to produce
- The PDF format of the EDA report may vary widely; the title of the pages may sometimes overlap the title of the graphs or have a large white-space gap between them

Software Components:

_calc_dataframe_statistics.py

A module that contains functions for calculations for dataset summary statistics

- Input:
  - Dataset for exploratory data analysis
- Output:
  - Calculations for # of columns, # of rows, # of total values, # of NaNs, % of NaNs, # of duplicate rows, % of duplicate rows, # of numeric variables, and # of categorical variables

_calc_variable_statistics.py

A module that contains functions for calculations for the variable summary statistics

- Input:
  - Dataset for exploratory data analysis
- Output:
  - Results for numeric variables: mean, median, sum, variance, standard deviation, 25th percentile, 75th percentile, minimum, maximum, and skew
  - Results for all variables: variable type, if the column contains numeric variables, # of NaNs, % of NaNs, and # of unique values

_create_tables.py

A module that contains functions for creating summary statistics tables

- Input:
  - Dataset for exploratory data analysis
- Outputs:

    - Dataset and variable summary statistics tables

_format_tables.py
A module that contains functions that format the summary statistics tables

- Input:
    - Dataset and variable summary statistics tables
- Output:
    - Formatted dataset and variable summary statistics tables

_create_graphs.py
A module that contains functions for creating univariate and bivariate analysis graphs

- Input:
    - Dataset for exploratory data analysis
- Output:
    - Multiple plots: histogram plots for numeric variables, bar plots for categorical variables, correlation heat map plot, and scatter pair plot

_format_graphs.py
A module that contains functions that format the univariate and bivariate analysis graphs

- Input:
    - Univariate and bivariate graphs
- Output:
    - Formatted univariate and bivariate graphs

eda_assist.py
A module that contains an EDA class to perform EDA tasks and techniques given a dataset

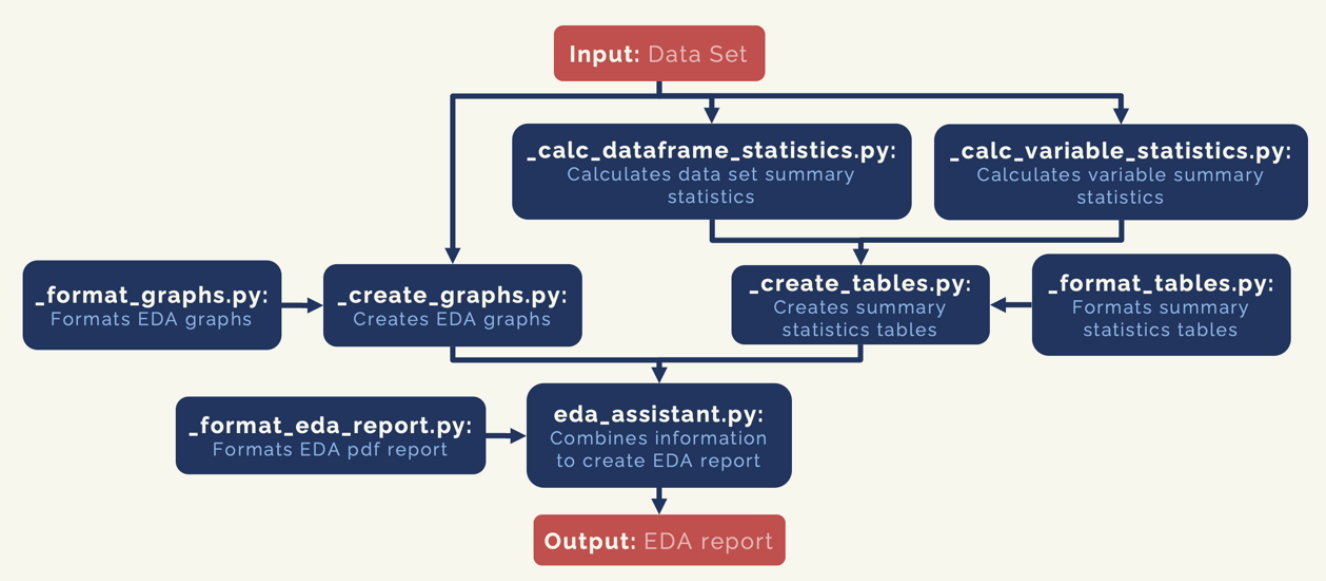
- Input:
    - The dataset's CSV file name path as a string
- Output:
    - A saved and opened PDF file of the EDA report

