ECS765P-Big Data Processing Coursework

By - Satyam Sharma (220760793)

PART A - TIME ANALYSIS

Objective A1 - Create a bar plot showing the number of transactions occuring every month between the start and end of the dataset.

Source Code files relevant to Objective A1

transactions.py

Execution Command

ccc create spark transactions.py -d -s

Command to save file to storage

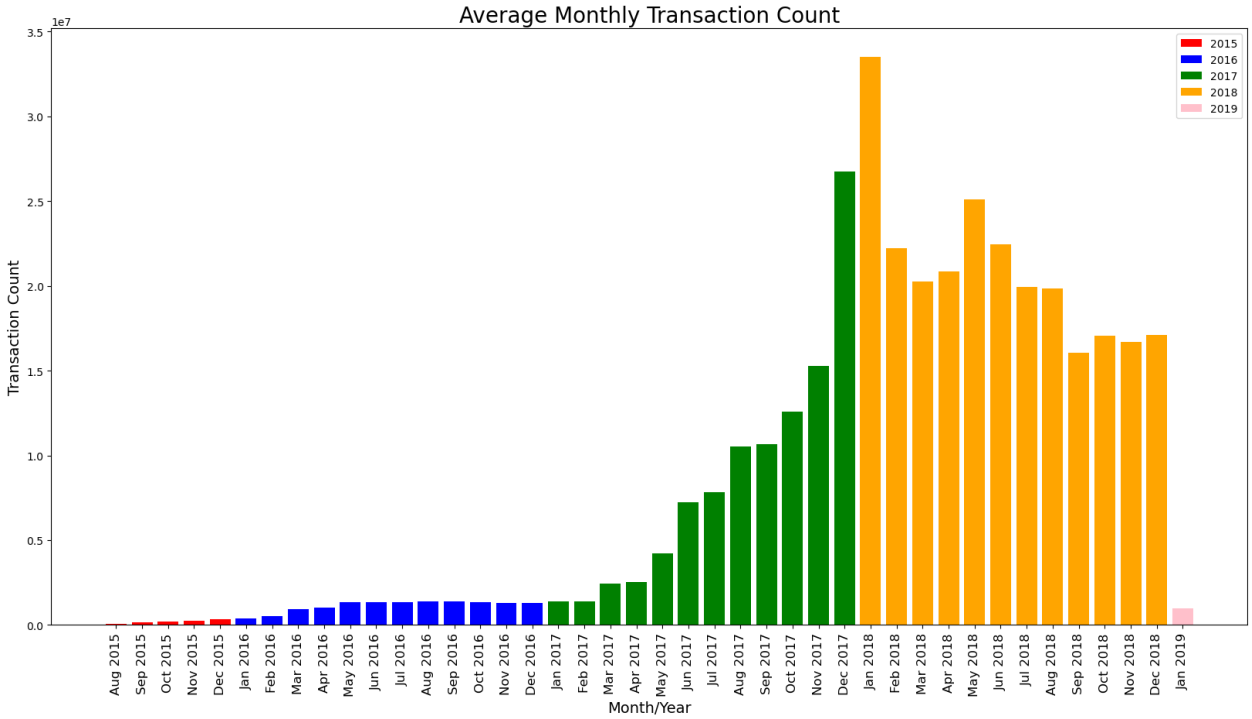
ccc method bucket cp bkt:part1a_/ ~/teaching_material/ECS765P/Big_Data_Coursework/

Output File

total_transactions.txt

Explanation

The first task involves analyzing a dataset of Ethereum transactions to determine the average monthly transaction count. To achieve this, a Spark session is created and functions are defined to filter out invalid data and extract the month and year from each transaction timestamp. The dataset is loaded from an S3 bucket and the filter and processing functions are applied using Spark transformations. The transaction counts are then aggregated by month and year using the reduceByKey function. The total number of transactions each month is saved in a text file called "total_transactions.txt" The results are then visualized using a bar plot created with Matplotlib in python to show the Average Monthly Transaction Count.



PART A - TIME ANALYSIS

Objective A2 - Create a bar plot showing the average value of transactions occuring in each month between the start and end of the dataset.

