ECS765P: Analysis of Ethereum Transactions and Smart Contracts

Name: Eleanor Prashamshini

Student ID: 220772291

Course: Big Data Processing

Course Code: ECS765P

Introduction

This report explains the data processing methodologies used to explore and analyse transactions on the Ethereum Blockchain Network from August 2015 till January 2019. There are three essential datasets - "blocks.csv", "transactions.csv" and "contracts.csv". Additionally, there is a set of active and inactive scams on the network in "scams.json". All the data is stored on the S3 repository bucket "/data-repository-bkt/ECS765/ethereum-parvulus".

Setup

First, we need to setup a connection between Spark and the data bucket to be able to read the data. This involves –

- Creating a Spark session.
- Setting environment variables for the S3 bucket such as the bucket name, access keys, and endpoint URL.
- Accessing and setting the Hadoop configuration properties of the Spark context to configure the S3 connection.

These steps are repeated at the beginning of every program.

Part A: Time Analysis

To explore the nature of transactions over time, we will observe the trend in number of transactions and average value of transactions.

Methodology

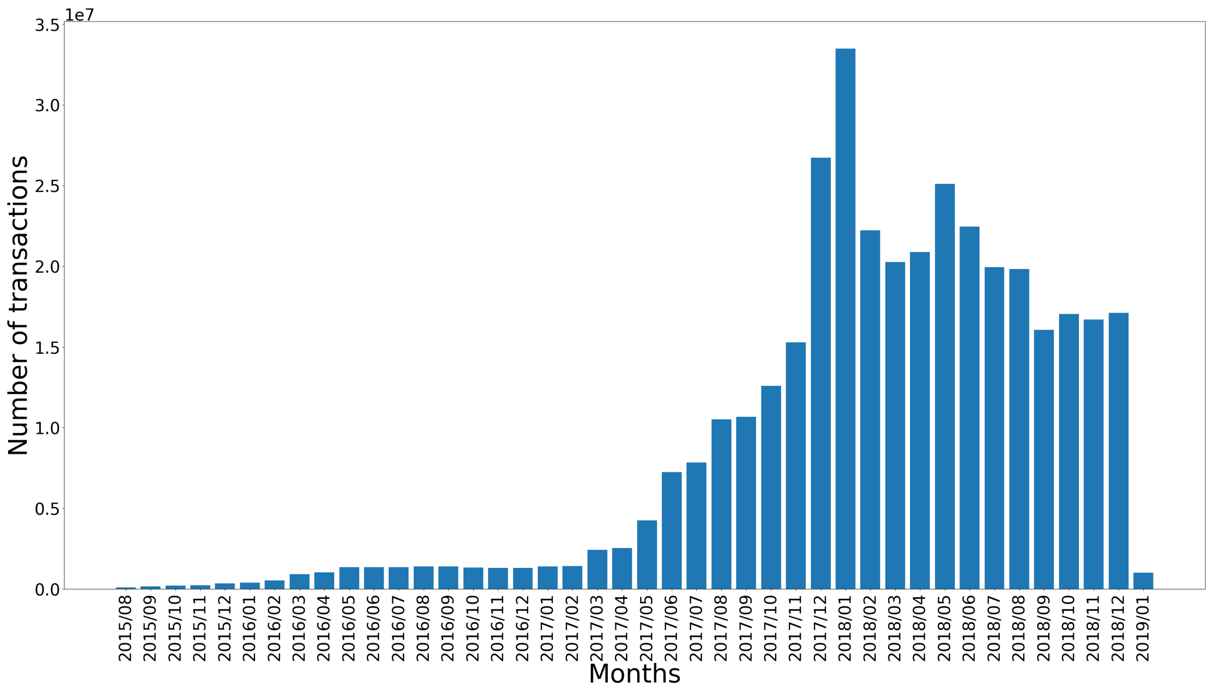
- Step 1: Read all transaction lines from "transactions.csv" and clean them. This ensures that a good transaction has 15 fields as expected, and the fields 'value' (index: 7) and 'block_timestamp' (index: 11) are in the required format.
- Step 2: Map the transactions with 'block_timestamp', in "YYYY/MM" format, as key and a tuple of 'value' and count 1 for each transaction as value.
- Step 3: Sum all the transaction values and transaction counts for each month.

