2021

Big Data Architecture & Governance

Northeastern University

Assignment Name:

Group Assignment 1 – Team 4

Student Names:

Apoorva Mishra

Atulya Sharma

Dhruv Panchal

Contents

Assignment 2
1.1. Case 2
1.2. Assignment Goals 2
1.2.1. Visualization Deliverables 2
1.2.2. Other deliverables 3
Documentation 4
1. Vision Diagram 4
2. Velero Screenshots 5
3. Data Profiling, Wrangling and Cleansing 18
4. Database Installation 31
5. Data Mapping and Integration 32
6. Data Validation and Data Visualization 35
7. System Integration and User Acceptance Testing 43
8. Challenges Encountered 45
9. End User Instructions 46

Assignment

1.1. Case

Each team should select a dataset to analyze and build an analytical dashboard as a Proof-of-concept to illustrate the value of data driven analytics. You need to present your dataset.

.

1.2. Assignment Goals

To work with datasets, Perform/Create:

- Group Assignment/Project on Velero with below mentioned activities:
  - Tasks/ Project Short/ Long Form/ Group Allocation/ Timesheet/ Issues & Risks.
- Data Profiling – Using Python profiling library, describe your understanding of the data.
- Data Wrangling and Cleansing - Pandas/Alteryx/XSV
  - Filtering and Aggregating if needed.
  - Missing value handling.
  - Deriving additional columns from existing datasets if needed.
  - Cleaning (removing blank spaces, formatting dates, Capitalizing etc.) .
- Database Installation: Install NEO4J database .
- Data Mapping and Integration to your Database for the Entire Dataset.
- Business and Technical Metadata – develop business term list describing all the data elements available in the file.
- Data Validation and Data Visualization using Python – Validate the data using python and provide a dashboard using python visualization libraries.
- System Integration and User Acceptance Testing - Test Cases – describe your validation & testing process.
- Risks/Issues of project.
- Describe challenges encountered and how you resolved them.
- End User Instructions (Steps to run your Dashboard) – provide a full description how to run your process:B
  - Database Creation and load.
  - Visualization interpretation - describe information regarding your findings.

1.2.1. VISUALIZATION DELIVERABLES

Once you wrangle/clean/join/integrate the data, import the data into **NEO4J** and illustrate how to use the appropriate graph to illustrate various aspects of analysis.

Questions to consider:

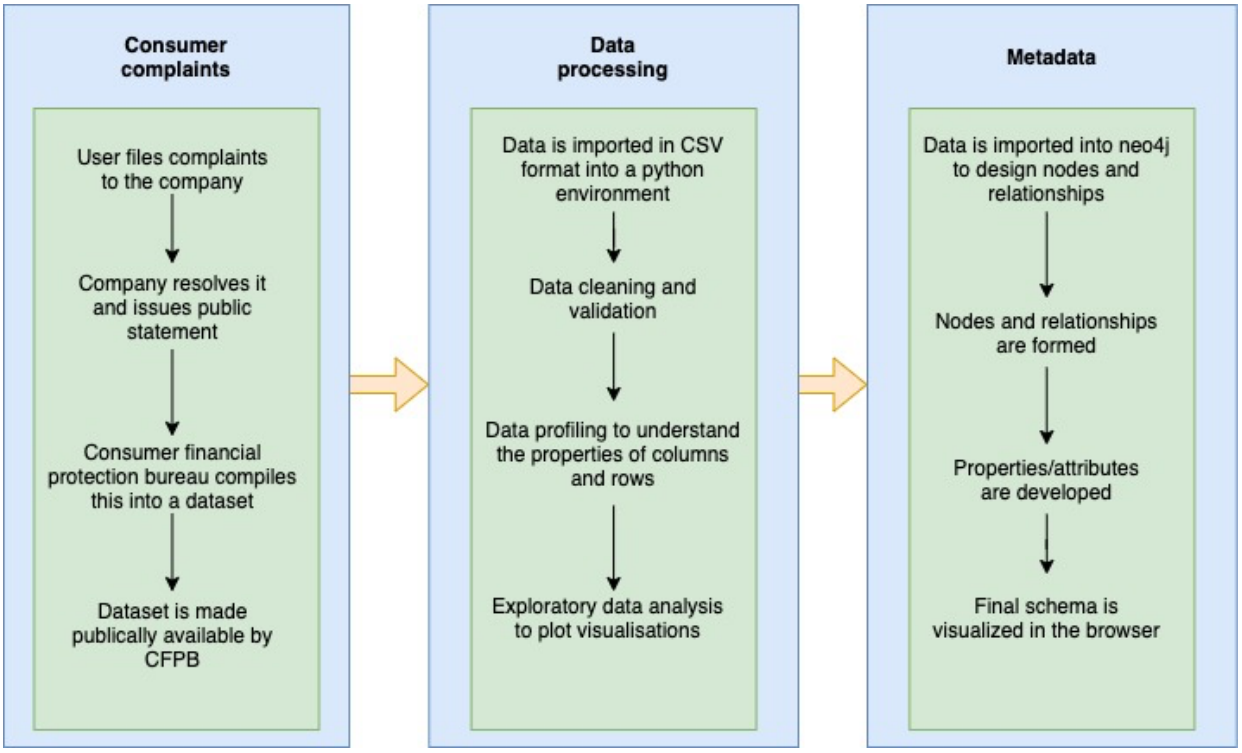
- Columns used for dimensions, and columns that are used for measurement.
- How would you generate new dimensions?
- Who would use this dashboard and how they benefit from your dashboard?
- What value would be generated using this dashboard?

1.2.2. OTHER DELIVERABLES

- Presentation of the entire work from the first step till the dashboards including the Velero screenshots.
- Business and technical metadata presentation – Identifying all available business terms and extracting related technical metadata.
- Complete explanation of the dashboard and usability.
- Complete instruction as how to implement and run the database load, technical meta data extraction, and dashboard.

Documentation

1. Vision Diagram



2. Velero Screenshots

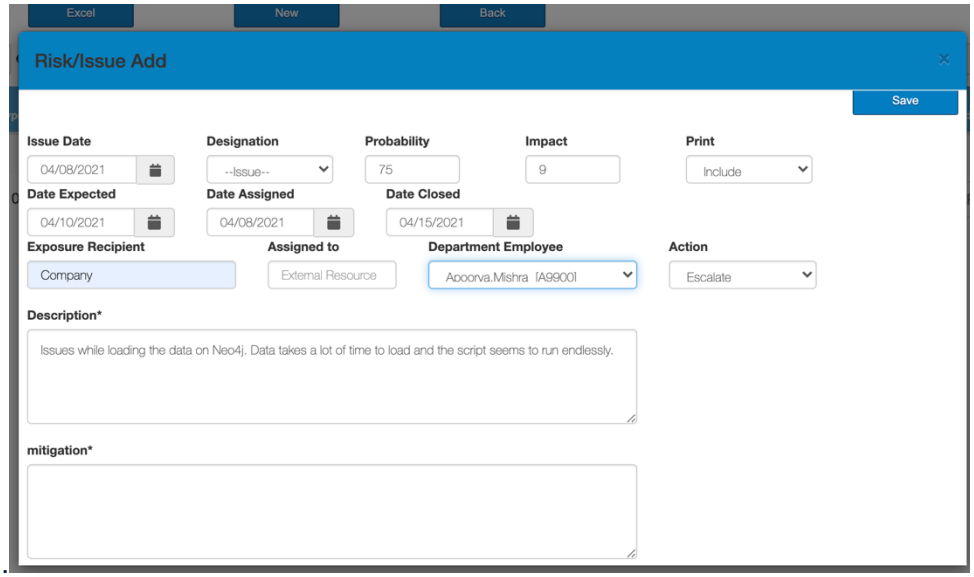
- Velero's Enterprise Transparancy Platform consists of enterprise portfolio planning and execution.
- Velero ETP enable executives quickly view the calculated estimated costs and full resource allocation information. Velero ETP includes fully integrated time, resource and budget management modules.
- Velero's integrated solution provides a fast and easy access to all aspect of portfolio planning and execution such as defining strategic objectives, estimated resources, budget, risk/issue, mandate, status and activities, with the ability to attach and upload related documents directly within the product

Below are several screenshots for different use cases in Velero.

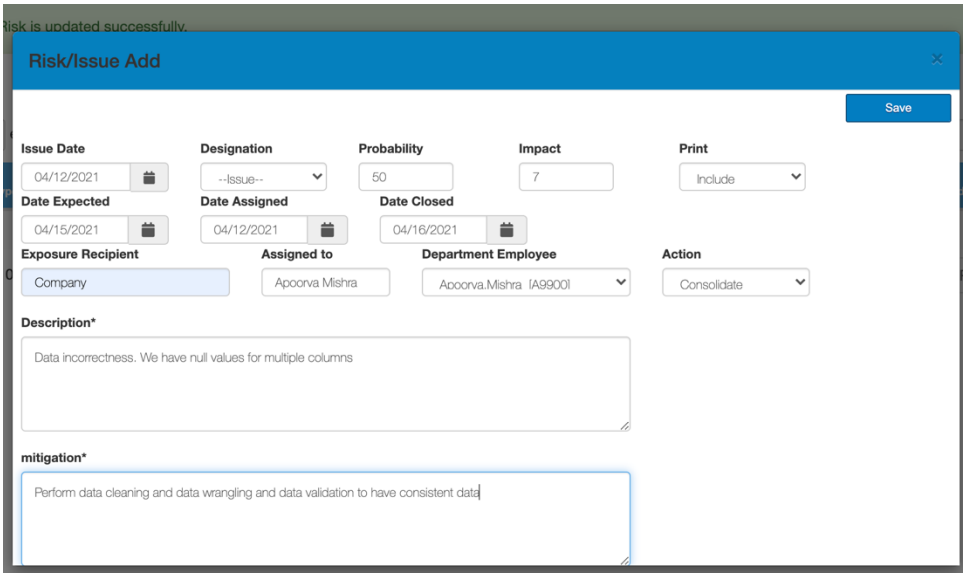
Issues/Risks

We have identified two risks that we have accounted in Velero.