Source Code files relevant to Objective A2

averagetransactions.py

Execution Command

ccc create spark averagetransactions.py -d -s

Command to save file to storage

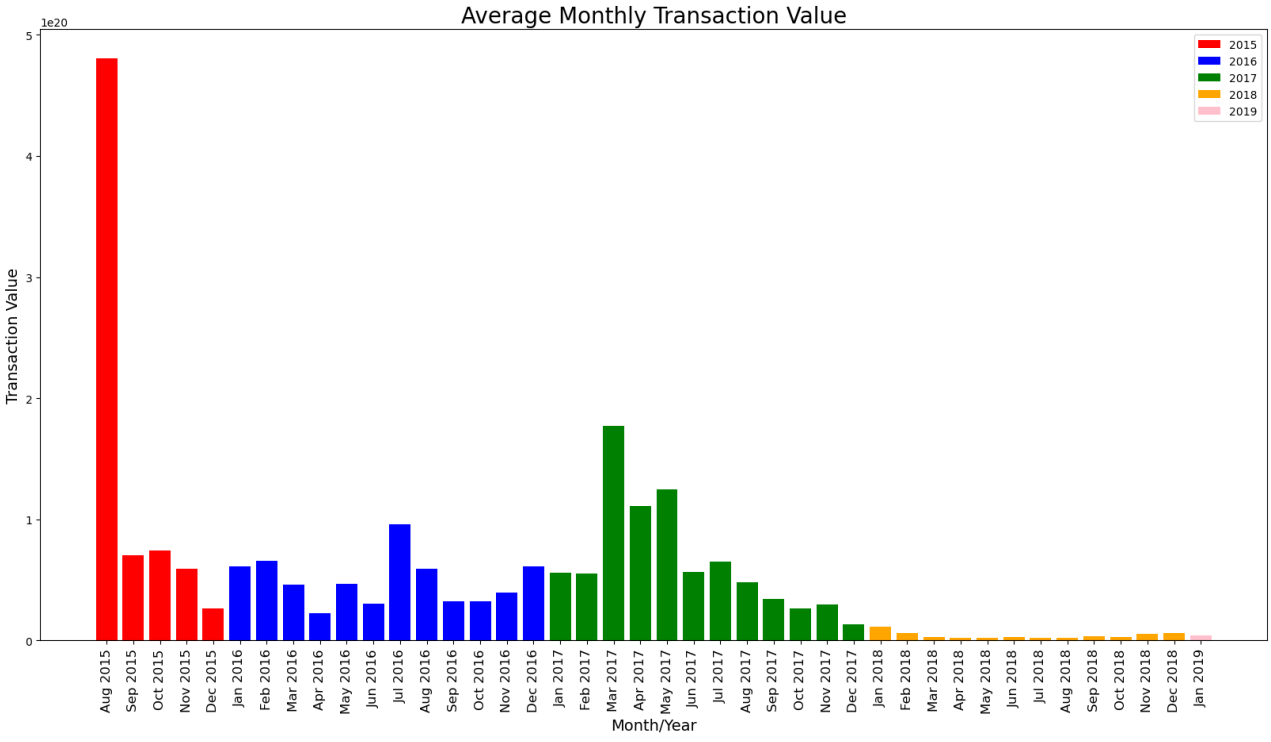
ccc method bucket cp bkt:part1b_/ ~/teaching_material/ECS765P/Big_Data_Coursework/

Output File

average_transactions.txt

Explanation

The second task involves analyzing the dataset of Ethereum transactions to determine the average value of transactions. To achieve this, a Spark session is created and functions are defined to filter out invalid data and extract the month and year from each transaction timestamp. The dataset is loaded from an S3 bucket and the filter and processing functions are applied using Spark transformations.The average transactions are then aggregated by month and year using the reduceByKey function. The average transaction value each month is saved in a text file called "average_transactions.txt" The results are then visualized using a bar plot created with Matplotlib in python to show the Average Monthly Transaction Value.



PART B - TEN MOST POPULAR SERVICES

Objective B - Evaluate the top 10 smart contracts by total ether received.