- Step 4: Average the transaction value for each month by dividing the total value by number of transactions.

Results

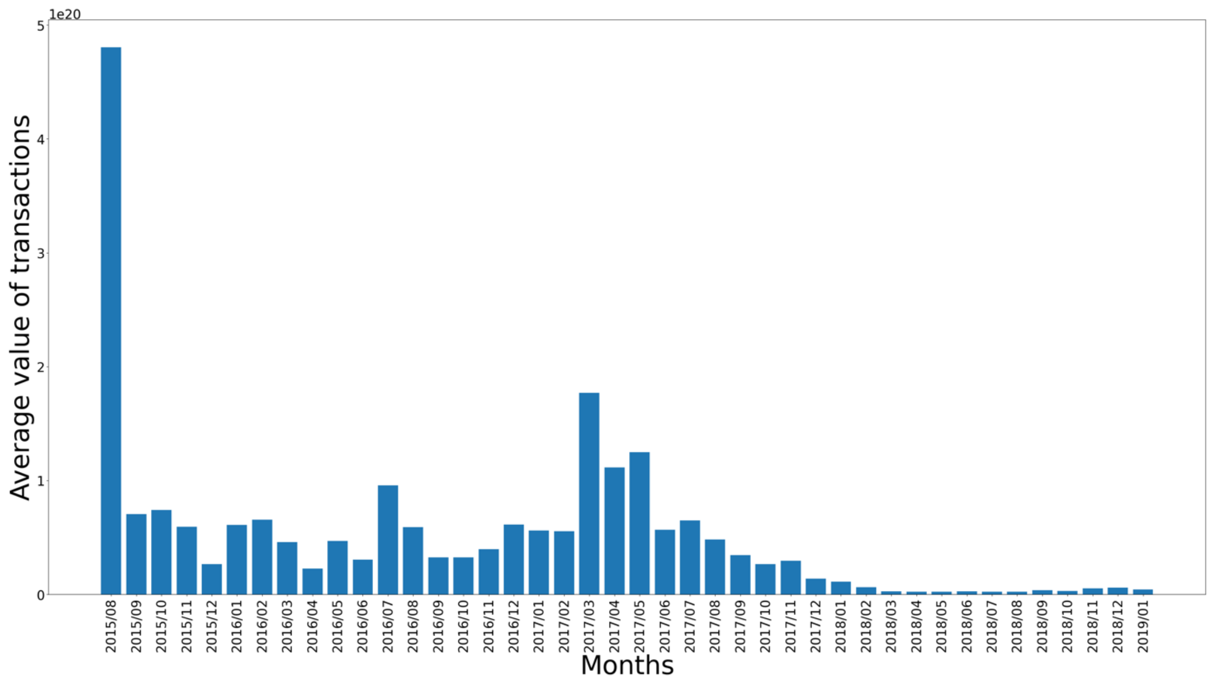
A.1: Number of transactions every month

It can be observed that the Ethereum Network saw an exponential increase in the number of transactions till January 2018, signifying its rise to fame and increase in usage. In the subsequent months, there appears to be notable fluctuations each month in the number of transactions.



A.2: Average value of transactions every month

Overall, we observe a decrease in the average value of transactions over time. Correlating this behaviour with the previous plot suggests an inverse relationship between number of transactions and average value of transaction, that is, as the number of transactions has increased the average value of each transaction has decreased over the months.



Part B: Top Ten Most Popular Services

To identify the most popular services (smart contracts), we calculate the sum of Ether received by addresses.

