Postgraduate Certificate in Cloud Native Computing
Postgraduate Certificate in Software Design with Artificial Intelligence

Applied Scripting Languages

Assignment 1

Student ID: A00267948

Student Name: Daniel Foth

Brief Description:
This assignment aims to design, implement and test a Python program to analyse and visualize a csv data set about information on bike sharing. In this document, the data, testing, visualisation and reflective learning will be described in detail.

Contents

Introduction

This assignment aims to design, implement and test a Python program to analyse and visualize a csv data set about information on bike sharing. In the Data section below you can find information about the data, how its laid out and where it was sourced from.

The design for the program should be modular and allow for the importing of custom data sets to be processed via user input. It should be able to output graphs for linear regression and normal distribution. More information can be found in the design section below.

There are multiple ways to test code. As part of this assignment manual, unit and user testing shall be completed and described in the testing section below.

The analysis and visualisation of the data will help evaluate how successfully the data was analysed as part of the conclusion below.

Lastly a reflective log will be available detailing how the code was written and what issues were overcome.

Data

The data set was taken from https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset

This data set was composed by Hadi Fanaee-T and records the number of users interacting with a bike sharing system. It can give us an interesting view of how the bikes are utilised in different weather conditions. It also shows us how many registered/casual users use the services.

Content

The data set includes 2 csv files. hours.csv and days.csv. hours.csv has 17 columns and 17389 entries. This will be the default dataset used for the analysis. days.csv has 16 columns and 731 entries.

Columns

The following columns are present:

COLUMN	DESCRIPTION
INSTANT	Index of the record
DTEDAY	The date of the record
SEASON	The season the record was taken in. (1=Springer, 2=Summer, 3=Fall, 4=Winter)
YR	The year of the record. (0=2011, 1=2012)
MNTH	The month of the record. (from 1 to 12)
HR	(not in days.csv) The hour of the record. (from 1 to 12)
HOLIDAY	Defines if a day is a holiday. (0=false, 1=true)
WEEKDAY	Defines if a day is a weekday. (0=false, 1=true)
WORKINGDAY	Defines if a day is a working day meaning it's a week day but not a holiday. (0=false, 1=true)
WEATHERSIT	This defines what the weather was like at the time of taking the record. (value from 1-4) 1. Clear, Few clouds, Partly cloudy, Partly cloudy 2. Mist & Cloudy, Mist & Broken clouds, Mist & Few clouds, Mist 3. Light Snow, Light Rain & Thunderstorm & Scattered clouds, Light Rain & Scattered clouds 4. Heavy Rain & Ice Pallets & Thunderstorm & Mist, Snow & Fog
TEMP	Normalized temperature in Celsius. (max is 41)
ATEMP	Normalized feeling temperature in Celsius. (max is 50)
HUM	Normalized humidity. (max is 100)

WINDSPEED	Normalized wind speed. (max is 67)
CASUAL	Number of causal users per record
REGISTERED	Number of registered users per record
CNT	Total number of users per record

Formation

To compose this data Hadi Fanaee-T used 3 sources:

1. Original Source: http://capitalbikeshare.com/system-data
2. Weather Information: http://www.freemeteo.com
3. Holiday Schedule: http://dchr.dc.gov/page/holiday-schedule

Why

This data set was chosen as it was interesting. There could be some interesting correlation between weather, workdays, holidays and the number of users of the Bike services. Maybe graphing this data in different ways could lead to some interesting results.

Design

The project was designed with ease of development in mind. Therefore, git was used for version control and code was made modular when repetition occurred. The code and design developed over time in an agile like way as it was hard to define exactly what the application should be capable of from the very start.