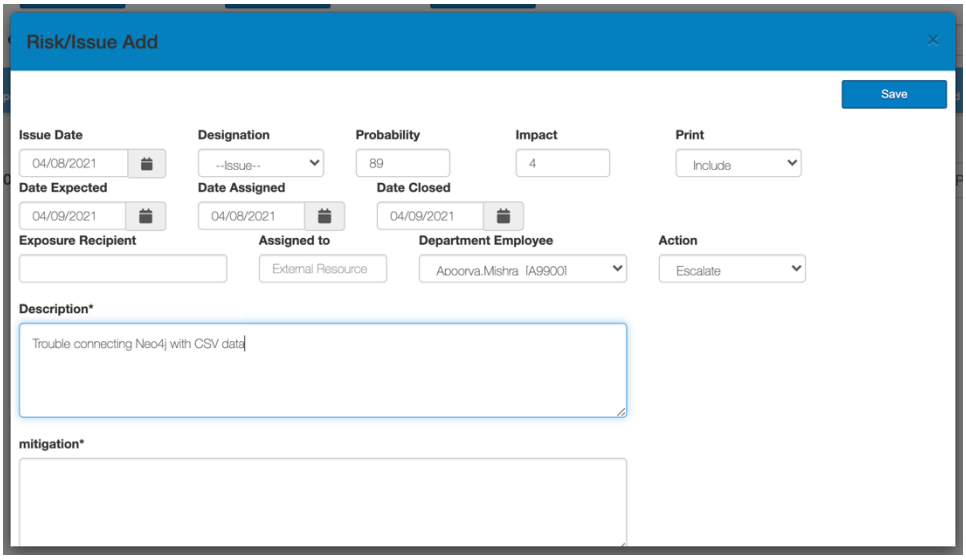
1. Issue while loading data in Neo4j



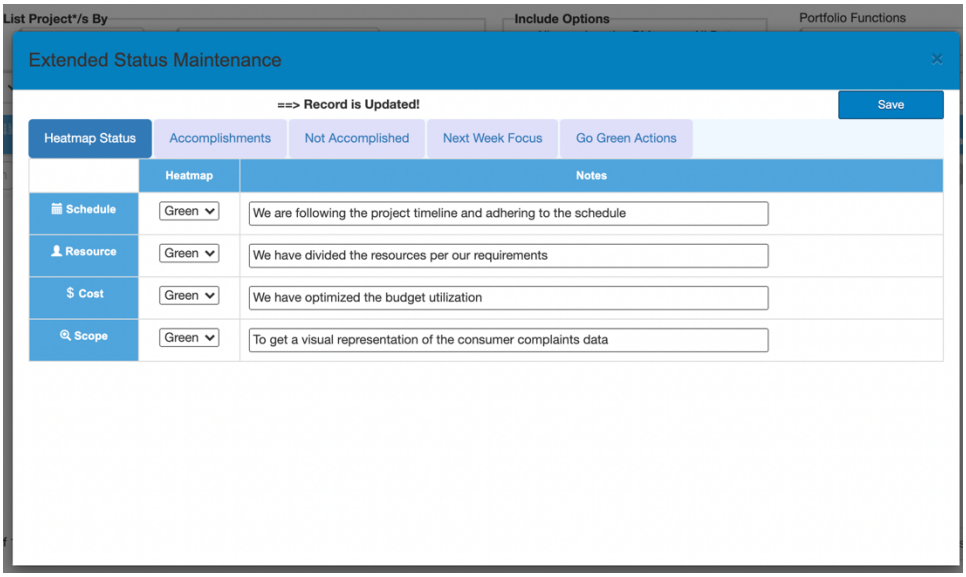
2. Data Incorrectness



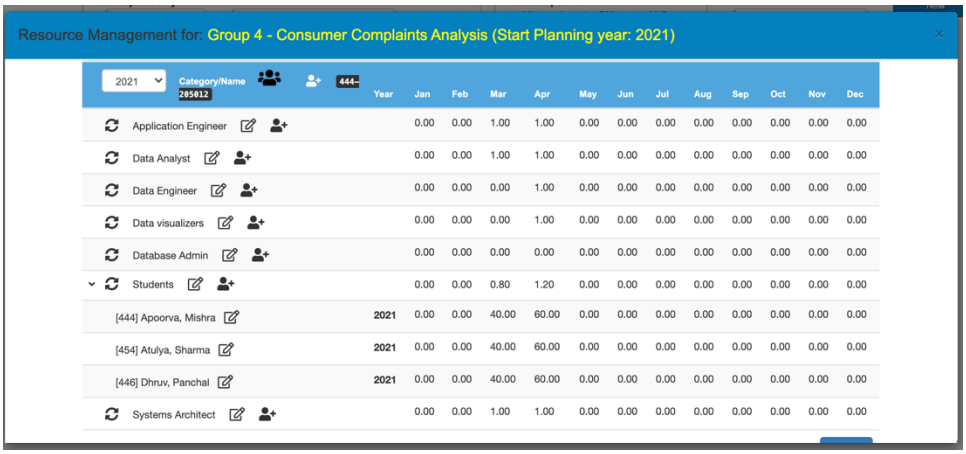
3. Trouble connecting data with Neo4j



Heat Map Status



Resource Planning

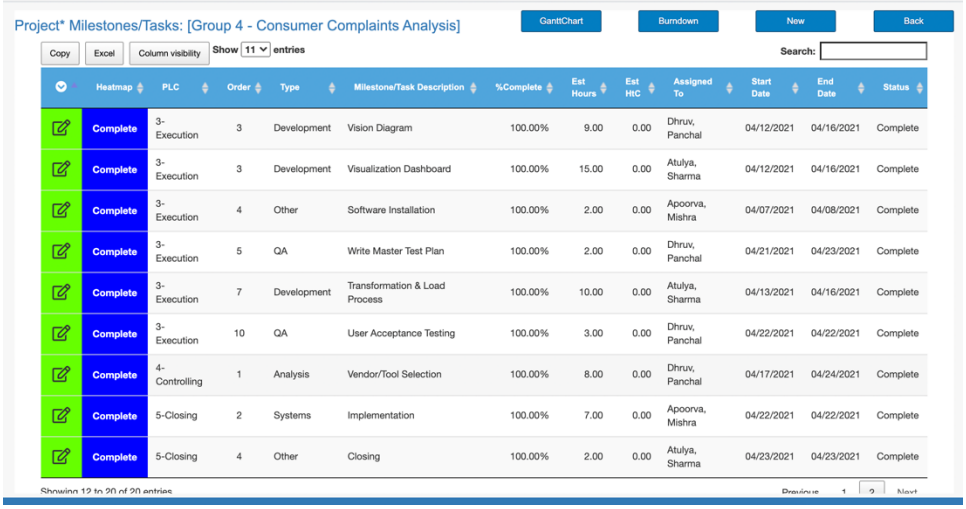


Milestone/Tasks with Allotted Times per Activity

Milestone/TaskList1

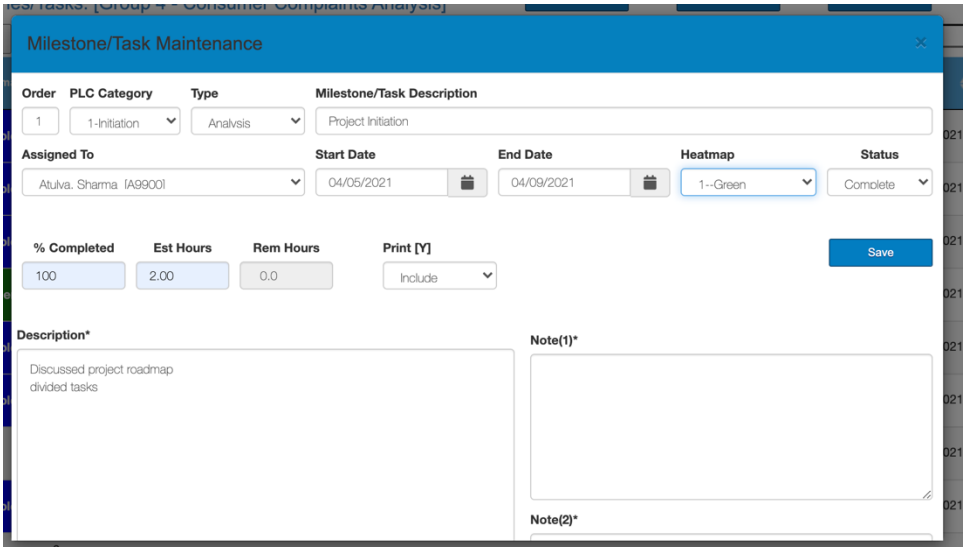
Heatmap	PLC	Order	Type	Milestone/Task Description	%Complete	Est Hours	Est HtC	Assigned To	Start Date	End Date	Status
Complete	1-Initiation	1	Analysis	Project Initiation	100.00%	2.00	0.00	Apoorva, Mishra	04/05/2021	04/09/2021	Complete
Complete	1-Initiation	1	Analysis	Project Initiation	100.00%	2.00	0.00	Atulya, Sharma	04/05/2021	04/09/2021	Complete
Complete	1-Initiation	2	Analysis	Architecture Design	100.00%	10.00	0.00	Atulya, Sharma	04/15/2021	04/24/2021	Complete
Complete	1-Initiation	3	Analysis	Non-functional Requirements	100.00%	15.00	0.00	Dhruv, Panchal	04/21/2021	04/24/2021	Complete
Green	2-Planning	1	Analysis	Talent Management & Optimization	90.00%	4.00	0.40	Dhruv, Panchal	04/17/2021	04/24/2021	Inprocess
Complete	2-Planning	1	Analysis	Project Planning	100.00%	5.00	0.00	Atulya, Sharma	04/08/2021	04/13/2021	Complete
Complete	3-Execution	1	Analysis	Architecture review & Approval	100.00%	16.00	0.00	Apoorva, Mishra	04/15/2021	04/22/2021	Complete
Complete	3-Execution	2	Development	Data Cleaning	100.00%	14.00	0.00	Atulya, Sharma	04/12/2021	04/16/2021	Complete
Complete	3-Execution	2	Development	Functional Requirements	100.00%	8.00	0.00	Atulya, Sharma	04/17/2021	04/24/2021	Complete
Complete	3-Execution	3	Development	Data Extraction	100.00%	5.00	0.00	Atulya, Sharma	04/19/2021	04/21/2021	Complete
Complete	3-Execution	3	Development	Neo4j Script Development	100.00%	19.00	0.00	Apoorva, Mishra	04/12/2021	04/16/2021	Complete

Milestone/TaskList2

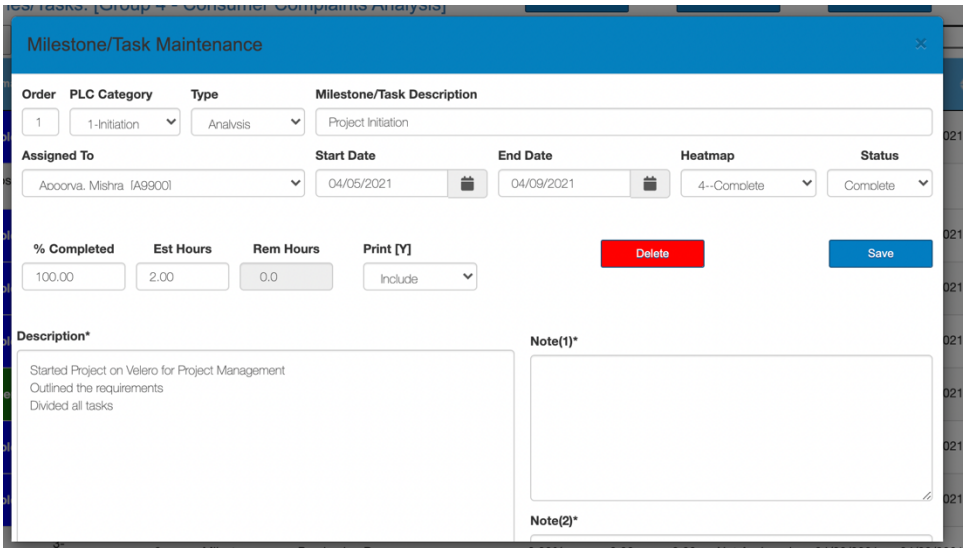


Heatmap	PLC	Order	Type	Milestone/Task Description	%Complete	Est Hours	Est HtC	Assigned To	Start Date	End Date	Status
Complete	3-Execution	3	Development	Vision Diagram	100.00%	9.00	0.00	Dhruv, Panchal	04/12/2021	04/16/2021	Complete
Complete	3-Execution	3	Development	Visualization Dashboard	100.00%	15.00	0.00	Atulya, Sharma	04/12/2021	04/16/2021	Complete
Complete	3-Execution	4	Other	Software Installation	100.00%	2.00	0.00	Apoorva, Mishra	04/07/2021	04/08/2021	Complete
Complete	3-Execution	5	QA	Write Master Test Plan	100.00%	2.00	0.00	Dhruv, Panchal	04/21/2021	04/23/2021	Complete
Complete	3-Execution	7	Development	Transformation & Load Process	100.00%	10.00	0.00	Atulya, Sharma	04/13/2021	04/16/2021	Complete
Complete	3-Execution	10	QA	User Acceptance Testing	100.00%	3.00	0.00	Dhruv, Panchal	04/22/2021	04/22/2021	Complete
Complete	4-Controlling	1	Analysis	Vendor/Tool Selection	100.00%	8.00	0.00	Dhruv, Panchal	04/17/2021	04/24/2021	Complete
Complete	5-Closing	2	Systems	Implementation	100.00%	7.00	0.00	Apoorva, Mishra	04/22/2021	04/22/2021	Complete
Complete	5-Closing	4	Other	Closing	100.00%	2.00	0.00	Atulya, Sharma	04/23/2021	04/23/2021	Complete