Methodology

- Step 1: Read transaction lines from "transactions.csv" and clean them, such that every line has 15 fields including a populated 'to_address' field (index: 6) and 'value' (index: 7) in the required format.
- Step 2: Map 'to_address' with 'value' for all transactions, and sum all the values for each address.
- Step 3: Read all contract lines from "contracts.csv". Check if there are 6 fields in each line and if 'address' (index: 0) is present and not a header.
- Step 4: Extract all the addresses using a map function.
- Step 5: Join the transactions RDD and contracts RDD.
- Step 6: Lastly, map the results in the format of contract and value, and further, order them in descending order of value to get the top 10 services.

Results

	CONTRACT ADDRESS	**TOTAL VALUE**
1	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	8.41553636999415e+25
2	0x7727e5113d1d161373623e5f49fd568b4f543a9e	4.562712851291527e+25

3	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	4.255298913641317e+25
4	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	2.1104195138093656e+25
5	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	1.5543077635263766e+25
6	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	1.0719485945628915e+25
7	0x341e790174e3a4d35b65fdc067b6b5634a61caea	8.379000751917755e+24
8	0x58ae42a38d6b33a1e31492b60465fa80da595755	2.902709187105735e+24
9	0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3	1.23808611452004e+24
10	0xe28e72fcf78647adce1f1252f240bbfaebd63bcc	1.1724264325158215e+24

Part C: Top Ten Most Active Miners

To recognise the most active miners, we compute the total size of all the blocks mined by the miner.

Methodology

- Step 1: Read all block data from "blocks.csv" and filter the good lines which have 19 fields, where the 'miner' (index: 9) field has value and 'size' (index: 12) is in the required format.
- Step 2: Map the blocks with the 'miner' as key and 'size' as value.
- Step 3: Add all the sizes for each miner and sort in descending order of size to find top 10 miners.

Results

	MINER	**TOTAL VALUE**
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

D.1: Data Overhead

There are five columns identified in "blocks" schema that may not be required for the functioning of a cryptocurrency, namely sha3_uncles (index: 4), logs_bloom (index: 5), transactions_root (index: 6), state_root (index: 7) and receipts_root (index: 8). To estimate the overhead caused by these columns, we find the number of bits of each field and sum everything.

Methodology

- Step 1: Read all data from "blocks.csv" and filter the block lines with 19 fields as expected.
  Step 2: Calculate the number of bits for each of the five field by taking the length of the hexadecimal value, subtracting 2 for the "0x" characters in the beginning and multiplying by 4, the number of bits used to represent one character in hexadecimal. Create a tuple of these five values and map it to the key 1.
  *Formula: (len(field_value) - 2) * 4*
- Step 3: Sum the bits for each field.
- Step 4: Total all the bits.

Results

	COLUMN NAME	**BITS**
1	sha3_uncles	1792000256
2	logs_bloom	14336002048
3	transactions_root	1792000256

4	state_root	1792000256
5	receipts_root	1792000256
	TOTAL BITS	**21504003072**

The data overhead of the above columns is approximately 2.688 GB (1 bit = 1.25e-10 GB).

D.2: Gas Guzzlers

Gas is the fee required to successfully conduct a transaction or execute a contract on the Ethereum platform.

D.2.1: Gas price and usage over time

To analyse if contract transactions have started consuming gas greedily over time, we need to observe the trend of average gas price and average gas used monthly.