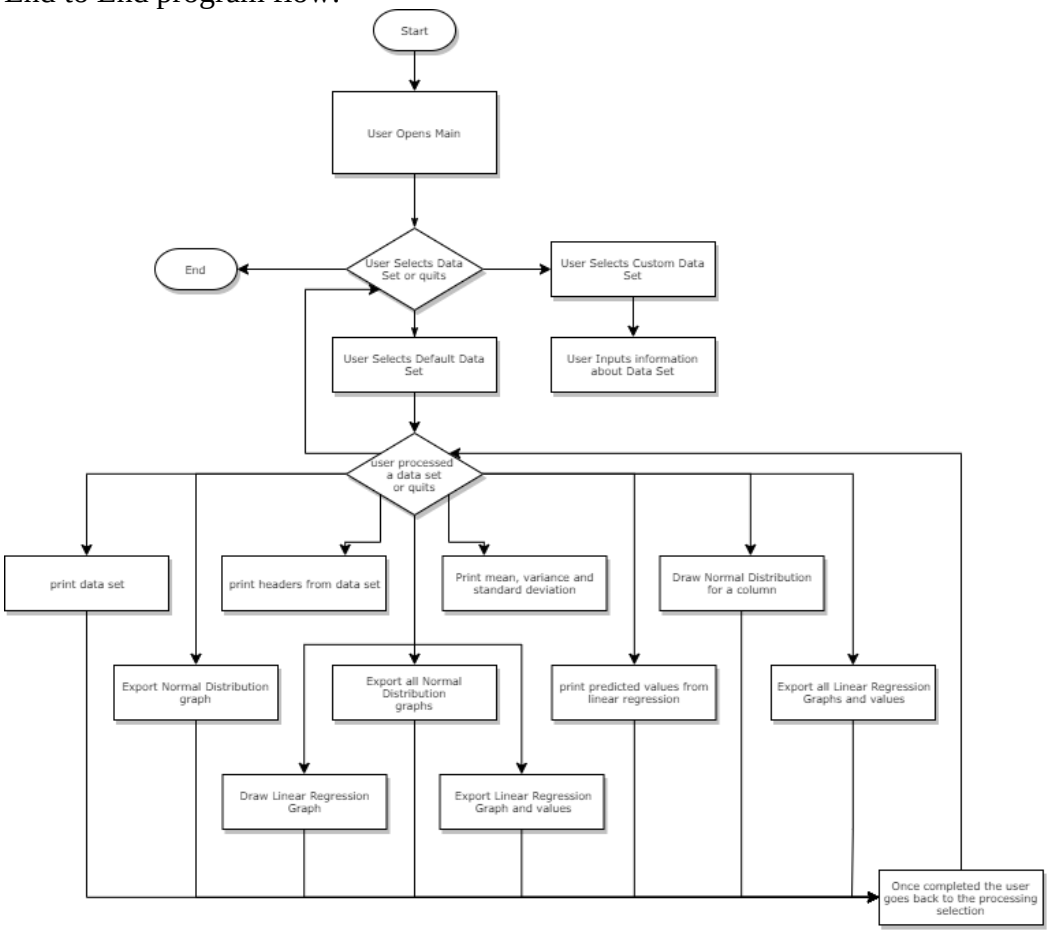
Git

The GitHub repo can be found here: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch. It shows how the design has been built incrementally with improvement over time.

Program Flow

The python program can load in the provided data set or any other csv data file. When loading a custom file, it will ask the user where the file is, what are the column types and what is the delimiter. In both cases, when the file is loaded, the user can select how to process the file. He can print/export data based on normal distribution or linear regression using integer/floating-point columns in the dataset. Exporting data can be done for all columns at once for just one.

End to End program flow:



Modularity

Keeping modular design in mind, classes were used to split up the code where applicable. The project consists of 3 main components:

1. Main Class – responsible for collecting user input and triggering processing on the DataFrame class.
2. DataFrame Class – responsible for processing of the data set with the help of the Utility Classes
3. Utility Classes – responsible for keeping the code maintainable and modular. This includes any maths, input and graphing utilities that are used in multiple places.

The current file structure for the design is laid out as below:

```
├── Documents
│   ├── asl_assignment1_template.docx
│   ├── ~$l_assignment1_template.docx
│   └── Student Plagiarism Disclaimer Form.pdf
├── path.txt
├── README.md
└── src
    ├── constant.py
    ├── data_frame.py
    ├── __init__.py
    ├── input
    │   ├── day.csv
    │   ├── hour.csv
    │   └── Readme.txt
    ├── main.py
    ├── output
    ├── tests
    │   ├── __init__.py
    │   ├── test_data_frame.py
    │   ├── test_exception_utils.py
    │   ├── test_files
    │   │   ├── test.csv
    │   │   └── test_invalid.csv
    │   ├── test_graph_utils.py
    │   ├── test_helpers
    │   │   └── std_in_out_helper.py
    │   ├── test_maths_utils.py
    │   └── test_text_utils.py
    └── utils
        ├── exception_utils.py
        ├── graph_utils.py
        ├── maths_util.py
        └── text_utils.py

8 directories, 25 files
```

- Any documentation is present in the Documents directory or root of the project.
- All of the code is in the src directory and the main() function can be found in main.py.
- The files used for processing are stored in src/input and any output is put in src/output. These paths are accessed relatively so the project can be moved to different file systems.
- The tests are stored under src/tests with any helper functions in src/tests/test_helpers.

Testing

Three kinds of testing techniques were performed to test the python program. These include Manual testing, Unit testing and User testing.

Manual Testing

The code was manually tested on a system with Python 3.7.4 [MSC v.1915 64 bit (AMD64)] on windows 32 mainly within the PyCharm Editor. Manual testing was done for each independent change. When the tiny independent changes made up a new feature the whole application was end to end tested with different flows in mind. The testing was done with the main data set and two test.csv files in the src/test directory.

Unit Testing

Due to time constraints and how the module was laid out the tests were mostly written after the coding for the different components was complete, but bugs were ironed out from the overall design using these. Thanks to unit tests some flows that weren't tested often, one which broke over time, are now tested every push using automation. This made development more efficient as not everything needs to be tested now.

These unit tests can be triggered using pytest. They must be triggered manually from the src/tests directory as they are dependant on relative paths to get the test.scv files.
There are a lot of end to end flows relying on user input which are hard to unit test which lowers the overall test coverage.
Here is the current breakdown from pytest:

```
============================= test session starts =============================
platform win32 -- Python 3.7.4, pytest-5.2.1, py-1.8.0, pluggy-0.13.0
rootdir: C:\Users\efotdan\projects\personal\ASL-CA1-Data-Science-from-Scratch\src\tests
plugins: arraydiff-0.3, doctestplus-0.4.0, openfiles-0.4.0, remotedata-0.3.2
collected 37 items

test_data_frame.py ......                                                [ 16%]
test_exception_utils.py .......                                          [ 35%]
test_graph_utils.py ..                                                   [ 40%]
test_maths_utils.py ..........                                           [ 67%]
test_text_utils.py ............                                          [100%]

============================== 37 passed in 2.15s ==============================
```

User Testing

There were two user testing sessions to see how the application preformed. Both usability and weird inputs were tested by the users.