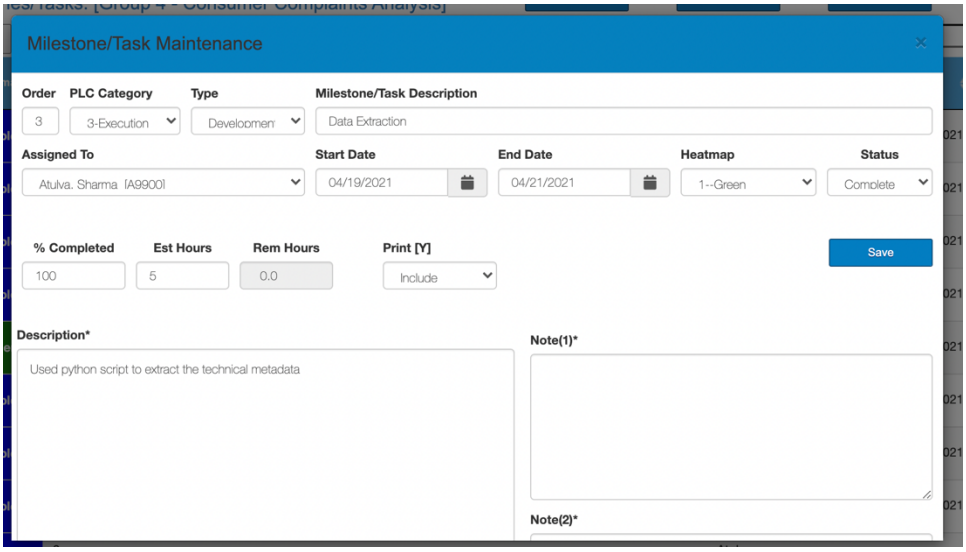
Project Initiation



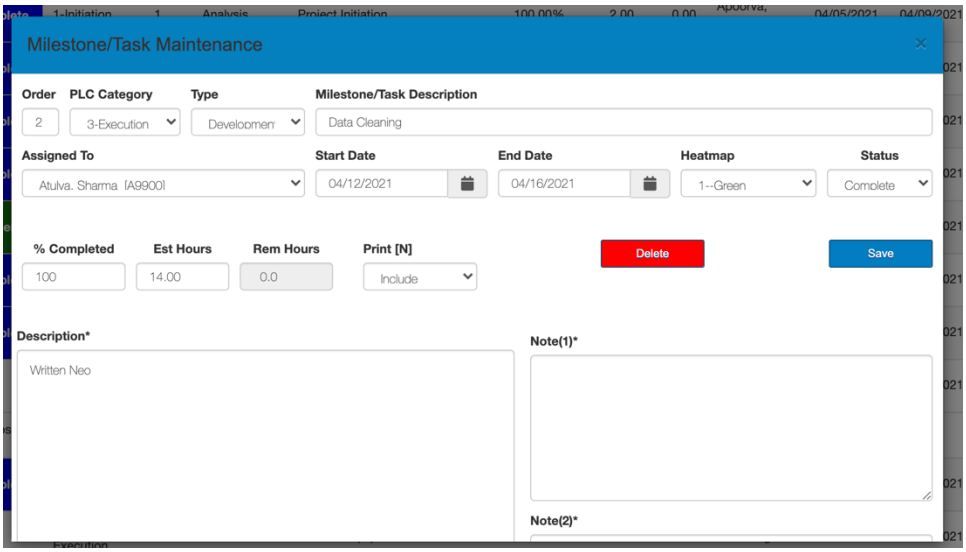
Project Initiation



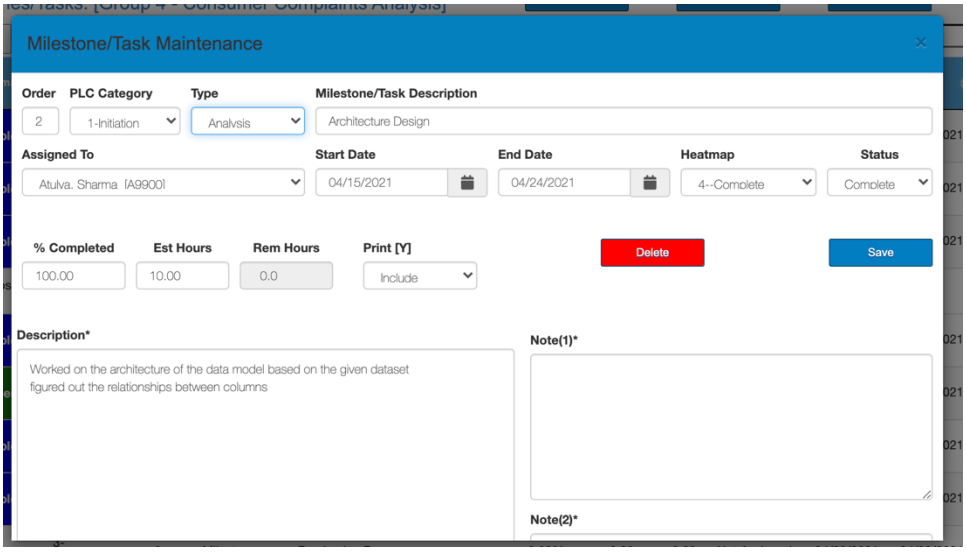
Data Extraction



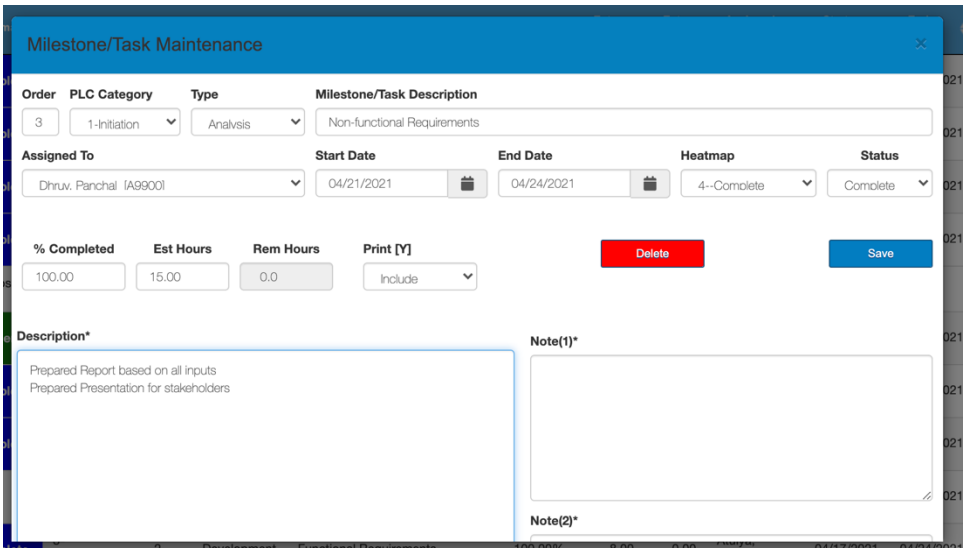
Data Cleaning



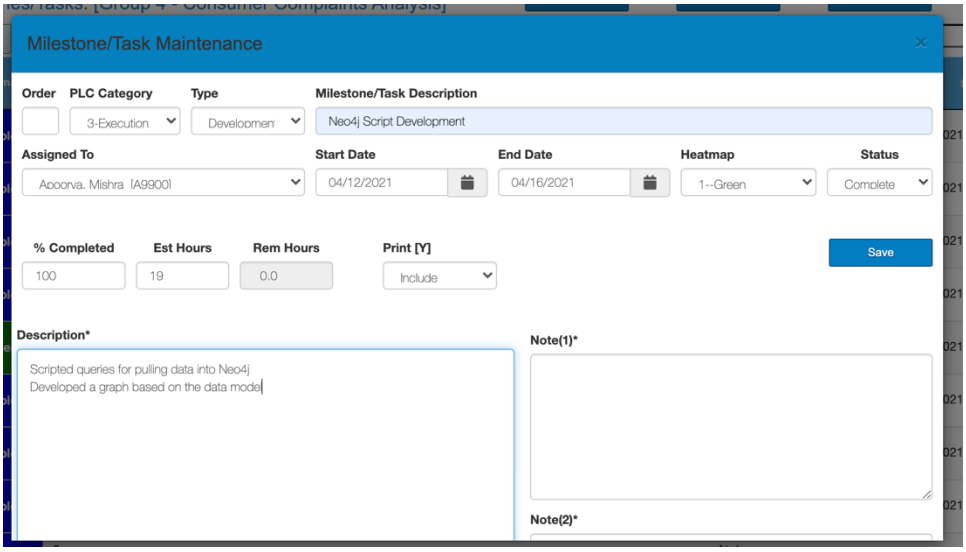
Architecture Design



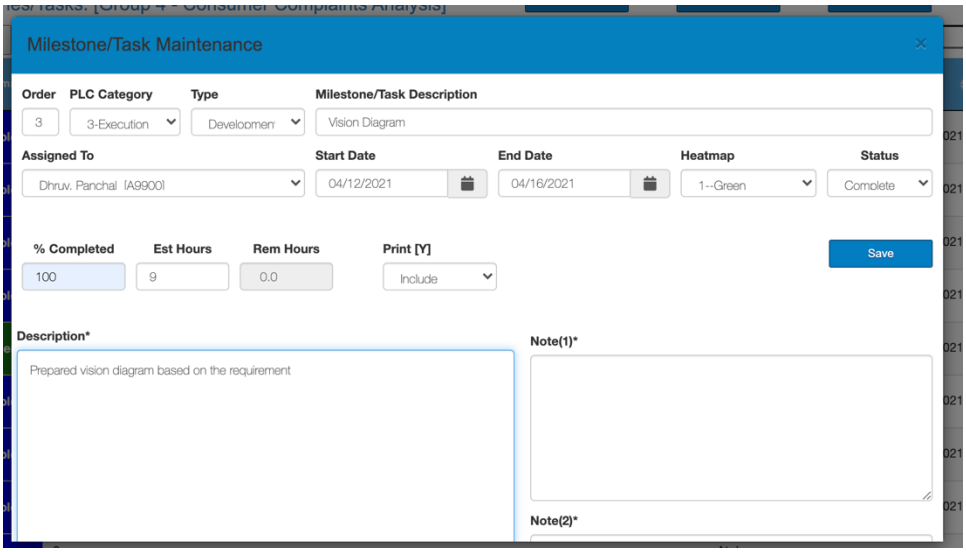
Non-Functional Requirement



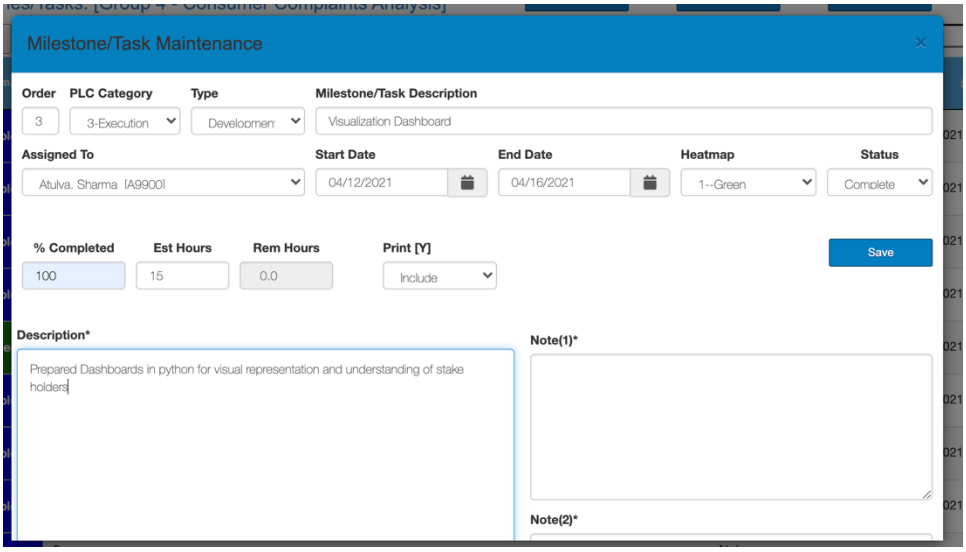
Neo4j Script Development



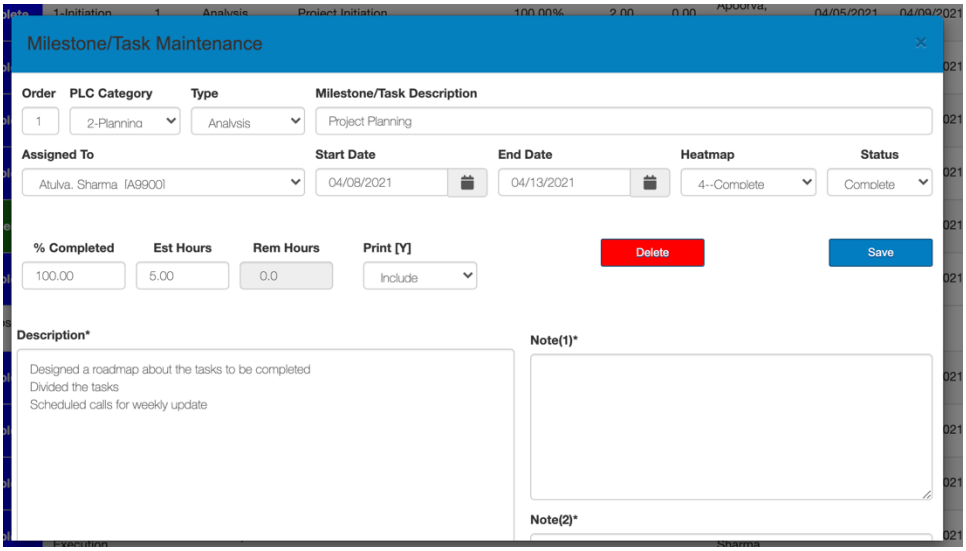
Vision Diagram



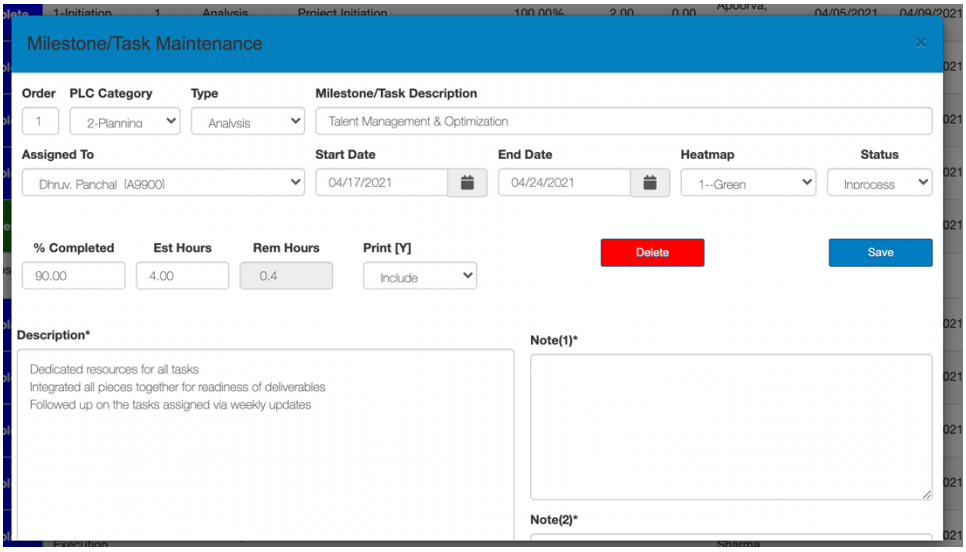
Visualization Dashboards



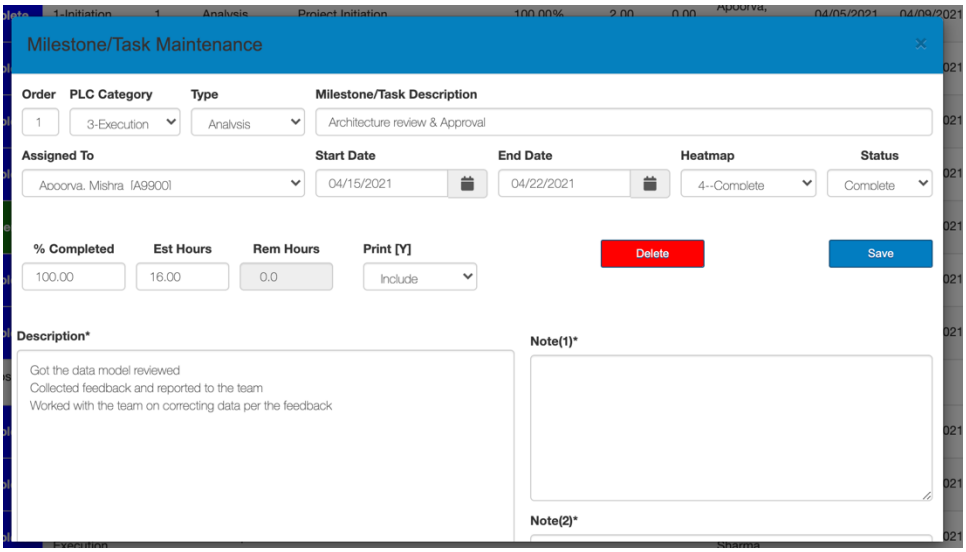
Project Planning



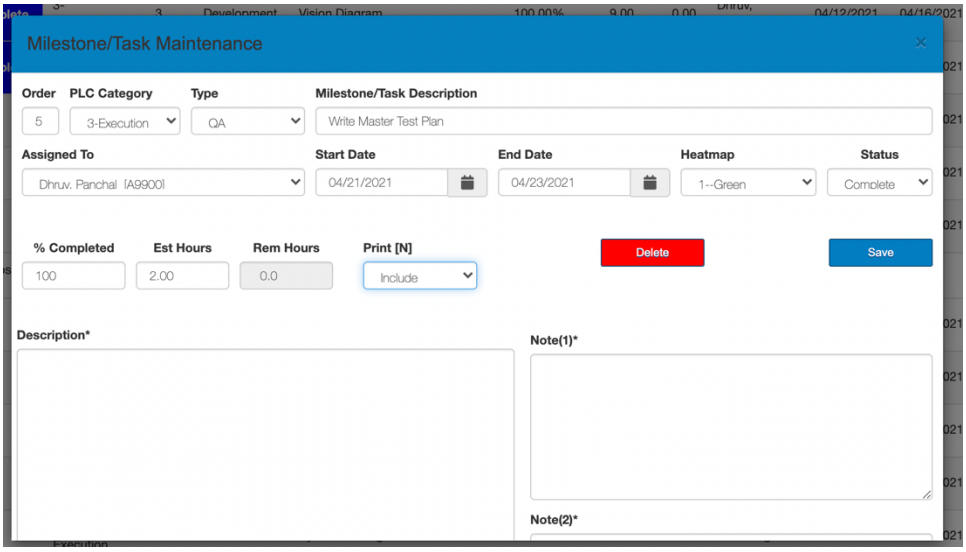
Talent Management & Optimization



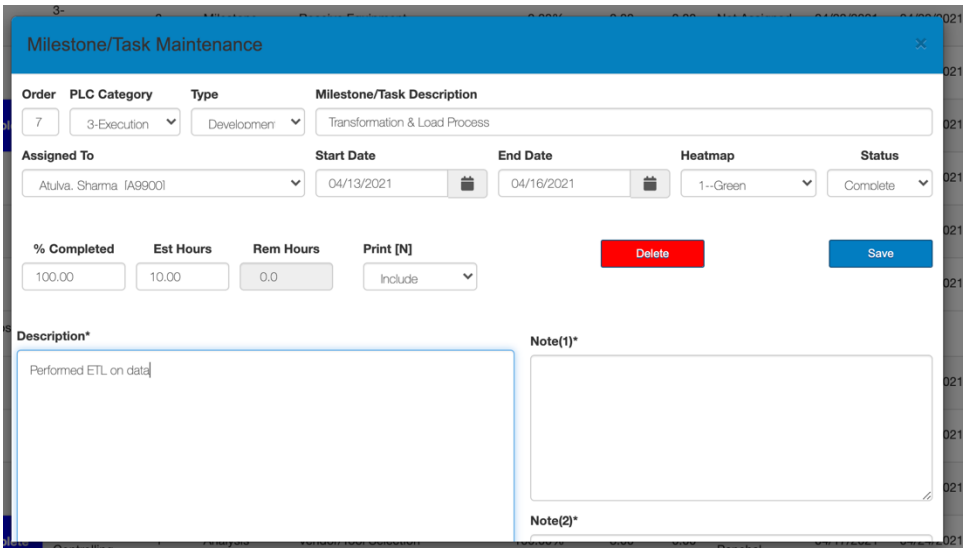
Architecture Review & Approval



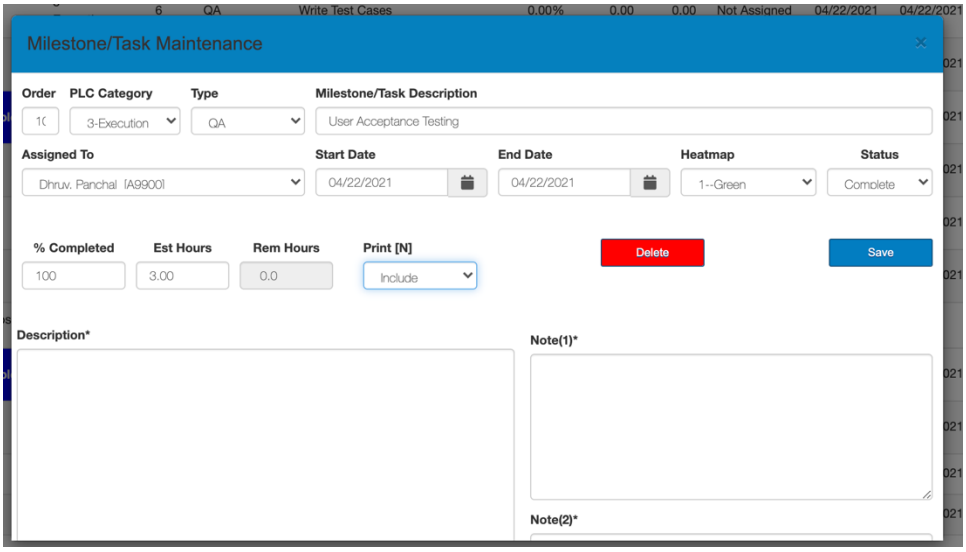
Write Master Test Plan



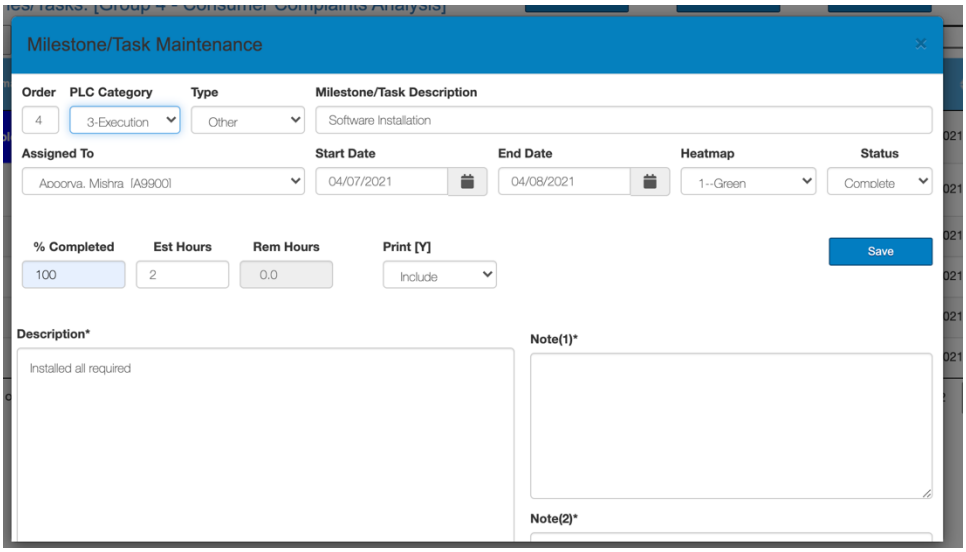
Transformation and Load Processing



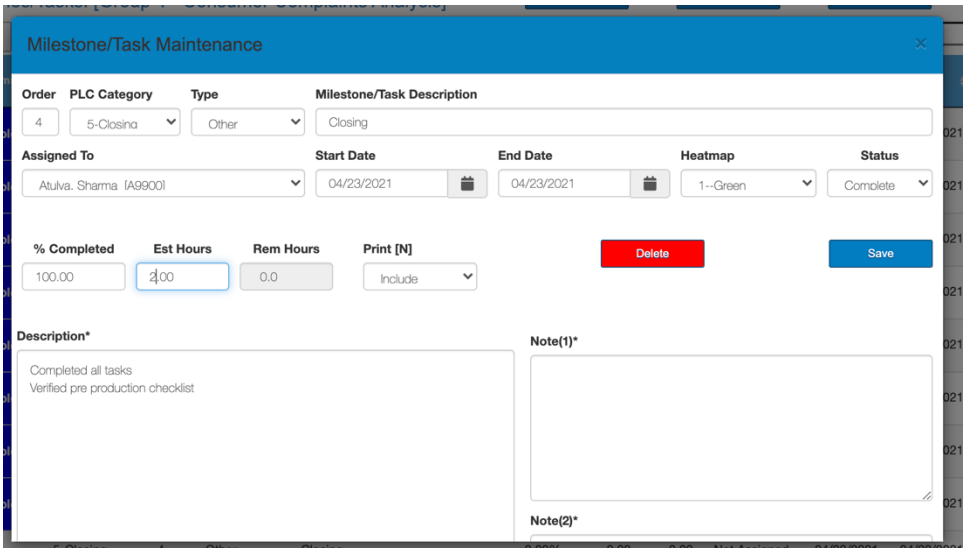
User Acceptance Testing



Software Installation



Closing


Milestone/Task Maintenance
Order
PLC Category
Type
Milestone/Task Description
4
5-Closing
Other
Closing
Assigned To
Start Date
End Date
Heatmap
Status
Atulya. Sharma [A9900]
04/23/2021
04/23/2021
1--Green
Complete
% Completed
Est Hours
Rem Hours
Print [N]
Delete
Save
100.00
2.00
0.0
Include
Description*
Completed all tasks
Verified pre production checklist
Note(1)*
Note(2)*

3. Data Profiling, Wrangling and Cleansing

The dataset we are using is called “Consumer Complaints”. This data is a collection of complaints about consumer financial products and services that was sent to the companies for response. Complaints are published after the company responds, confirming a commercial relationship with the consumer, or after 15 days, whichever comes first.

The dataset comprises of Consumer Complaints on financial products, and we’ll see how to classify consumer complaints text into these categories: Debt collection, Consumer Loan, Mortgage, Credit card, Credit reporting, Student loan, Bank account or service, Payday loan, Money transfers, Other financial service, Prepaid card.