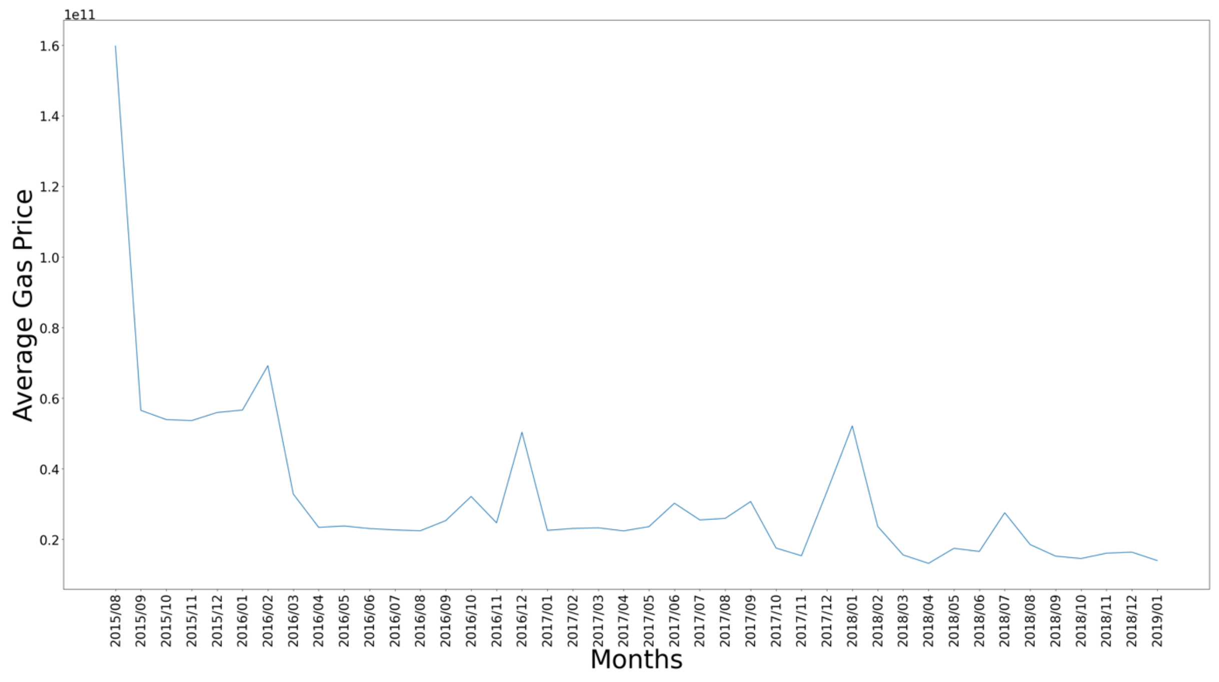
Methodology

- Step 1: Load transaction lines from "transactions.csv" and ensure that each line has 15 fields, and the fields 'gas' (index: 8), 'gas_price' (index: 9) and 'block_timestamp' (index: 11) are in the required format.
- Step 2: Create a map with 'block_timestamp', in "YYYY/MM" format, as key and a tuple of 'gas', ‘gas_price’ and count 1 for each transaction.
- Step 3: Reduce for each month by summing the gas prices, gas used and number of transactions.
- Step 4: Average the gas price and gas used for each month by dividing the total value by number of transactions.