Session 1 Feedback (Around Nov 5th):

- More input validation needed – added extra validation so the application is more resilient to errors input
- Graph colours clashing and hard to look at – changed to similar cooler colours so they don't clash
- Graphs were not clearing correctly – now clear graphs on reload
- Data did not load correctly on reload – now clear pyplot context before new graph is rendered

- Linear Regression doesn't suit most of the default data set – this is very valid so extra analysis was taken and a Distribution graph was implemented for the next test

Session 2 Feedback (Around Nov 23rd):

- When selecting string column, the error prompt said to only select an integer – changed prompt to ask for an integer/float column
- What directory is the files exported to? – added a print to tell where the files are exported
- Single export seems to output to the wrong date folder – default value was persisting across all calls to function, so date is now always passed in as a param.
- All linear regression export files should be consistent, start or end with lr – added lr_ to start of csv and txt export for linear regression to match the format for png files
- When reloading a custom data set the previous data types are used – default value was persisting across all calls to function, so data types are now always passed in.
- When asking for data types the last one always ends up on 2 lines – fixed formatting using .strip() as there was a line break for the last value.

Analysis and Visualisation

There are three types of data provided by the program. The information about the data set, normal distribution and linear regression outputs. The information about the data set is printed to the screen. Normal distribution and linear regression provide both graphical and text output. They can also be exported to files.

Data Set Information

The whole data set can be printed to screen as a table separated using tabs. The headings can also be listed with the data types assigned to them.

Printed Headers:

```
header 0: instant - int
header 1: dteday - str
header 2: season - int
header 3: yr - int
header 4: mnth - int
header 5: hr - int
header 6: holiday - int
header 7: weekday - int
header 8: workingday - int
header 9: weathersit - int
header 10: temp - float
header 11: atemp - float
header 12: hum - float
header 13: windspeed - float
header 14: casual - int
header 15: registered - int
header 16: cnt - int
```

Printed Data:

```
6032    2011-09-13    3    0    9    21    0    2    1    1    0.64    0.5909    0.78    0.1642    45    200    245
6033    2011-09-13    3    0    9    22    0    2    1    1    0.64    0.5909    0.78    0.1343    24    120    144
6034    2011-09-13    3    0    9    23    0    2    1    1    0.64    0.5909    0.78    0.1045    15    59    74
6035    2011-09-14    3    0    9    0    0    3    1    1    0.62    0.5909    0.78    0.0896    5    28    33
6036    2011-09-14    3    0    9    1    0    3    1    1    0.62    0.5909    0.78    0.0896    1    7    8
6037    2011-09-14    3    0    9    2    0    3    1    1    0.6    0.5606    0.83    0.0    1    4    5
6038    2011-09-14    3    0    9    3    0    3    1    1    0.6    0.5455    0.88    0.1343    1    7    8
6039    2011-09-14    3    0    9    4    0    3    1    1    0.6    0.5606    0.83    0.0896    1    8    9
6040    2011-09-14    3    0    9    5    0    3    1    1    0.58    0.5455    0.88    0.1045    1    30    31
6041    2011-09-14    3    0    9    6    0    3    1    1    0.58    0.5455    0.88    0.1045    7    138    145
6042    2011-09-14    3    0    9    7    0    3    1    1    0.6    0.5606    0.83    0.1045    20    350    370
```

Normal Distribution.

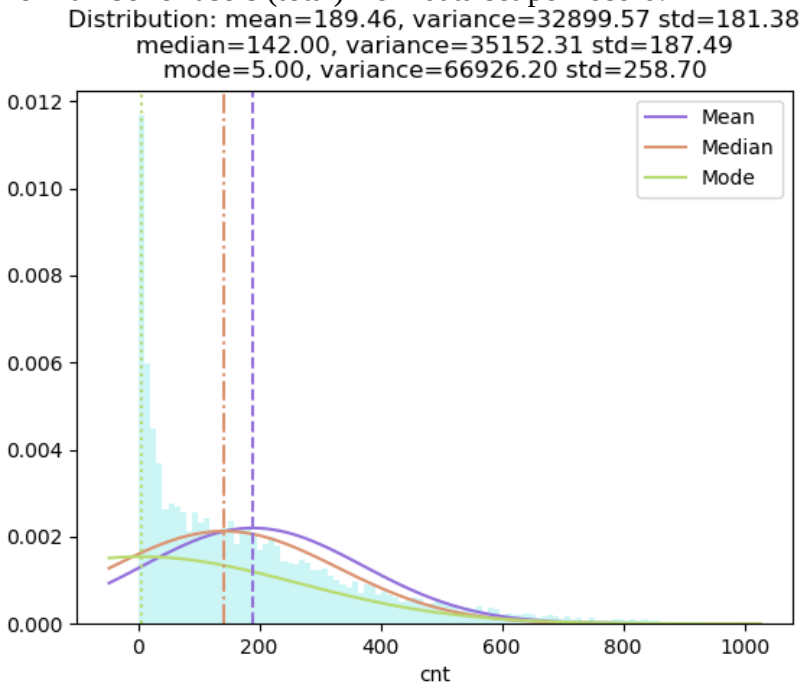
The data printable to screen based on this includes the mean, median and mode calculations for the selected column. For each of these the variance and standard deviation is also printed.