Source Code files relevant to Objective B

topservices.py

Execution Command

ccc create spark topservices.py -d -s

Command to save file to storage

ccc method bucket cp bkt:partb_/ ~/teaching_material/ECS765P/Big_Data_Coursework/

Output File

top_10_services.txt

Explanation

In this task we evaluate the top 10 smart contracts by the total Ethereum received, we first read both the contracts and transactions dataset. We then map the to_address and value from the transactions dataset, while the address and count are mapped from the contracts dataset. The datasets are then joined using the .join() function with to_address and address as the joining criteria. The resulting addresses and Ethereum values are mapped together, and we obtain the top 10 smart contracts using the takeOrdered function. The output is saved in a text file named "top_10_services.txt" and loaded into pandas dataframe using python to show the output below.

Rank	ID	Ethereum Value
1	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	84155363699941767867374641
2	0x7727e5113d1d161373623e5f49fd568b4f543a9e	45627128512915344587749920
3	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	42552989136413198919298969
4	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	21104195138093660050000000
5	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	15543077635263742254719409
6	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	10719485945628946136524680
7	0x341e790174e3a4d35b65fdc067b6b5634a61caea	8379000751917755624057500
8	0x58ae42a38d6b33a1e31492b60465fa80da595755	2902709187105736532863818
9	0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3	1238086114520042000000000
10	0xe28e72fcf78647adce1f1252f240bbfaebd63bcc	1172426432515823142714582

PART C - TOP TEN MOST ACTIVE MINERS

Objective C - Evaluate the top 10 miners by the size of the blocks mined.

Source Code files relevant to Objective C

topminers.py

Execution Command

ccc create spark topminers.py -d -s

Command to save file to storage

ccc method bucket cp bkt:partc_/ ~/teaching_material/ECS765P/Big_Data_Coursework/

Output File

top_10_miners.txt

Explanation

In this task we only require the blocks dataset. The 'miner' and 'size' data are then combined and reduced using the reduceByKey function to obtain the block size. The top 10 miners are extracted using the takeOrdered function and saved as a text file named "top10_miners.txt". The resulting text file was loaded into a pandas dataframe using Python.

Rank	Miners	Block Size
1	0xea674fdde714fd979de3edf0f56aa9716b898ec8	17453393724
2	0x829bd824b016326a401d083b33d092293333a830	12310472526
3	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	8825710065
4	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	8451574409
5	0xb2930b35844a230f00e51431acae96fe543a0347	6614130661
6	0x2a65aca4d5fc5b5c859090a6c34d164135398226	3173096011
7	0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1152847020
8	0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1134151226
9	0x1e9939daaad6924ad004c2560e90804164900341	1080436358
10	0x61c808d82a3ac53231750dadc13c777b59310bd9	692942577

PART D - DATA EXPLORATION

Objective 1 - Utilising the provided scam dataset, what is the most lucrative form of scam? How does this change throughout time, and does this correlate with certain known scams going offline/inactive? To obtain the marks for this catergory you should provide the id of the most lucrative scam and a graph showing how the ether received has changed over time for the dataset.