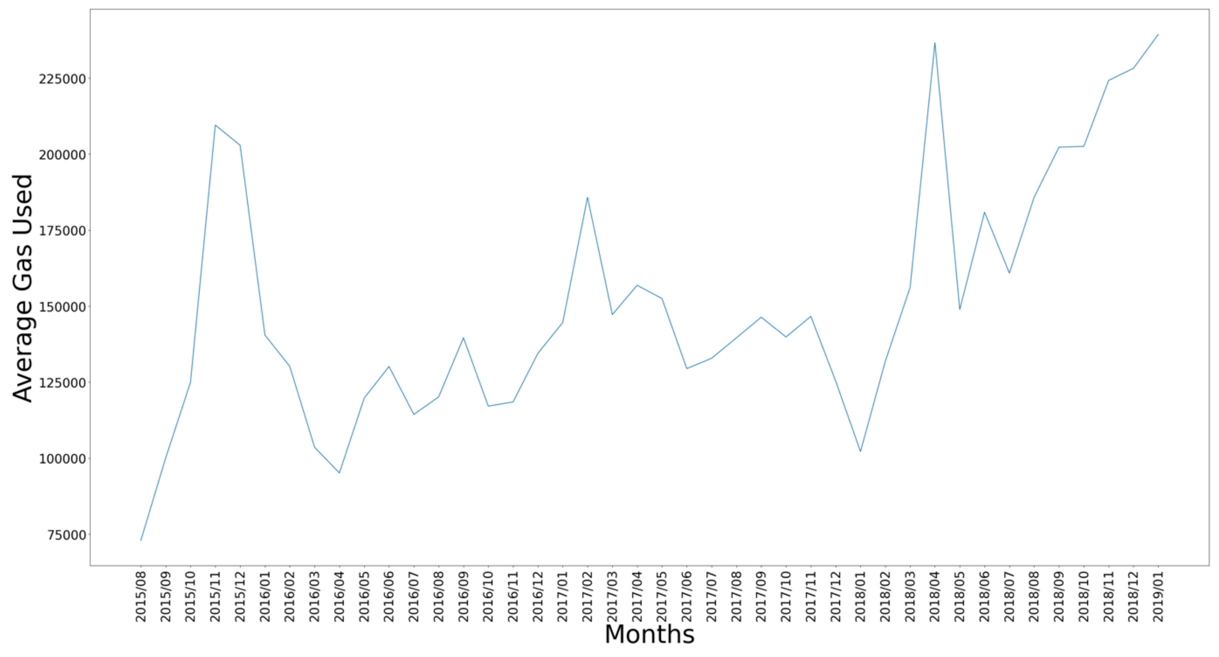
Results

The gas price per transaction has reduced over time. On the other hand, average gas used by contract transactions has an inconsistent trend for most part. When observing the two graphs together, notice that when gas price rises sharply and falls, that is create a local peak, in the subsequent months gas used surges and drops as well. Example, after the fluctuation between 2016/11 and 2017/01 in gas price graph, there was a ripple effect causing fluctuation on gas used between 2017/01 and 2017/03. Similar pattern can be observed from 2017/11 to 2018/03 in gas price and from 2018/03 to 2018/05 in gas used. This could indicate that gas price is one of the factors that causes the fluctuations in gas usage, however there could be other factors contributing to this. Towards the end, we observe that gas usage has consistently increased for 6 months.

Average Gas Price per month



Average Gas Used per month



D.2.2: Comparison of average gas used by top services

To check if the popular services are gas guzzlers, we can compare the average gas used overall with average gas used by each contact transaction.

Methodology

- Step 1: Load all the transactions from “transactions.csv”.

- Step 2: Filter the transactions with a ‘to_address’ (index: 6) and map gas used by each transaction to a common key.
- Step 3: Sum all the gas used across all transactions (excluding contract creation) and average it by dividing the total by number of transactions.
- Step 4: Read all results of Part B and extract the addresses of all top 10 contracts.
- Step 5: Extract all the transactions with only the popular addresses and map each address with the gas used and count for each transaction.
- Step 6: Total all the gas used by each contract and divide it by the number of transactions to each address.

Results

Average gas used = 161725.69962368018

	CONTRACT ADDRESS	**AVG GAS USED**
1	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	83153.10930123563
2	0x7727e5113d1d161373623e5f49fd568b4f543a9e	89826.72419423984
3	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	42553.82170235774
4	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	39999.61330880607
5	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	136704.900966096
6	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	101932.03479546004
7	0x341e790174e3a4d35b65fdc067b6b5634a61caea	198832.66666666666
8	0x58ae42a38d6b33a1e31492b60465fa80da595755	50212.80846597866
9	0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3	33973.648482614575
10	0xe28e72fcf78647adce1f1252f240bbfaebd63bcc	150945.27918781727

D.3: Scams

Schemes utilised to defraud users on the Ethereum Network.

D.3.1: Most lucrative scams

To recognise the most lucrative scams, we need to calculate the total value they have received by deceiving people.

Methodology

- Step 1: Load “scams.json” and use flatMap() to separate scams and further map individual address from ‘addresses’ list with their respective scam ‘id’.
- Step 2: Repeat steps in previous tasks of reading all transactions, cleaning them, mapping ‘to_address’ (index: 6) with value and summing all the values for each address.
- Step 3: Join the scams RDD and transactions RDD by address.
- Step 4: Remap all the records with scam ID as key and add all the values for each scam ID. Further, arrange in descending order to identify the most lucrative scams.