Sample print for total number of users from data set per record:

```
header 16: cnt - int
Please select the column (integer/float):16
For Mean: The Standard Deviation is 181.38, The Variance is 32899.57 and the Mean in 189.46
For Median: The Standard Deviation is 187.49, The Variance is 35152.31 and the Median in 142.00
For Mode: The Standard Deviation is 258.70, The Variance is 66926.20 and the Mode in 5.00
```

The graphs also display the previously mentioned values as text along with a histogram chart that has the distribution for each average overlaid above it.

Sample graph for number of users (total) from data set per record:



Linear Regression

For linear regression the predicted y values and the linear function to calculate them can be printed on screen. When exporting, these values will be outputted to a csv and txt file respectively.

Csv Output:

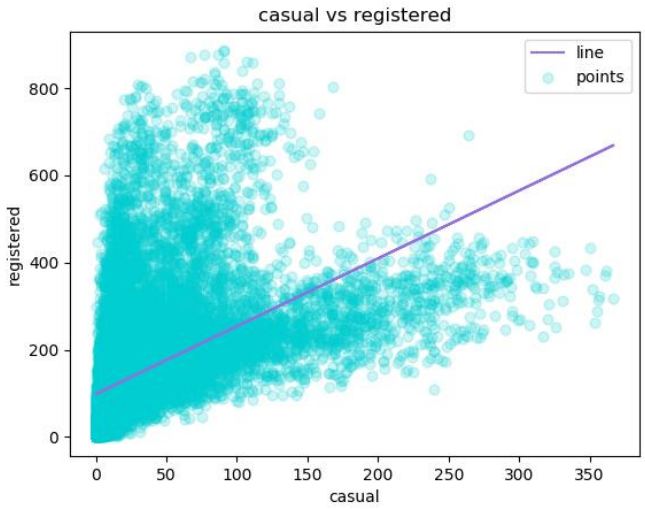
```
casual,registered,Predicted Y
3,3,102.9680873780252
8,32,110.74419861153669
5,27,106.0785318714298
3,10,102.9680873780252
0,1,98.30242063791832
0,1,98.30242063791832
2,0,101.41286513132292
1,2,99.85764288462062
1,7,99.85764288462062
8,6,110.74419861153669
12,24,116.96508759834587
26,30,138.73819905217803
29,55,143.40386579228493
```

Formula Output:

```
Formula: Y = 1.5552222467022967 * X + 98.30242063791832, slope of the line: 1.5552222467022967, Y Intercept of the Line 98.30242063791832
```

The graphs for linear regression consist of two components. A scatter plot, which plots the x and y values against each other and a line which is plotted by using the x values and predicted y values using linear regression.

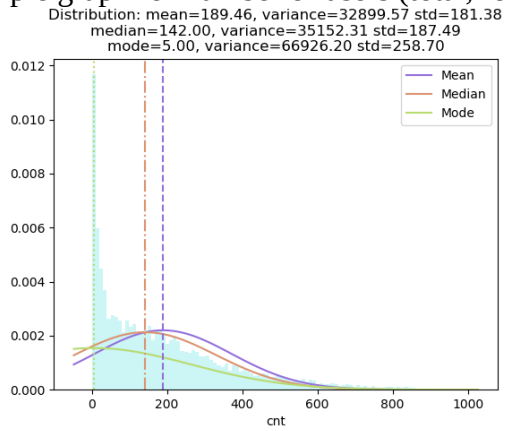
Plot of Casual vs Registered bike users:

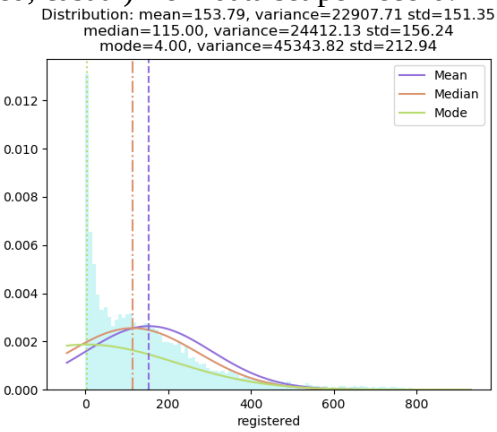


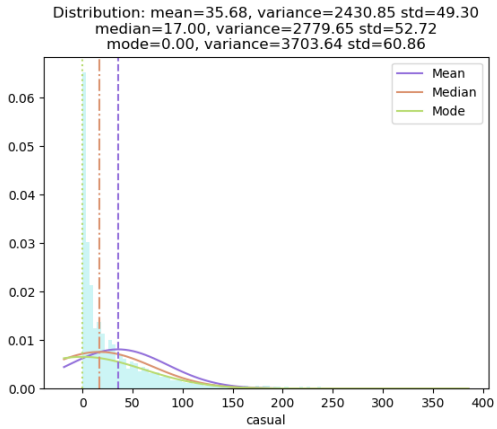
Conclusion

Normal Distribution.