_format_eda_report.py
A module that contains functions that format the EDA report

- Input:
    - Figures in the EDA report
- Output:
    - Formatted figures for the EDA report

Components Figure:

The figure below shows the flow for the software components of the library



Components Interaction:

1. User creates instance of the EDA class with an input of the dataset .csv file path and name as a string
2. User calls create_report method from the EDA class with an input of the file name that the user wants to save the EDA report to
3. In the background, the modules _calc_dataframe_statistics and _calc_varaible_statistics takes in the dataset and performs calculations for dataset and variable summary statistics
4. The module _create_tables takes the previous calculations and combines them into summary statistics tables while the module _format_tables formats these tables
5. Meanwhile, the module _create_graphs produces univariate and bivariate graphs from the data set and the module _format_graphs formats these plots
6. The module eda_assistant generates a PDF report combining figures for all tables and graphs produced while the module _format_eda_report formats the pdf file

Design Decision Discussion

Background

In this section, I will address two major design decisions I made during the development of the EDA-assistant library. The first design decision I made was in how I decided to separate the concerns of the package into the various modules present. The second decision pertains to why I chose to create an EDA class for the specific purpose of this library. In the subsequent sections, I will dive into a more detailed explanation and discuss my thought process and reasoning behind these design choices.

Separation of Concerns

The structure of the package is broken down into very detailed specific tasks and modules. In particular, the concerns are abstracted into calculating dataset summary statistics, calculating variable summary statistics, creating the summary statistics tables, formatting the summary statistics tables, creating the EDA graphs, formatting the EDA graphs, creating the EDA class to generate the EDA report, and formatting the EDA report. Although this may arguably seem to be too granular of separation of concerns, my decision to break it down as such is based on the motivation to avoid any possible conflict or confusion of tasks and to make future adaptations and changes easier to implement.

Initially, I had decided to divide the modules within the package into only 4 concerns: calculating summary statistics, creating tables, creating graphs, and generating the EDA report. However, as I began expanding, improving, and continuously debugging the code, I quickly realized how tedious it was to scroll through 100+ lines of code in order to find and reference a task. For instance, when I combined all the calculations for summary statistics into one module, this proved to be a very long list of calculations that was often difficult to discern and know which calculations clearly served for which purpose in the main report. Abstracting the library into more intricate tasks made the most sense in the context of the purpose and use for this library. I believe this design decision will help avoid future confusion for location of methods; if a developer wanted to improve on the existing library, he or she would know exactly which module or task to target in order to successfully do so. Additionally, all modules except for eda_assistant are kept private so that for the end user of the library, they are only exposed to the module that matters to them: creating the EDA report.

EDA Class

As mentioned briefly before in the functional specification, the EDA report is generated as a method for the EDA class that the user must instantiate at the start to utilize this library. The purpose of creating an EDA class is to be able to have the users of this library be able to perform multiple EDA tasks and techniques on a dataset of interest. Initially, after reviewing many other existing libraries to gain inspiration and ideas on how to create my eda_assistant package, my plan was to construct a Report class that would allow users to manipulate and select what information would go onto the EDA report. However, after reflecting on the target users of this library, I realized that providing the option

to select what information could be added onto the EDA report is redundant and unnecessary. Users such as data scientists and data analysts would care more about gaining as much insight as possible for their dataset and just having all the information available at once rather than wanting to have the freedom to manipulate the EDA information provided. This led to the decision to reimplement the use of the eda_assistant library to include an EDA class where the user can perform many EDA tasks with it. Although I did not have enough time to expand these methods into the package, some ideas that I have for the future include an outlier detection and removal tool and an easy method to impute and handle missing values.