Source Code files relevant to Objective D1

scams.py

conversion.py

Execution Command

ccc create spark scams.py -d -s

Command to save file to storage

ccc method bucket cp bkt:partd_scams_/

~/teaching_material/ECS765P/Big_Data_Coursework/

Output File

most_lucrative_scam_forms.txt

ether_received_over_time.txt

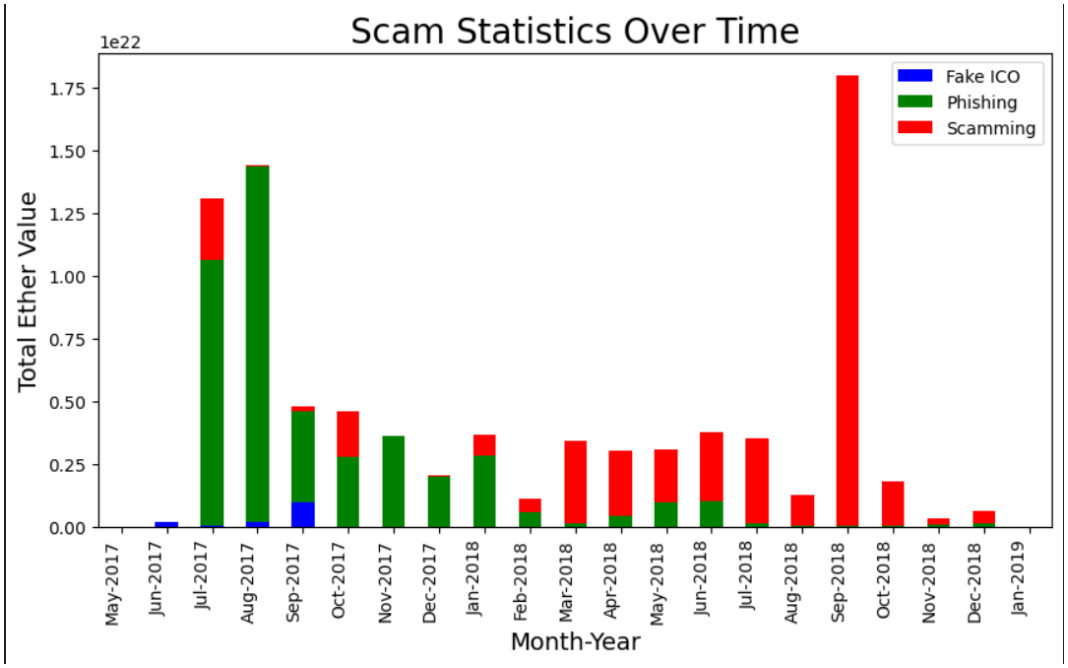
Explanation

In this task, we first convert the scams.json file to a csv file, which is accomplished through the use of the conversion.py script. The resulting scams.csv file contains several parameters, including identity, name, url, coin, category, subcategory, index, and status. This file is then uploaded to the data repository bucket for future use.

Next, we need to identify the most profitable scams by obtaining their IDs. This involves reading both the scams.csv and transactions.csv files and mapping the index, ID, and category from the former, and the address and ether value from the latter. The two datasets are then joined based on their common addresses using the .join() function, and the resulting dataset is reduced using the reduceByKey function to obtain the total ether profit for each scam, which is then mapped with the ID and scam type as the key and total ether profit as the value. The top 15 most profitable scams are then identified using the takeOrdered function, and their output is saved in the "most_lucrative_scam_forms.txt" file and we use pandas dataframe in python to display the results.

Finally, we need to analyze how the total ether received changed over time. To do this, we use the transactions.csv and scams.csv files, mapping the index and category from the latter and the address, date, and value from the former. The datasets are then joined using the .join() function, and the resulting dataset is mapped with the date and scam type as the key and ether value as the value. The data is then reduced using the reduceByKey function to obtain the total ether received for each month, and the output is saved in the " ether_received_over_time.txt" file. The visualisation of total ether value received over time is plotted using Matplotlib in python and the graph is titled - "Scam Statistics Over Time".

Rank	Scam Id	Type of Scam	Ether Value
1	5622	Scamming	1.670908e+22
2	2135	Phishing	6.583972e+21
3	90	Phishing	5.972590e+21
4	2258	Phishing	3.462808e+21
5	2137	Phishing	3.389914e+21
6	2132	Scamming	2.428075e+21
7	88	Phishing	2.067751e+21
8	2358	Scamming	1.835177e+21
9	2556	Phishing	1.803047e+21
10	1200	Phishing	1.630577e+21
11	2181	Phishing	1.163904e+21
12	41	Fake ICO	1.151303e+21
13	5820	Scamming	1.133973e+21
14	86	Phishing	8.944561e+20
15	2193	Phishing	8.827100e+20



Deductions from the Visualisation

The most profitable scam in August 2017 was "Phishing", whereas in September 2018 it was "Scamming". "Fake IPO" remains pretty low in contrast to the other two scams. It is noticeable that during the period from July to September in years 2017 and 2018, there was a noticeable surge in Ethereum scams.

PART D - DATA EXPLORATION

Objective 2 - For any transaction on Ethereum a user must supply gas. How has gas price changed over time? Have contracts become more complicated, requiring more gas, or less so? How does this correlate with your results seen within Part B. To obtain these marks you should provide a graph showing how gas price has changed over time, a graph showing how gas used for contract transactions has changed over time and identify if the most popular contracts use more or less than the average gas_used.