Sample graph for number of users (total, registered, casual) from data set per record:

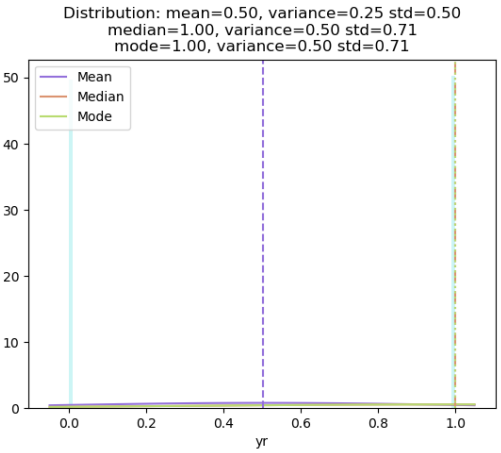






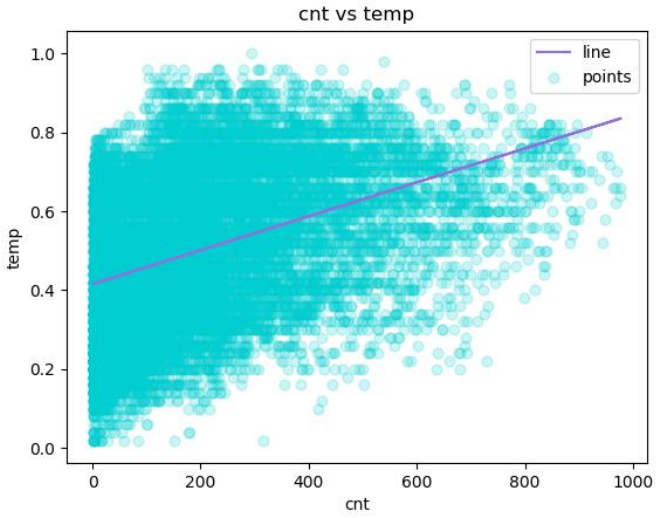
From samples like the above we can see that more registered users use the service then casual ones which gives us quick and interesting insights into the dataset and how the columns are distributed. This can be very useful when we quickly need to analyse a column.

On the other hand, we can end up with graphs like below which don't really tell us anything at all about the dataset. This is because the program is generic enough to be able to plot any integer/float values. In these cases, we need human expertise to figure out if the output is useful or not.

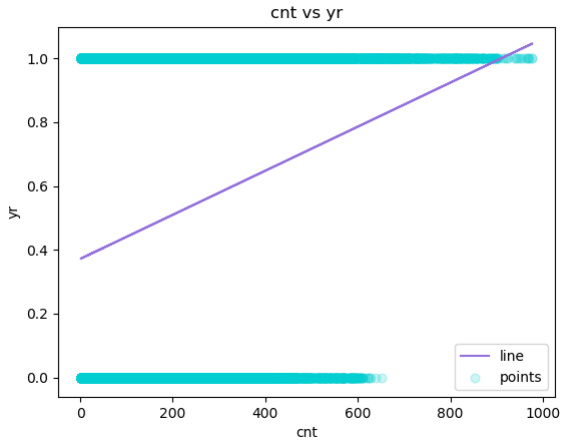


Linear Regression

Running linear regression on this data set can show us some interesting increases and decreases in data between different plots. For example, the graph below tells us that there is a positive correlation between the amount of bike users using the temperature. The higher the temperature the more users.



Once again like in normal distribution since we can plot any values as x and y columns we can come up with some illegible graphs, like total users vs year. Since there are only two values for year (0=2011, 1=2012) we see the influx of users, but the graph is very confusing at best. It takes a lot of effort to figure out what some graphs are trying to say since at first glance the line seems to be drawn randomly.



Overall Conclusion

In conclusion, from the output I can see a few interesting correlations. Like the increase in users over time or how weather effects the use of the bike sharing services.

The ability to load in any data set is also very useful as allows for analysing most data available. This is great for a quick analysis but if you want to see some more detailed correlations algorithms like multiple polynomial regression would be better suited. But using such an algorithm requires customised code every time which takes extra time.

Appendix 1: Reflective Learning Log

The work log follows the timing from the git repo below:
https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/
All logs will have a link to the commit they are referring to. They are also numbers so 1 in Understanding Achieved matches 1 in Work Completed

Log #1

Date: Oct 16, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/e43bc1dbb2f1335530dba162b4d79ac85391325c
Work Completed:

1. Set up git project – ReadMe, git ignore for python.
2. Added main.py with support for loading in a simple csv.

Understanding Achieved:

1. The git project would allow me to work on most machines that have the correct version of python. The git ignore file created with the help of Spyder would also stop me from committing junk to the repo.
2. I was able to load a csv into a dictionary using fileIO. The dictionary had column names as keys and values stored as a list. This allowed for processing of columns easily and efficiently as now I can loop over and run calculations easily on these.