Library Comparison

Background

The software library I chose to compare EDA-assistant to is named DataPrep:

- The documentation and information for DataPrep can be found here
- The GitHub repository for DataPrep can be found here

Specifically, within the library of DataPrep, I wanted to focus on and compare the create_report package because the functionality and purpose of this most closely matches the library I created. In the following sections, I plan to discuss major design differences between DataPrep's create_report package and EDA-assistant; I will focus on separation of concerns, class creation & attributes, and report output.

Separation of Concerns

Within DataPrep's create_report package, there only contains 2 modules: formatter.py and report.py. Reviewing the contents within these modules, formatter contains methods to format the various sections of the report such as the overview, variables, and correlation sections. In addition, formatter also contains methods to calculate the values for the overview and variables section. In report, the module contains an implementation of the Report class that consists of methods to save, open, and render it. As mentioned previously in the design discussion section of this written report, my package contains more granular separation of concerns and more modules to break down the tasks for calculations, creating and formatting tables, creating and formatting graphs, and creating and formatting the report.

Class Creation & Attributes

DataPrep's create_report package contains a Report class that includes various methods that the user can call to view and save the report. In addition, the attributes that the class takes in are a loaded data frame with the dataset values and a title for the report. Contrastingly, EDA-assistant implements an EDA class that currently allows users to call a method to create a report, and in the future will allow users to call methods for handling outliers and missing values. The attribute taken into the class is the dataset file name path as a string.

Report Output

The output of DataPrep's create_report package produces a well formatted report containing aesthetically designed tables and graphs. It includes a dataset statistics section with summary statistics values, a variable statistics section with summary statistics values and a bar plot for each numeric value, a bivariate interactive plot displaying the scatter plot and regression line between two selected variables, a correlation plot, and a missing values plot. DataPrep's output can be saved as an HTML output or shown in the user's browser. The output for EDA-assistant contains dataset statistics, variable statistics, histograms for numeric and categorical variables, a correlation plot, and a scatter pair plot all contained within a PDF file.

Extensibility

On the topic of extensibility, I do believe some aspects of this package are extensible and that there is still some room for improvement in other areas. First and foremost, the package is extensible because it leaves the option to add additional methods to the existing EDA class as I mentioned previously. For instance, if another developer wanted to add an outlier detector and remover functionality to the class, they would just need to create additional modules to extend the functionality and add the new method to the existing class. This would not disrupt any existing functionalities of the package and likely would not break the code in any way.

An additional aspect of the library that is also extensible is in creating and formatting additional graphs for the report. Currently, the package is designed so that each type of graph is contained within its own method to both create and format it. If a developer wanted to construct a new plot, he or she would simply add new methods to create this plot and format it into the respective create_graphs and

format_graphs modules. Furthermore, if someone were to extend on the format of the existing plots, such as adding varying color schemes, they could easily do so within the format_graphs modules without breaking anything else in the library or having to additionally change code in other modules.

On the other hand, the library could use some improvements on the extensibility for adding additional calculations to the summary statistics tables. If a developer wanted to add an additional calculation metric to the statistics table, he or she would have to update 3 or more different modules in order to do so. First, they would need to add the function to calculate the measure into calc_dataframe_statistics or calc_variable_statistics. Next, they would need to add this calculation to create_tables as well as format_tables. Finally, they would have to update the test code to include this new added measure. Although I did attempt to mitigate this issue, none of the solutions that I tried ended up working.

Conclusion

Reflecting on the entirety of this project, I wanted to conclude this report by stating that this was a great learning experience for me. I learned that considering software design is valuable and important in software development because it ultimately affects the end user. This project helped open my eyes to the software design considerations of the products and software I use daily. Although I don't realize or reflect on this often, there is a lot of thought that goes into the consideration of software design to make our lives easier. Finally, I learned that there are many different ways to incorporate software design and each method has its advantages and disadvantages; as long as we are constantly reflecting on how the end user is affected, we will continue to improve and expand for the better.