Results

The most lucrative scam is –

‘id’: 5622

‘category’: Scamming

‘value’: 16709083588073530571339

D.2.2: Types of scams and their activity over time

The ‘category’ field in scams tells us the form of scams. We can use this identify different forms of scams and their pattern over time.

Methodology

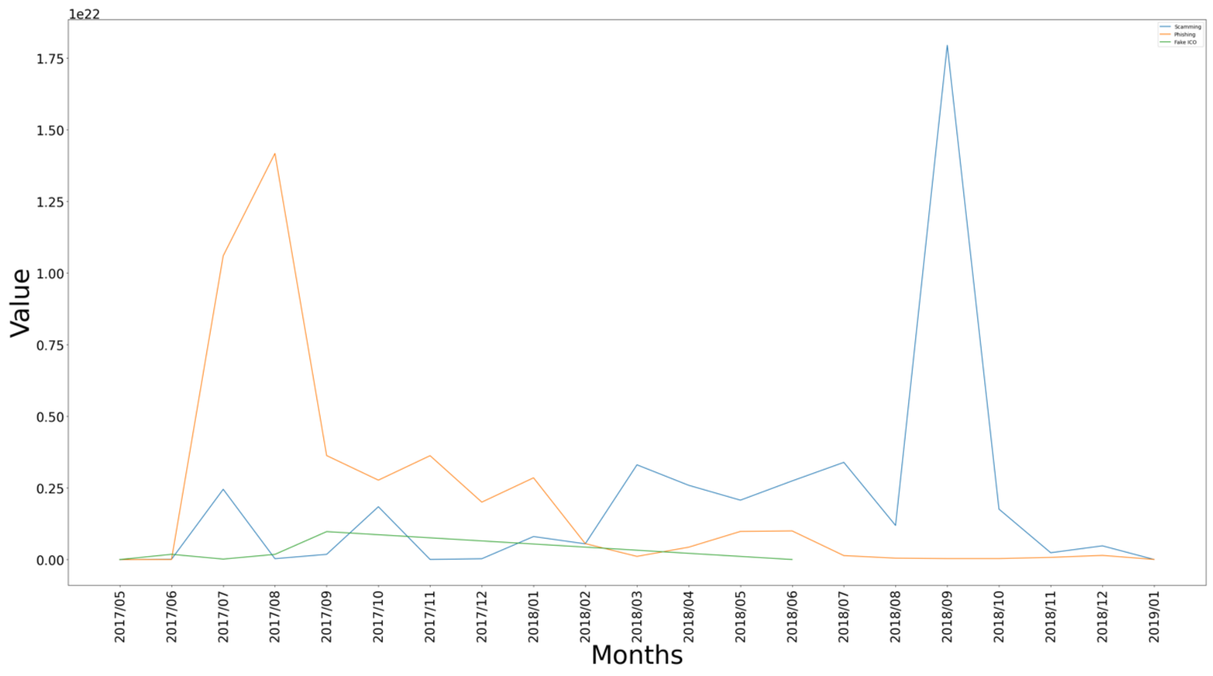
- Step 1: Load all transactions and repeat steps from previous task of mapping ‘to_address’ (index: 6) with ‘value’ and ‘block_timestamp’ in “YYYY/MM” format.
- Step 2: Read all scams from “scams.json” and use flatMap() to separate each scam.
- Step 3: For the category “Scamming”, extract all related addresses and join with the transactions RDD.
- Step 4: Further, create a map with each month and the value.
- Step 5: Repeat steps 3 and 4 for “Phishing” and “Fake ICO” categories as well.

Results

Total Scamming 4.162698862311359e+22

Total Phishing 4.321856144727655e+22

Total Fake ICO 1.35645756688963e+21



Note: Scamming (Blue), Phishing (Orange) and Fake ICO (Green)

We observe that different forms of scams peak at different times and that Fake ICO is the least active, while Phishing has fetched the largest value.

Conclusion

This project has given insight into activities on a blockchain network, inparticular Ethereum. Through these tasks I have improved my understanding and interpretation of requirements and better extracted information from large volumes of data efficiently.