Data Profiling has been done using the pandas profiling library on python. It gives us information that is an extension of the info provided after using **df.describe()** function of pandas. A detailed report can be generated in the form of an interactive HTML page or can be integrated directly into the python environment.

Initial Installation of Pandas Profiling Library

Pandas Profiling can be installed by running the following on the Command Line Interface:

pip install pandas-profiling

Getting Started

To use pandas-profiling, we need to import it to our notebook using the following code:

Import numpy as np
Import pandas as pd
From pandas_profiling import ProfileReport

This is followed by loading the dataset on the python notebook.

```
df1 = pd.read_csv("Consumer_Complaints.csv")
```

To view the dataset, type the following code:

```
df1.head()
```

Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	Company	State	ZIP code	Tags	Consumer consent provided?	Submitted via	Date sent to company
07/29/2013	Consumer Loan	Vehicle loan	Managing the loan or lease	NaN	NaN	NaN	Wells Fargo & Company	VA	24540	NaN	NaN	Phone	07/30/2013
07/29/2013	Bank account or service	Checking account	Using a debit or ATM card	NaN	NaN	NaN	Wells Fargo & Company	CA	95992	Older American	NaN	Web	07/31/2013
07/29/2013	Bank account or service	Checking account	Account opening, closing, or management	NaN	NaN	NaN	Santander Bank US	NY	10065	NaN	NaN	Fax	07/31/2013
07/29/2013	Bank account or service	Checking account	Deposits and withdrawals	NaN	NaN	NaN	Wells Fargo & Company	GA	30084	NaN	NaN	Web	07/30/2013
07/29/2013	Mortgage	Conventional fixed mortgage	Loan servicing, payments, escrow account	NaN	NaN	NaN	Franklin Credit Management	CT	06106	NaN	NaN	Web	07/30/2013

By using the function df1.describe(), we get the following outcome:

	Complaint ID
count	6.705980e+05
mean	1.145473e+06
std	6.468630e+05
min	1.000000e+00
25%	5.930575e+05
50%	1.174558e+06
75%	1.718337e+06
max	2.218987e+06

This obviously does not give a lot of information about the dataset that can be used to do analysis. Thus, profiling is needed.