Source Code files relevant to Objective D2

gasguzzlers.py

Execution Command

ccc create spark gasguzzlers.py -d -s

Command to save file to storage

ccc method bucket cp bkt:partd_gg_/ ~/teaching_material/ECS765P/Big_Data_Coursework/

Output Files

average_gas_price.txt

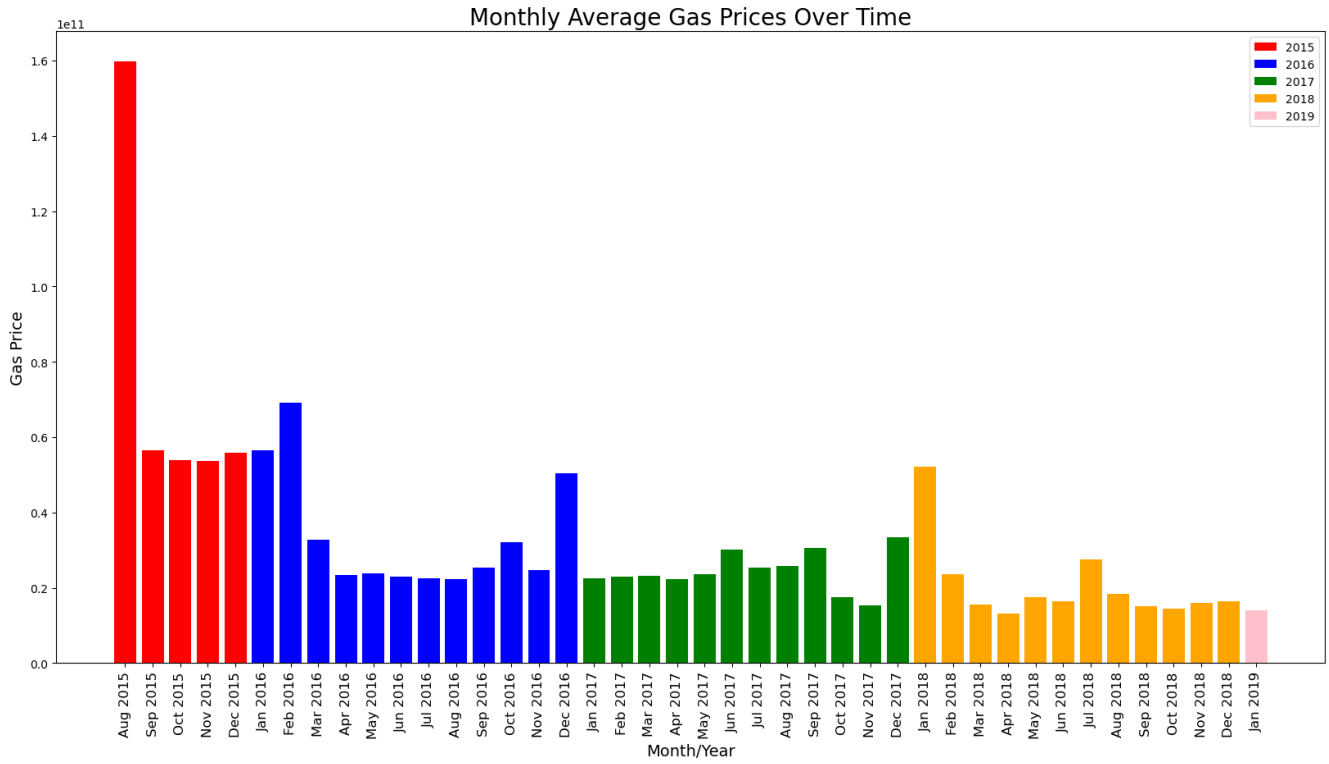
average_gas_used.txt

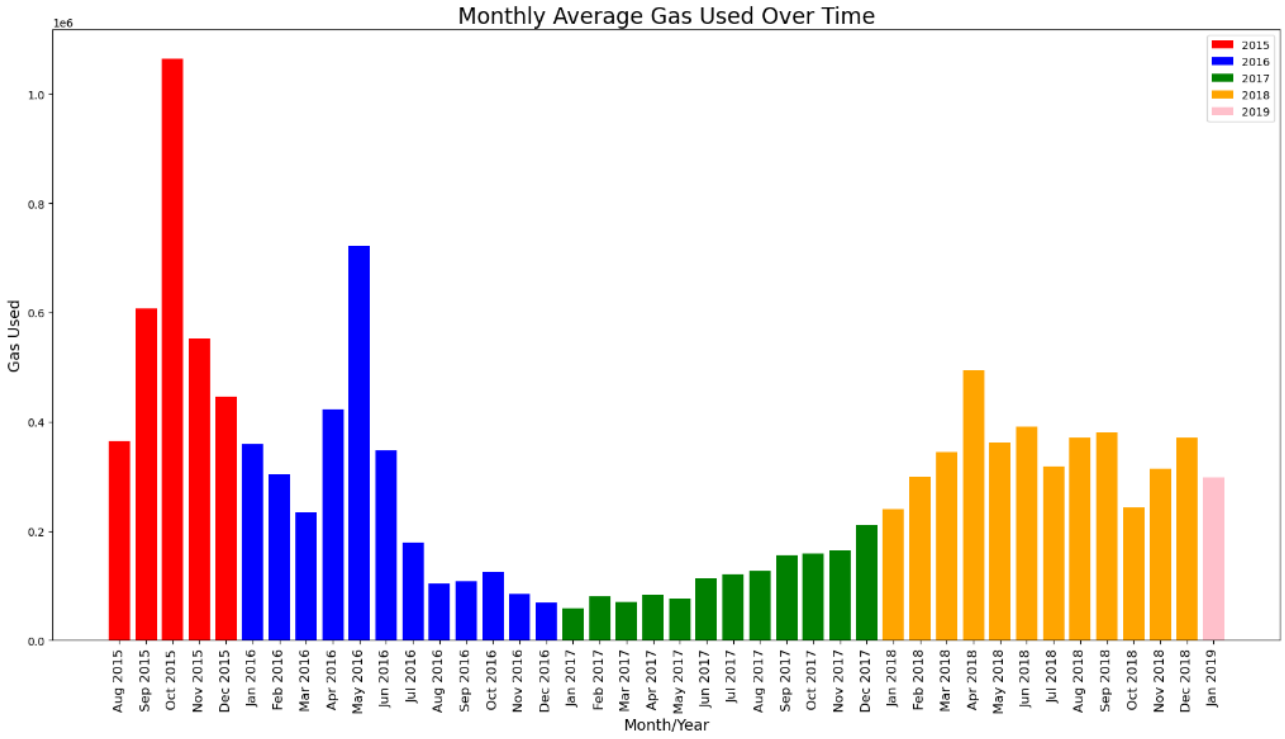
Explanation

Here in this task, we have two datasets: transactions.csv and contracts.csv. To obtain the average gas price change over time, the date is mapped as the key and the gas price and count as the value from transactions.csv dataset. The mapped function is then reduced using the reduceByKey function to get the total gas price and count. The total gas price is divided by the total count to obtain the average gas price, which is mapped to the date to represent the average gas price each month. The output is saved in "average_gas_price.txt" file, and the data is plotted using Matplotlib library in python.

In order to obtain the average gas used over time, the to_address is mapped as the key and the date and gas are mapped as the value from transactions.csv dataset. The address is mapped as key and count as the value from contracts.csv dataset. Both datasets are joined using the .join() function. After joining, the date is mapped as key and the gas used and count as the value. The mapped function is then reduced using the reduceByKey function and is divided by the total count to obtain the average gas used, which is mapped to the date to represent the average gas used. The output is saved in " average_gas_used.txt" file, and the data is plotted using Matplotlib library in python.

The two visualisations are shown on the next page -





Deductions from these Visualisations

Looking at the average gas price data between 2015 and 2018, we can observe a general downward trend with some noticeable variations. In August 2015, the gas price spiked to an exceptionally high level but gradually decreased in the following months. The year 2016 saw a wider range of gas prices, while in 2017, the prices remained within a narrower scope with two high values. Moving to 2018, the year began with relatively elevated prices that gradually declined towards the end of the year.

In contrast, when examining the average gas usage data during the same time period, fluctuations are observed. In 2015, there was a significant increase in gas usage especially in October 2015. In 2016, there were fluctuations as well, with the highest gas usage occurring in May. In 2018, gas usage was more volatile, with an increasing pattern from January till the end of the year. In January 2019, the prices drop from December 2018 but is still high.