Log #2

Date: Oct 26, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/56f4084d58c6a9cd22bc318a5af93738c061680c
Work Completed:

1. Added DataFrame.py class
2. Moved csv loading to the new class
3. Moved to working with PyCharm instead of Sypder

Understanding Achieved:

1. The DataFrame class was my first step to adding modularity to the code. As part of my design I decided that all processing should be handled by this. To do this I had to research how oop works in python as this wasn't part of the course material.
2. Since all processing will be handled by this class, I moved the csv loading into it to create the first function. Then instantiated and ran it using main.py.

Log #3

Date: Oct 26, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/e7106b696d8871b4b06d24f0a3f503dba83af6d9
Work Completed:

1. Added support for loading custom csv files using user input.
2. Selected a data set and decided what algorithms to run on it.
3. Added support to print default or custom data set to screen.

Understanding Achieved:

1. In main.py I created a menu that prompts the user to select a data set and enter the path if it's a custom one. While testing, I quickly found out that this processing can throw exceptions, so I put that on my to do list for later.

2. The data set I selected was about Bike sharing and I decided I wanted to run linear regression and find all the averages and graph them in an interesting way. To do this I found resource material to help me understand what I was trying to do. This can be found in the references.
3. Lastly to do something simple I printed the data set to the screen after it was selected by a user to make sure it gets loaded correctly. I also added support for custom data sets in case I ever wanted to change it. This meant that most of my code had to be generic enough to work with any csv which is something I hope to keep up as the work progresses.

Log #4

Date: Oct 26, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/7123a1c547f3ee1dc66b2232e8caec7e091ca7e6
Work Completed:

1. Added a separate menu to open and process data

Understanding Achieved:

1. When a data set was opened using the first menu, the user would be prompted on how to process it. I added some empty functions like run_linear_regression() with empty prints to test the menu worked correctly. I did this to help me figure out the end to end application flow and to divide the application into reusable functions. This would help me create an understanding for the application I was trying to create.

Log #5

Date: Oct 26, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/7ef606ce92274b99e642a600d5c40978a7025ea6
Work Completed:

1. Added TO DO statements,
2. Expanded menus to list all linear regression functions of program,
3. Added placeholder functions for the missing functionality.

Understanding Achieved:

1. The TO DO statements at the top of each file would help me keep track of my priorities. To do this I had to figure out how to comment python code.
2. I added any missing menu items to make sure all of the linear regression functionality is covered for. This let me open the main.py program and quickly test changes to DataFrame.py
3. I added more place holder functions to figure out how to split up the work.

Log #6

Date: Oct 29, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/5100266be6c849289acdd19dcef0079303e0dcf4
Work Completed:

1. Added support for user specified column typing (int/float/string)
2. Added utility class for checking user inputs
3. Added comments to functions in DataFrame class

Understanding Achieved:

1. When the user loads a data set, I needed to know what types of columns they are using. I can't just assume everything will be an integer. Therefore, I prompted to see what columns they are and loaded the data set using these rather than loading the data as strings which is what I did up until now. For the default data set I just hard coded this. Once again, I realised that incorrect conversions lead to exceptions, so I added that to my todo list.
2. I created TextUtils.py which is responsible for testing if the user input is what I expect it to be. If it isn't the user will get prompted to input again. This makes sure I only accept inputs my code can process like "yes", "no", "int", "float" or any other valid instance of these strings.
3. I added comments to my code to keep things clean and readable for anyone else using my application/code.

Log #7

Date: Oct 29, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/496e0e883738bbb62220bf66c6e4a197418a314e
Work Completed:

1. Implemented detailed header printing

Understanding Achieved:

1. I iterated over the headers in the dictionary and printed them nicely to the screen in the DataFrame class. Then I used this function in the menu in main.py so the user could run this to see the headers and their data types. I plan to use this when the user needs to select a column for processing.

Log #8

Date: Oct 26, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/0a9313b65565ea6ab613ea2258da3ee7d382535d
Work Completed:

1. Added exception handling for string to integer conversions
2. Updated TextUtils with comments
3. Made functions more generic in TextUtils.
4. TextUtils now prints menus

Understanding Achieved:

1. I implemented a new class called ExceptionUtils. This class will be responsible for any common functions that throw exceptions. For example, getting an integer using user input or converting a string to integer/float. This allows me to keep code more generic and to handle exceptions without the application breaking.
2. TextUtils now has comments explaining all the functions and their parameters for maintenance purposes.
3. I realised I was doing a lot of the same kind of processing in TextUtils to test for yes, no, int, string values. I figured I could make this more generic, so I created a function that takes a value and list. Then checks if the value is in the list. Now all the functions use the generic one to find the output. This has made the code a bit more readable too since before I checked against all the strings using a single if statement.
4. Lastly main was getting cluttered with having to handle menus so I added a function that can print an array of menu items and one that can get user input in one line. This

has made main easier to look at. I intend to turn those menus into constants later to pull them out of main and make it even more manageable.