Profiling Report Before Data Cleansing/Wrangling

We have used the following code to get the profiling report:

```
profile = ProfileReport(df1, title = 'Consumer Complaints Report')
```

We then have the option to either view the report within Jupyter Notebook or in the form of a separate document by using:

```
profile_eda.to_notebook_iframe()
```

OR

```
profile_eda.to_file("Post_EDA_Consumer_Complaints_Report.html")
```

Understanding of the Profiling Report

The data has 18 columns with 670,598 **rows**. **23%** of the cells have missing data. Out of these, 15 columns are Categorical (String), 2 are Boolean (true/false) and 1 is Numeric.

Dataset statistics

Number of variables	18
Number of observations	670598
Missing cells	2770590
Missing cells (%)	23.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	92.1 MiB
Average record size in memory	144.0 B

Variable types

Categorical	15
Boolean	2
Numeric	1

The Dataset has multiple columns with high cardinality. This basically means that that these columns have many distinct values, but they are not necessarily unique throughout the dataset.

Another Observation that can be made is that multiple columns also have null or NaN values.

“Complaint ID” is the only column with unique values where none of the rows have are missing values.

Warnings

`Date received` has a high cardinality: 1818 distinct values	High cardinality
`Issue` has a high cardinality: 95 distinct values	High cardinality
`Sub-issue` has a high cardinality: 68 distinct values	High cardinality
`Consumer complaint narrative` has a high cardinality: 112690 distinct values	High cardinality
`Company` has a high cardinality: 3933 distinct values	High cardinality
`State` has a high cardinality: 62 distinct values	High cardinality
`ZIP code` has a high cardinality: 27889 distinct values	High cardinality
`Date sent to company` has a high cardinality: 1767 distinct values	High cardinality
`Submitted via` is highly correlated with `Consumer consent provided?`	High correlation
`Consumer consent provided?` is highly correlated with `Submitted via`	High correlation
`Product` is highly correlated with `Sub-issue` and 1 other fields	High correlation
`Sub-issue` is highly correlated with `Product` and 1 other fields	High correlation
`Issue` is highly correlated with `Product` and 1 other fields	High correlation
`Sub-product` has 198202 (29.6%) missing values	Missing
`Sub-issue` has 400730 (59.8%) missing values	Missing
`Consumer complaint narrative` has 555894 (82.9%) missing values	Missing
`Company public response` has 525401 (78.3%) missing values	Missing
`Tags` has 575868 (85.9%) missing values	Missing
`Consumer consent provided?` has 462447 (69.0%) missing values	Missing
`Consumer disputed?` has 41419 (6.2%) missing values	Missing
`Consumer complaint narrative` is uniformly distributed	Uniform
`Complaint ID` has unique values	Unique

Analysis of each column:

- Date received: This column defines the date at which the user complaint was filed. This has no missing data but has high cardinality.

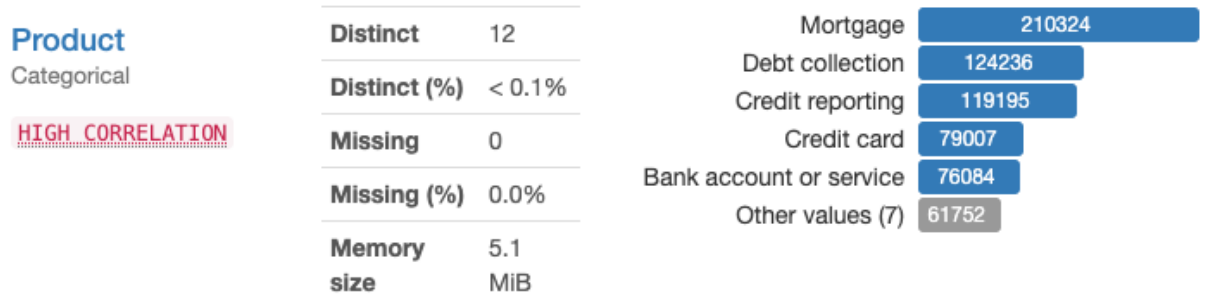
Date received
Categorical

`HIGH CARDINALITY`

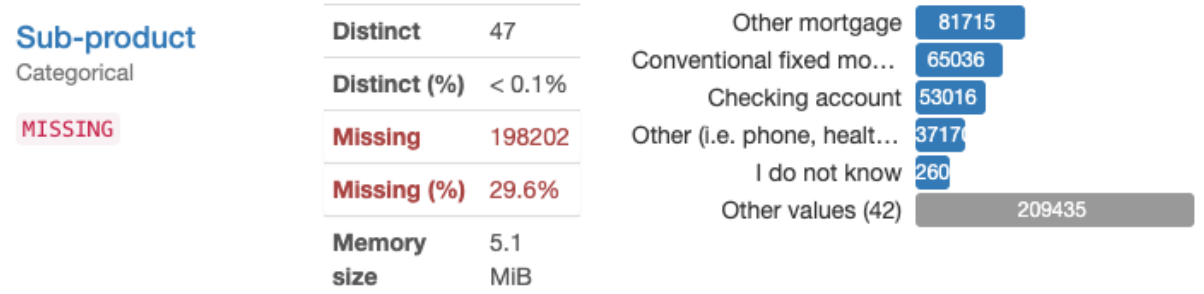
Distinct	1818
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	5.1 MiB

08/27/2015	963
06/26/2014	916
07/06/2016	914
08/26/2015	912
09/20/2016	909
Other values (1813)	665984

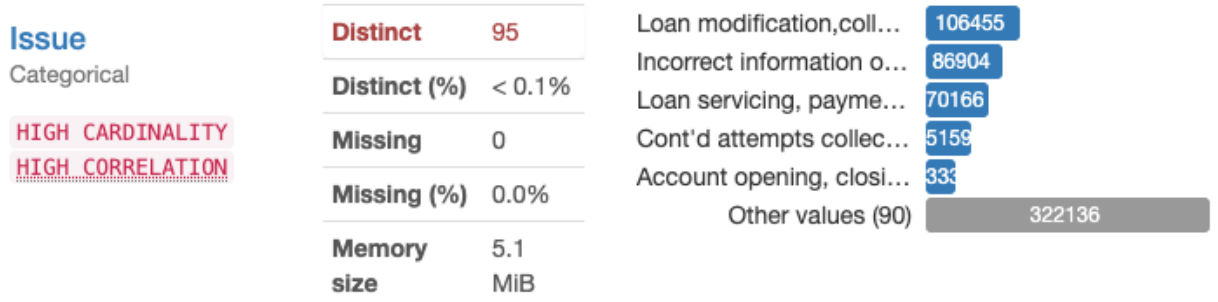
- Product: This column defines the product the complaint was filed. This has no missing data. It contains 12 distinct values and has high correlation with Sub-issue and one other field.



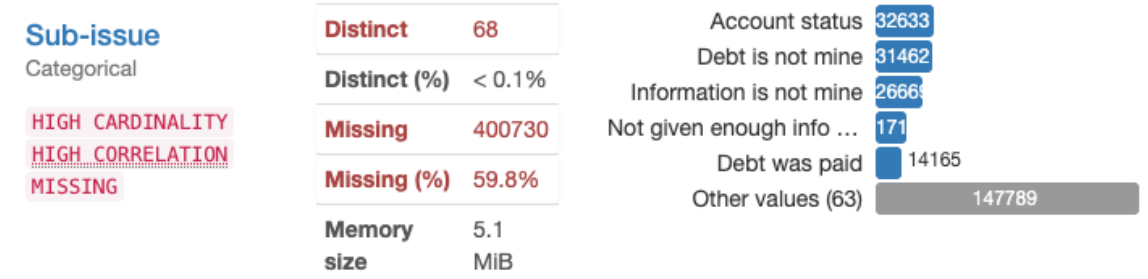
- Sub-product: This column is a sub column for the product category and defines the product on a granular level.



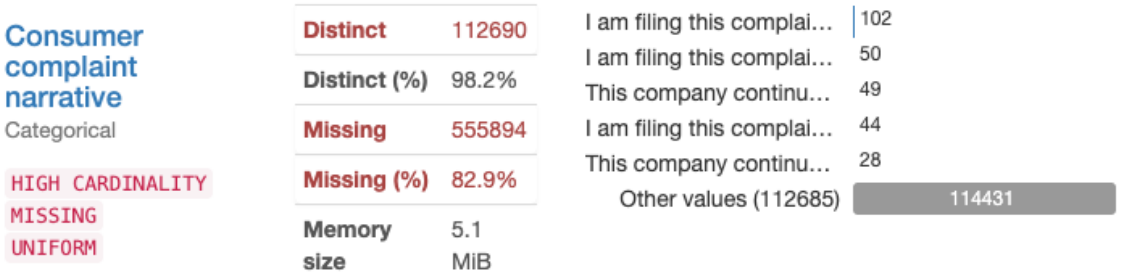
- Issue: This column defines the issue for which the complaint was filed. This has no missing data but has multiple distinct values and it also high correlated to 2 columns.



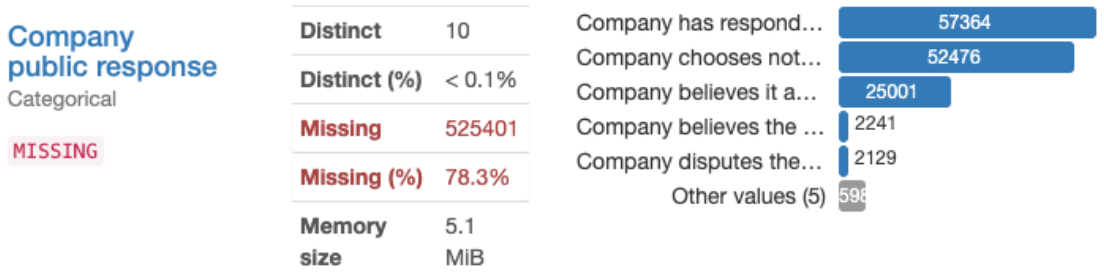
- Sub-issue: This column is a secondary column to the issue column and defines the issue on a granular level.



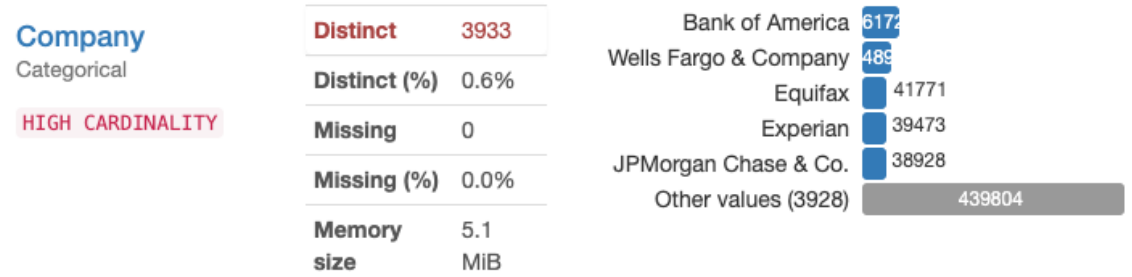
- Consumer complaint narrative: This column is the textual representation of the customers issue and has majority of the rows missing but has a uniform distribution of values.



- Company public response: This column defines the public response the company gave for the complaint filed. It has 78.3% values missing.



- Company: This column gives the name of the company against whom the complaint is filed. This a no missing values but more than 3900 distinct values.

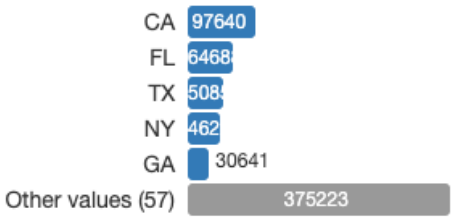


- State: Defines the US state the complaint was filed in. This column has less than 1% of the values missing.

State
Categorical

HIGH CARDINALITY

Distinct	62
Distinct (%)	< 0.1%
Missing	5305
Missing (%)	0.8%
Memory size	5.1 MiB

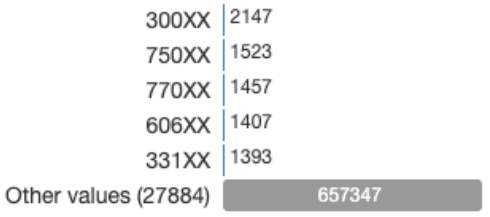


- Zip code: This column defines the zipcode of the complaint. This is a sub column of the state and defines it on a granular level. This should have been a numeric but due to certain columns having incorrect values, this column is not the most consistent.