Log #9

Date: Oct 30, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/624c364da29a50092687df2421a4335a47027f36
Work Completed:

1. Added 1st iteration of the linear regression algorithm
2. Added MathsUtil to process arrays easily
3. Added ability to plot the output from linear regression

Understanding Achieved:

1. I implemented the first iteration of the linear regression algorithm. This was able to predict the slope and y-intercept of the line. I used it to predict where the y values would be based on the line formula.
2. For the calculations I realised it would be very messy to keep the functions to calculate values based on arrays in the DataFrame class, so I created a MathsUtil. This new class can sum/multiply/add/average arrays and numbers together. It's my custom equivalent of numpy with only the functions I need.
3. Using the 2 columns and the predicted-y values I was able to do a scatter plot with a line going through it. To do this I had to look up the pyplot documentation as we didn't cover this in class at the time. The line wasn't displaying correctly which implied I did something wrong. It was always pointing away from the scatter plot which it should go straight through.

Log #10

Date: Oct 30, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/0679c830fddc4192c6492d1ad890356c5e802bd9
Work Completed:

1. Moved slope, y-intercept estimation functions to MathUtil
2. Added export for linear regression
3. Still trying to resolve issues with Linear Regression

Understanding Achieved:

1. The function for estimating slope and y-intercept seemed better suited to the MathUtil than the DataFrame class so I moved in there instead. This makes it more usable if I ever need to pull MathUtil to do linear calculations elsewhere.
2. The next step was to export the processed data so it can be embedded elsewhere if needed. I created an exported file for the graph, predicted-y values and the line formula. At this point I didn't do exception handling as I wasn't very familiar with python FileIO yet.
3. I still couldn't resolve this issue with the predicted-y values being incorrect after checking all the formulas.

Log #11

Date: Oct 30, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/92bd277be9a492551e6732909dff402a52423bfa
Work Completed:

1. Fixed linear regression issues
2. Made minor tweaks to paths and moved some generic code to the correct places

Understanding Achieved:

1. I buckled down and went through the formulas manually on 2 columns with 4 rows each. Then used the debugger to compare my results with the calculations of my program. When I got to the output of the slope and y-intercept functions I realised that they were flipped. My slope was the y-intercept and vice versa. I changed the names for the functions around and it worked. The graph was showing correctly too.
2. After fixing the algorithm I did some tidying up like changing the output directories, adding more comments, making some functions more generic etc.

Log #12

Date: Nov 05, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/865e15ccb025a97b6e75b1aabb0d62a48e0dbab5
Work Completed:

1. Completed User Testing and fixed issues.
2. Decreased duplication

Understanding Achieved:

1. I asked a friend to do some user testing. The person was unfamiliar with the project. Once we completed this I then fixed any issues the person found. This is detailed in the testing section of the document.
2. Once again to make the code more reusable I made functions more generic and reusable.

Log #13

Date: Nov 05, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/5d7f6b0263e12ca9f531d89fbd92927e8dee77e4
Work Completed:

5. Added extra comments to all files.

Understanding Achieved:

4. After I was done, I realised that the format for the comments wasn't to the python specification thanks to my editor giving me a few small warnings and that the comments should be inside of the methods rather than above them. Since it would take time to fix this, I decided to keep the format I already had

Log #14

Date: Nov 05, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/2e5d8b28f2faf670aad7ad53958353076e1c76c7
Work Completed:

1. Removed Duplication
2. Added export all for linear regression

Understanding Achieved:

1. To remove more duplication and simplify the DataFrame class, I created a GraphUtil. This handles all the rendering and exporting of graphs using pyplot. The GraphUtil has a common function for creating graphs used both by the show on screen, export and export all features.

2. I also added the ability to export all possible graph combinations for linear regression since it can take a lot of time to go through all the columns to export everything you might need. I added this as I noticed that during the user test it took my friend too long to export the graphs they wanted.

Log #15

Date: Nov 05, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/e91cd1bb811241d5fe5035fca59996b62d9a3b09
Work Completed:

1. Added graceful handling of fileIO

Understanding Achieved:

1. Gracefully handing FileIO exceptions has been on my backlog for the longest time. This is the easiest way to break my program since I started handling all conversion exceptions. I covered all the fileIO code with try catch statements. I waited for this to be covered in class as I wasn't sure how to approach this. This was helpful as I learned about the "with" keyword which handles IO in a way when everything gets gleaned up if an Exception was thrown.

Log #16

Date: Nov 05, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/cabba8bb1729e6317b34f8dedb3217902b5960be
Work Completed:

1. Created constants.py
2. Fixed issue with default data types being set to int by default
3. Added custom delimiter support

Understanding Achieved:

3. I created a constants file that contains the strings used in the application. This allows me to go into one file and change settings from there rather then search for it in code. If used well it can also make the code more readable as it removes random values and gives them a name instead. To do this I had to go research the standard for constant files. Which has the naming convention that uses capitals and underscores for the constants i.e. DATE_FOTMAT.
4. When loading data sets, I had a feature which would prompt the user to see if all columns are integers. At some point I broke this and the data set dint load. To fix this I needed to update an if statement that checks if the data types are correct. This was the point where I learned that I can't assume my code will always work when I make changes. It made me decide to add some tests soon.
5. I added support for inputting a custom delimiter when loading a data set since csv files can be split in different ways.

Log #17

Date: Nov 8, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/38e0d5bdc47ce55787acff0ff24195a881434939
Work Completed:

1. Added tests for Exception Utils.

Understanding Achieved:

1. I added most of the tests for the ExceptionUtils class to make sure it works correctly. I decided to use the unittest library and I was running these tests through the PyCharm IDE.