ZIP code
Categorical

HIGH CARDINALITY

Distinct	27889
Distinct (%)	4.2%
Missing	5324
Missing (%)	0.8%
Memory size	5.1 MiB



- Tags: This column defines the tags given to the complaint if any. Majority of rows have no value.

Tags
Categorical

MISSING

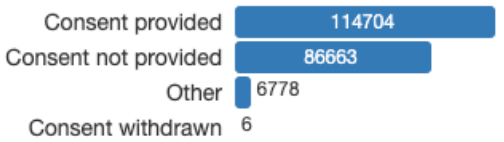
Distinct	3
Distinct (%)	< 0.1%
Missing	575868
Missing (%)	85.9%
Memory size	5.1 MiB

- Consumer consent provided: This column says whether consent was given to access information.

Consumer consent provided?
Categorical

HIGH CORRELATION
MISSING

Distinct	4
Distinct (%)	< 0.1%
Missing	462447
Missing (%)	69.0%
Memory size	5.1 MiB

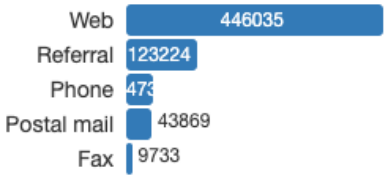


- Submitted via: This column defines the way the complaint was filed.

Submitted via
Categorical

HIGH CORRELATION

Distinct	6
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	5.1 MiB

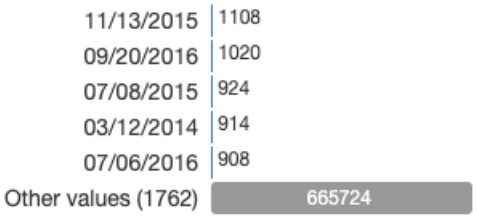


- Date sent to company: This tells the date the complaint was sent to the company.

Date sent to company
Categorical

HIGH CARDINALITY

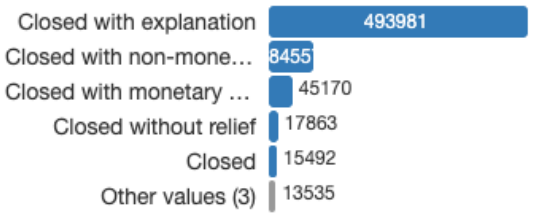
Distinct	1767
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	5.1 MiB



- Company response to consumer: This column defines the response that the company gave to the consumer.

Company response to consumer
Categorical

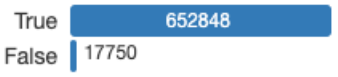
Distinct	8
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	5.1 MiB



- Timely_response: This column shows whether the response was timely.

Timely response?
Boolean

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	655.0 KiB



- Consumer disputed: This column describes the outcome of the resolution or the complaint.

Consumer disputed?
Boolean

MISSING

Distinct	2
Distinct (%)	< 0.1%
Missing	41419
Missing (%)	6.2%
Memory size	1.3 MiB

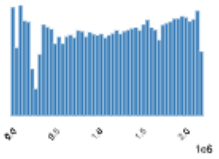
- Complaint ID: This column defines the compliant id if the compliant.

Complaint ID
Real number ($\mathbb{R}_{\geq 0}$)

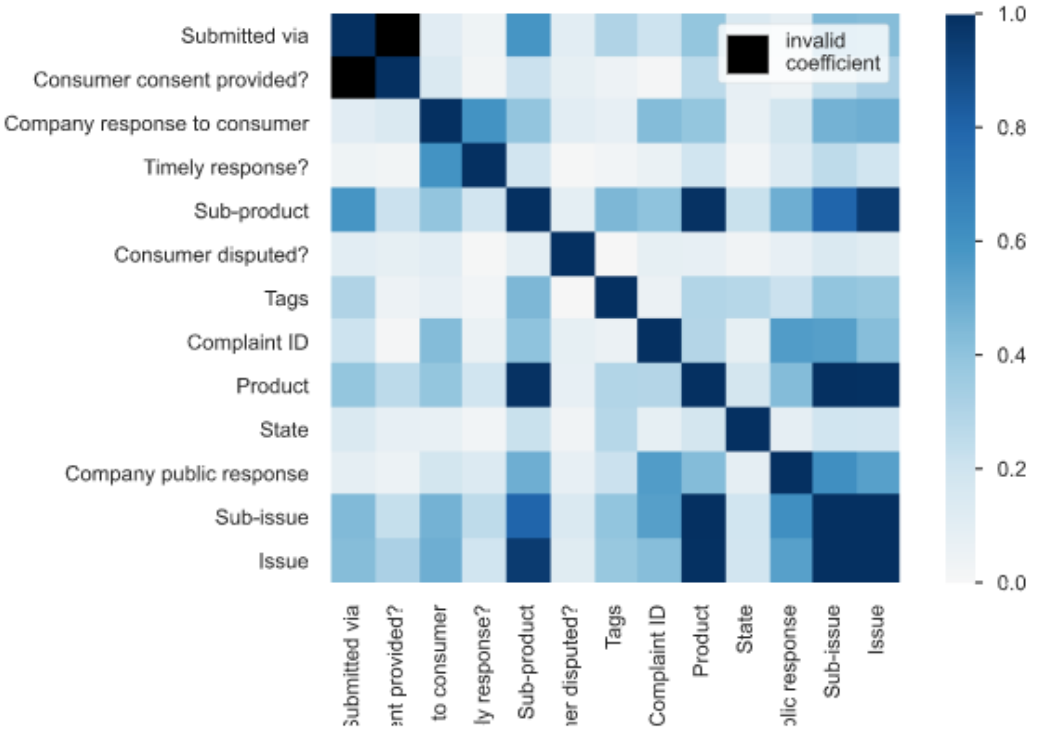
UNIQUE

Distinct	670598
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

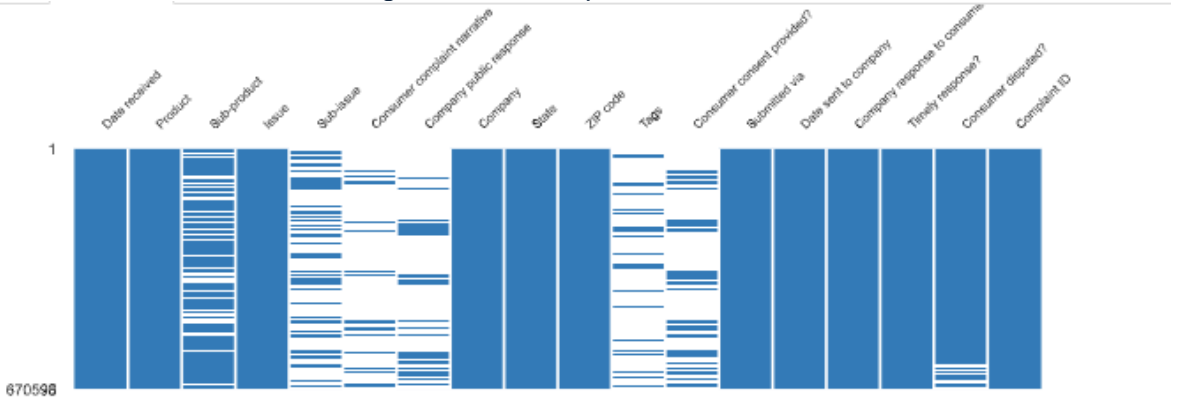
Mean	1145473.241
Minimum	1
Maximum	2218987
Zeros	0
Zeros (%)	0.0%
Memory size	5.1 MiB



The data was then checked for correlation and this the matrix that was formed. This shows the strong dependency between some columns but also the invalid coefficient error. We try to solve this by cleansing this data.



We also see the amount of missing data with this plot.



We then perform some EDA and data cleansing to improve this data. The following steps were performed to fix the data.

- Renaming Columns to remove spaces between words.

```
# Renaming Columns

df2.rename(columns = {'Date received' : 'Date_Received', 'Product' : 'Product', 'Sub-product' : 'Sub_Product' ,
                      'Issue' : 'Issue' , 'Sub-issue' : 'Sub_Issue' ,
                       'Consumer complaint narrative' : 'Consumer_Complaint_Narrative' ,
                      'Company public response' : 'Company_Public_Response' , 'Company' : 'Company' ,
                      'State' : 'State' , 'ZIP code' : 'Zip_Code' , 'Tags' : 'Tags' ,
                  'Consumer consent provided?' : 'Consumer_Consent_Provided' ,
                      'Submitted via' : 'Submitted_Via' , 'Date sent to company' : 'Data_Sent_To_Company' ,
                       'Company response to consumer' : 'Company_Response_To_Consumer' ,
                  'Timely response?' : 'Timely_Response' , 'Consumer disputed?' : 'Consumer_Disputed' ,
                      'Complaint ID' : 'Complaint_ID'}, inplace = True)
```

- Filling empty cells with ‘unknown’

```
df3.Sub_Product = df2.Sub_Product.fillna('Unknown')
df3.Sub_Issue = df2.Sub_Issue.fillna('Unknown')
df3.Consumer_Complaint_Narrative = df2.Consumer_Complaint_Narrative.fillna('Unknown')
df3.Company_Public_Response = df2.Company_Public_Response.fillna('Unknown')
df3.Tags = df2.Tags.fillna('No_Tags')
df3.Consumer_Consent_Provided = df2.Consumer_Consent_Provided.fillna('Unknown')
df3.Consumer_Disputed = df2.Consumer_Disputed.fillna('Unknown')
df3.State = df2.State.fillna('Unknown')
df3.Zip_Code = df2.Zip_Code.fillna('00000')
```

- Date_Received and Data_Sent_To_Company are converted to datetime datatype. Multiple values are replaced in different columns to make the dataset as consistent as possible.

```
df3['Date_Received'] = pd.to_datetime(df3['Date_Received'])
df3['Data_Sent_To_Company'] = pd.to_datetime(df3['Data_Sent_To_Company'])
df3['Zip_Code'] = df3['Zip_Code'].str.replace('X', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('-', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('`', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('[', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace(' ', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('+', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('.', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('/', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('$', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('!', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('*', '0')
df3['Zip_Code'] = df3['Zip_Code'].str.replace('(', '0')
df3['Product'] = df3['Product'].str.replace(' ', '_')
df3['Sub_Product'] = df3['Sub_Product'].str.replace(' ', '_')
df3['Issue'] = df3['Issue'].str.replace(' ', '_')
df3['Sub_Issue'] = df3['Sub_Issue'].str.replace(' ', '_')
df3['Consumer_Complaint_Narrative'] = df3['Consumer_Complaint_Narrative'].str.replace(' ', '_')
df3['Company_Public_Response'] = df3['Company_Public_Response'].str.replace(' ', '_')
df3['Company'] = df3['Company'].str.replace(' ', '_')
df3['Tags'] = df3['Tags'].str.replace(' ', '_')
df3['Consumer_Consent_Provided'] = df3['Consumer_Consent_Provided'].str.replace(' ', '_')
df3['Company_Response_To_Consumer'] = df3['Company_Response_To_Consumer'].str.replace(' ', '_')
df3
```

- Reording the columns.

```
df4 = df3[['Complaint_ID', 'Company', 'Date_Received', 'Data_Sent_To_Company', 'Product',
           'Sub_Product','Issue', 'Sub_Issue', 'Submitted_Via', 'Consumer_Complaint_Narrative',
           'State', 'Zip_Code', 'Tags','Company_Public_Response', 'Company_Response_To_Consumer',
           'Timely_Response', 'Consumer_Disputed','Consumer_Consent_Provided']]
```

- Zip code was converted into int datatype.

```
df4['Zip_Code'] = df4['Zip_Code'].astype(int)
df4
```

- Finally, the new dataset is downloaded as a csv.

```
df4.to_csv('Post_EDA.csv', index = False)
```

```
df4.shape
```

```
(670598, 18)
```

- Our final dataset after cleansing.

Complaint_ID	Company	Date_Received	Data_Sent_To_Company	Product	Sub_Product
468882	Wells_Fargo_&_Company	2013-07-29	2013-07-30	Consumer_Loan	Vehicle_loan
468889	Wells_Fargo_&_Company	2013-07-29	2013-07-31	Bank_account_or_service	Checking_account
468879	Santander_Bank_US	2013-07-29	2013-07-31	Bank_account_or_service	Checking_account
468949	Wells_Fargo_&_Company	2013-07-29	2013-07-30	Bank_account_or_service	Checking_account
475823	Franklin_Credit_Management	2013-07-29	2013-07-30	Mortgage	Conventional_fixed_mortgage
...	...	...	...	...	...
2211891	Equifax	2016-11-16	2016-11-16	Credit_reporting	Unknown
2179768	Citibank	2016-10-26	2016-10-26	Mortgage	Conventional_adjustable_mortgage_(ARM)
2212094	Equifax	2016-11-17	2016-11-17	Credit_reporting	Unknown
2126003	Amex	2016-09-22	2016-09-22	Credit_card	Unknown
2062419	BancorpSouth_Bank	2016-08-13	2016-09-07	Consumer_Loan	Installment_loan

Post EDA the report was taken, and this is the result of the process.

- Missing valued have been fixed.

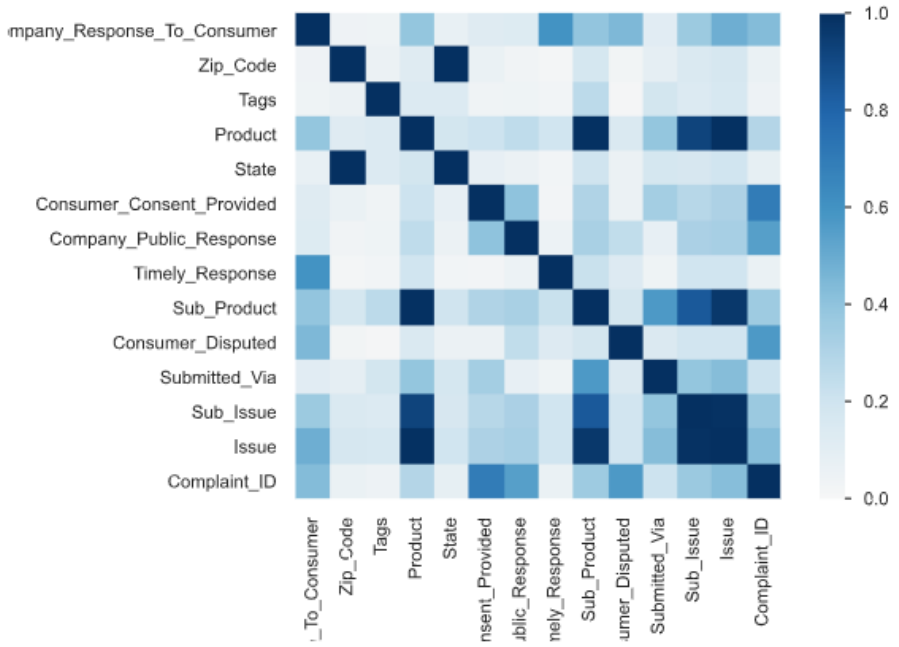
Dataset statistics

Number of variables	18
Number of observations	670598
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	92.1 MiB
Average record size in memory	144.0 B

Variable types

Numeric	2
Categorical	15
Boolean	1

- The correlation has improved greatly. We do not have any invalid correlations anymore.



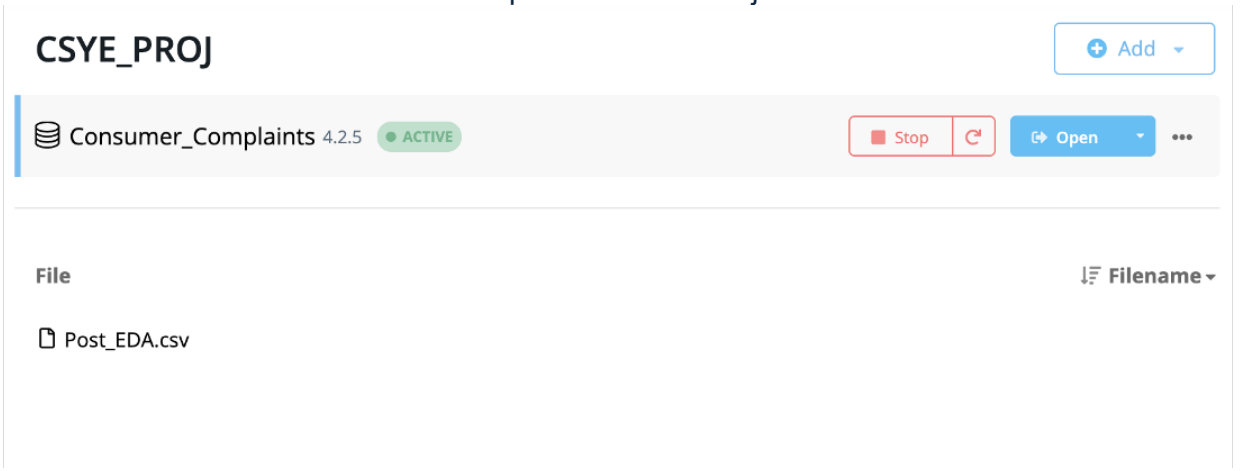
4. Database Installation

Neo4j

Neo4j is a NoSQL Graph database that uses Cypher Query Language to perform operations on the data that is being loaded.

The first step in Neo4j is to create a new Project where the Database can pe added. We created a project called “CSYE_PROJ”.

Adding the database is the next step. We have Created a local DBMS “Consumer_Complaints” and then the dataset is added to the import folder of Neo4j.



5. Data Mapping and Integration

Data Mapping begins with the creation of constraints.

```
CREATE CONSTRAINT ON (complaint:Complaint) ASSERT complaint.ComplaintId IS UNIQUE;
CREATE CONSTRAINT ON (company:Company) ASSERT company.name IS UNIQUE;
CREATE CONSTRAINT ON (response:Response) ASSERT response.companyResponse IS UNIQUE;
CREATE CONSTRAINT ON (product:Product) ASSERT product.ProductName IS UNIQUE;
CREATE CONSTRAINT ON (state:State) ASSERT state.StateName IS UNIQUE;
CREATE CONSTRAINT ON (zipcode:Zipcode) ASSERT zipcode.ZipId IS UNIQUE;
CREATE CONSTRAINT ON (subProduct:SubProduct) ASSERT subProduct.SubProductName IS UNIQUE;
CREATE CONSTRAINT ON (issue:Issue) ASSERT issue.IssueName IS UNIQUE;
CREATE CONSTRAINT ON (subIssue:SubIssue) ASSERT subIssue.SubIssueName IS UNIQUE;
```

Creating Nodes Complaint and Company with their Relationship.

```
//Create Node Complaint
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
ON CREATE SET complaint.DateReceived = row.Date_Received;
```

```
//Create Node Company
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (company:Company {name:row.Company});
```

```
//Create Relationship complaint-against-company
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (company:Company {name:row.Company})
MERGE (complaint)-[:AGAINST]→(company);
```

Creating Node Response and its Relationship with Company.

```
//Create Node Response
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (response:Response {companyResponse:row.Company_Response_To_Consumer});
```

```
//Create Relationship response-from-company
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (company:Company {name:row.Company})
MATCH (response:Response {companyResponse:row.Company_Response_To_Consumer})
MERGE (response)-[:FROM]→(company);
```

Creating Nodes Product and Sub-Product and their Relationships with Complaint and each other.

```
//Create Node Product
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (product:Product {ProductName:row.Product});
```

```
//Create Relationship complaint-about-product
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (product:Product {ProductName:row.Product})
MERGE (complaint)-[:ABOUT]→(product);
```

```
//Create Node SubProduct
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (subProduct:SubProduct {SubProductName:row.Sub_Product});
```

```
//Create Relationship complaint-about-subproduct
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (subProduct:SubProduct {SubProductName:row.Sub_Product})
MERGE (complaint)-[:ABOUT]→(subProduct);
```

```
//Create Relationship product-contains-subproduct
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (product:Product {ProductName:row.Product})
MATCH (subProduct:SubProduct {SubProductName:row.Sub_Product})
MERGE (product)-[:CONTAINS]→(subProduct);
```

Creating Node State and Zip Code.

```
//Create Node State
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (state:State {StateName:row.State});
```

```
//Create Node Zip
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (zipcode:Zipcode {ZipId:toInteger(row.Zip_Code)});
```

Creating Relationship between State & Complaint and Zip Code & Complaint.

```
//Create Relationship complaint-originatedfrom-state
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (state:State {StateName:row.State})
MERGE (complaint)-[:ORIGINATED_FROM]→(state);
```

```
//Create Relationship complaint-originatedfrom-zip
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (zipcode:Zipcode {ZipId:toInteger(row.Zip_Code)})
MERGE (complaint)-[:ORIGINATED_FROM]→(zipcode);
```

Creating Nodes Issue & Sub Issue and their Relationship with Complaint.

```
//Create Node Issue
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (issue:Issue {IssueName:row.Issue});
```

```
//Create Relationship complaint-with-issue
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (issue:Issue {IssueName:row.Issue})
MERGE (complaint)-[:WITH]→(issue);
```

```
//Create Node SubIssue
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MERGE (subIssue:SubIssue {SubIssueName:row.Sub_Issue});
```

```
//Create Relationship complaint-with-subissue
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (complaint:Complaint {ComplaintId:toInteger(row.Complaint_ID)})
MATCH (subIssue:SubIssue {SubIssueName:row.Sub_Issue})
MERGE (complaint)-[:WITH]→(subIssue);
```

Creating Relationship between Issue and Sub Issue.

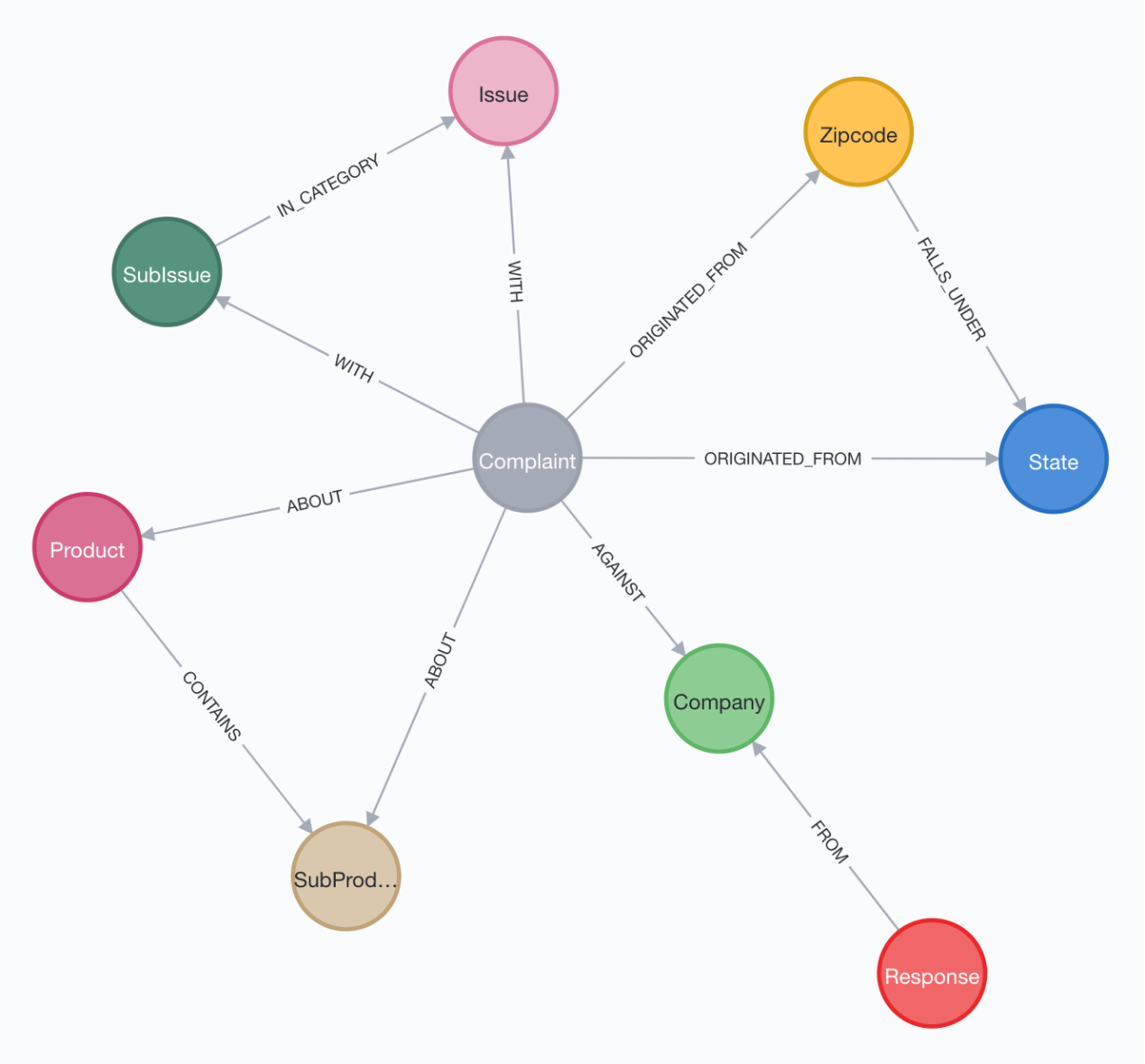
```
//Create Relationship subissue-incategory-issue
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (issue:Issue {IssueName:row.Issue})
MATCH (subIssue:SubIssue {SubIssueName:row.Sub_Issue})
MERGE (subIssue)-[:IN_CATEGORY]→(issue);
```

Creating Relationship between Zip Code and State.

```
//Create Relationship Zip-fallsunder-State
:auto USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///Post_EDA.csv' AS row
MATCH (state:State {StateName:row.State})
MATCH (zipcode:Zipcode {ZipId:toInteger(row.Zip_Code)})
MERGE (zipcode)-[:FALLS_UNDER]→(state);
```

Schema Visualization.