Log #18

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/acecccc7f0e02dc535f297e90c416d6545819fdd
Work Completed:

1. Updated project structure
2. Fixed names for some ExceptionUtils tests

Understanding Achieved:

1. I ran my code on a different device with an older version of python 3.7, it was giving me issues so updated the path structure for my code. It seemed to work on that device after that, but the IDE was still complaining about imports not being correct.
2. While running my tests together I realised some of them had the same name, so they weren't running correctly as the number of passed tests didn't match the number of tests, I had written so I changed them and now all of them work.

Log #19

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/55a8f73e25cce7c7d7d89c91b392e82dc8779a56
Work Completed:

1. Updated imports
2. Creates helper for terminal input and output

Understanding Achieved:

6. To resolve the import errors I seen I needed to update the imports with the src.<class> path and __init__.py files to directories with the main() function in them. This fixed any import issues I had on both machines I used for this project. I had to put in a lot of effort to figure out how python recognises paths/local imports correctly.
7. I created a utility class for tests that can mock input and read printed output. This will simplify my tests especially since I already had to use this kind of code in the ExceptionUtils tests. I updated the existing tests to use the utility class too.

Log #20

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/3779ec97d93473048eb0d9a4ea56faee68396a13
Work Completed:

1. Added tests for the MathsUtils

Understanding Achieved:

1. I added tests to cover all the MathUtils.py functions. While writing these I figured out the slope and y-intercept functions were the wrong way around. This was because of the fix I applied before. Instead of changing the names I needed to change the y-prediction function instead. Everything was working correctly it was just named wrong.

Log #21

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/3b6574034bf8a98cf116b360e01ecaf165e547cf
Work Completed:

1. Added tests for TextUtils
2. Updated all tests and test files to start with test_
3. Updated files to use lower case and underscores

Understanding Achieved:

1. I added tests to cover all the text_utils.py functions. These were the simplest tests I had to write
2. I changed all the tests and test files to start with test_. This is because when I tried to run my tests with pytest none of them were being picked up. To figure this out I had to read up on the documentation for pytest first.
3. After changing the test file names, I decided to change the other ones too. This would make everything a bit more consistent.

Log #22

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/37215c1a6ce552b4b65ca508bf03cb23c6a9e3a3
Work Completed:

1. Added tests for data_frame.py & graph_tils.py

Understanding Achieved:

1. I added tests to cover a of functionality from the DataFrame and GraphUtils class. While writing a test to make sure faulty rows get dropped from the data frame, I noticed some values weren't dropped correctly so I had to fix this, but I was able to use the test to help me with it.

Log #23

Date: Nov 11, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/29ef79b06da042b3b8f27b0a65bc637e2b66cd67
Work Completed:

1. Updated directories to use lower case and underscores

Log #24

Date: Nov 14, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/cf7fde46e3d993818ec1b70381108fb6e7ca13c1
Work Completed:

1. Added support for Normal Distributions

Understanding Achieved:

1. Using the knowledge, I gathered and the design I implemented I added normal distribution graphs based on mean. I needed the same functionality including printing, graphing and exporting. It was simpler this time as I already had most of the base functionality I needed. I also added extra tests for the DataFrame class to make sure the new functionality works.

Log #25

Date: Nov 22, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/61fb09f27953236d7510975d40b41dc393191304
Work Completed:

1. Fixed plotting of graph for linear regression
2. Added median and mode to normal distribution plot
3. Added a line to show mean, median and mode to the ploy.

Understanding Achieved:

1. The linear regression plot was causing an infinite look because I called the wrong method in it.
2. To add more to the normal distribution added support for median and mode. This allowed me to print extra statistics about the data set to evaluate it better. From that alone I learned that the mean and median tend to be very similar, but the mode is way different for most.
3. I wanted to be able to visually evaluate these averages too, so I plotted the normal curves too. But to see the exact point for the averages I needed to do a bit of research. I found the axvline function which let me plot a line through the x axis.

Log #26

Date: Nov 23, 2019
Commit: https://github.com/DanielsHappyWorks/ASL-CA1-Data-Science-from-Scratch/commit/996b6261e0a3bccf44105317ddbcb1200ed6ee06
Work Completed:

1. Completed User Testing and fixed issues.

Understanding Achieved:

2. I asked the same friend to do some user testing again. Once we completed this I then fixed any issues the person found. This is detailed in the testing section of the document.

Appendix 2: References

1. Fanaee-T, H. (2019). *UCI Machine Learning Repository: Bike Sharing Dataset Data Set*. [online] Archive.ics.uci.edu. Available at: https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset [Accessed 18 Nov. 2019].

2. GeeksforGeeks. (2019). *Linear Regression (Python Implementation) - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/linear-regression-python-implementation/ [Accessed 18 Nov. 2019].

3. En.wikipedia.org. (2019). *Simple linear regression*. [online] Available at: https://en.wikipedia.org/wiki/Simple_linear_regression [Accessed 18 Nov. 2019].

4. Mathsisfun.com. (2019). *Normal Distribution*. [online] Available at: https://www.mathsisfun.com/data/standard-normal-distribution.html [Accessed 18 Nov. 2019].