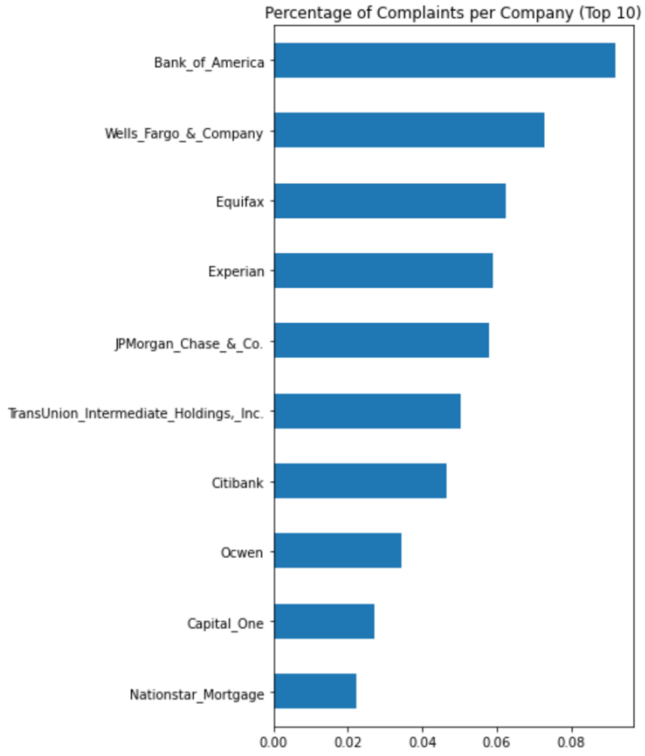
```
call db.schema.visualization();
```



6. Data Validation and Data Visualization

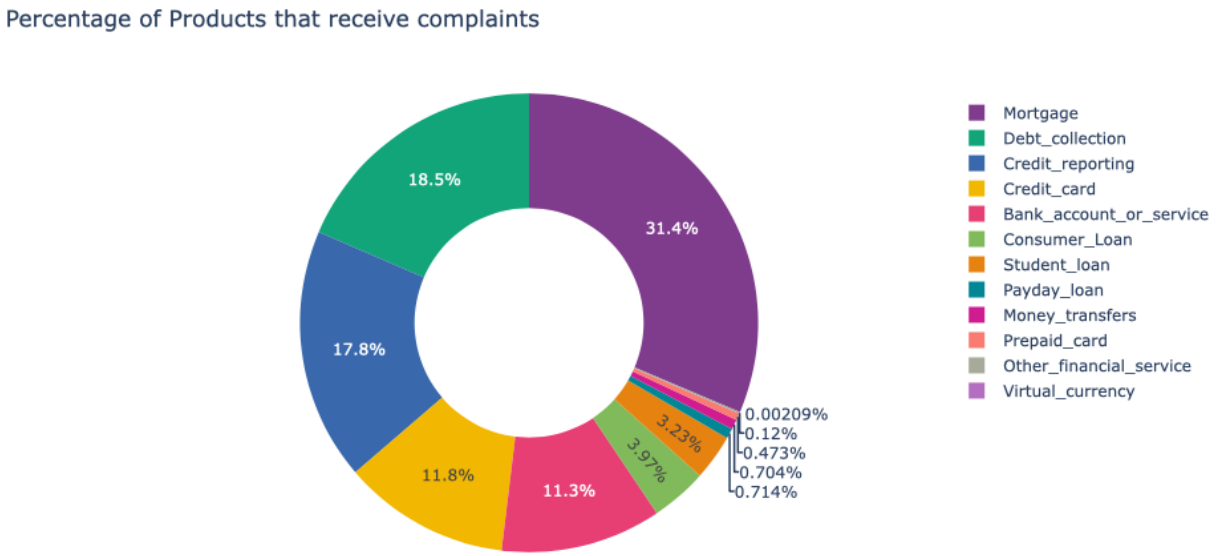
The bar chart shows the top 10 companies that have received the maximum amount of complaints

```
fig_dims = (5, 10)
fig, ax = plt.subplots(figsize=fig_dims)
df1['Company'].value_counts(normalize=True).sort_values(ascending=True)[-10:].plot(ax=ax, kind='barh')
ax.set_title('Percentage of Complaints per Company (Top 10)')
```



This donut plot displays the product the complaint is for.

```
pie_chart_1 = px.pie(data_frame= dfdict, values = 'Number_Of_Complaints', names = 'Product',
                  title = 'Percentage of Products that receive complaints',
                     color_discrete_sequence=px.colors.qualitative.Bold, hole=0.5)
pie_chart_1.update_layout(showlegend=True)
```

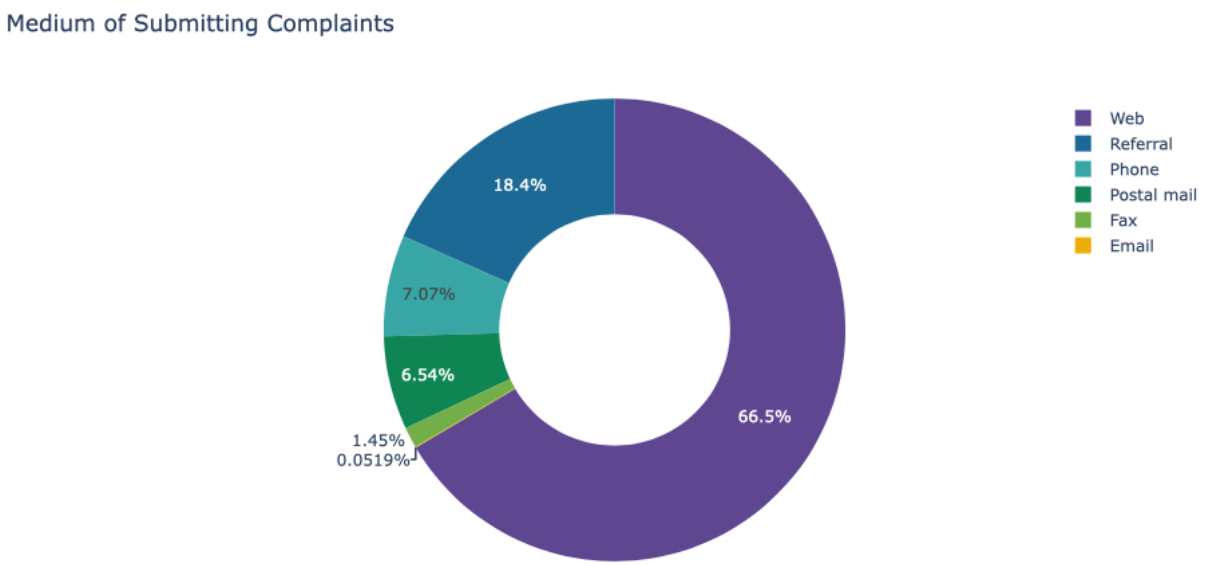


This plot shows the way the complaint was filed.

```
df_via = df1.groupby(["Submitted_Via"]).size().reset_index(name="Number_Of_Complaints")
df_via.head()
```

	Submitted_Via	Number_Of_Complaints
0	Email	348
1	Fax	9733
2	Phone	47389
3	Postal mail	43869
4	Referral	123224

```
pie_chart_2 = px.pie(data_frame= df_via, values = 'Number_Of_Complaints', names = 'Submitted_Via',
                     title = 'Medium of Submitting Complaints',color_discrete_sequence=px.colors.qualitative.Prism,
                       hole=0.5)
pie_chart_2.update_layout(showlegend=True)
```

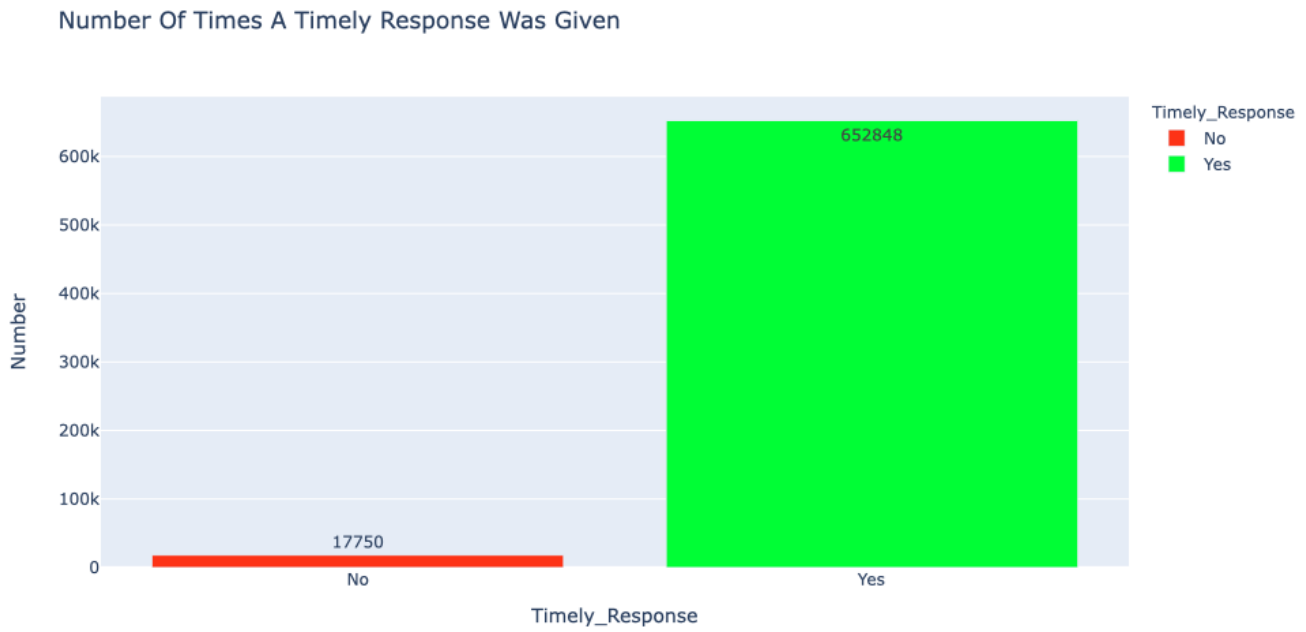


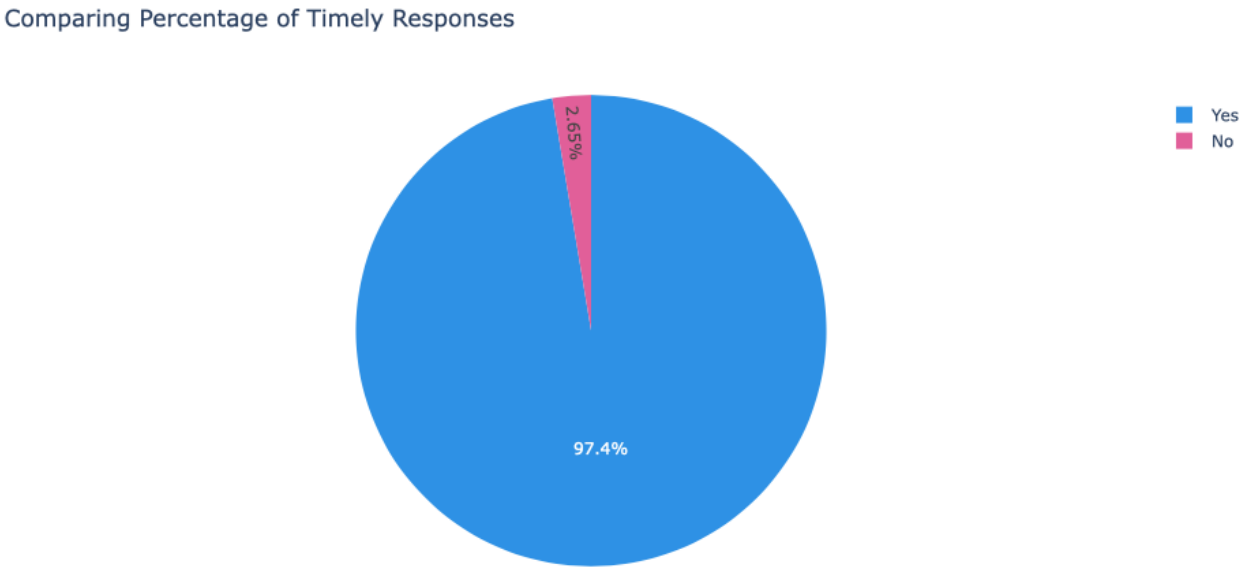
This bar chart displays the timely vs untimely response graph.

```
timely_resp_dict = df1.groupby(["Timely_Response"]).size().reset_index(name="Number")
timely_resp_dict.head()
```

	Timely_Response	Number
0	No	17750
1	Yes	652848

```
bar_chart_1 = px.bar(data_frame = timely_resp_dict, x = 'Timely_Response', y = 'Number', color = 'Timely_Response',
                     title = 'Number Of Times A Timely Response Was Given',text = 'Number',
                     color_discrete_sequence=px.colors.qualitative.Light24)
bar_chart_1
```



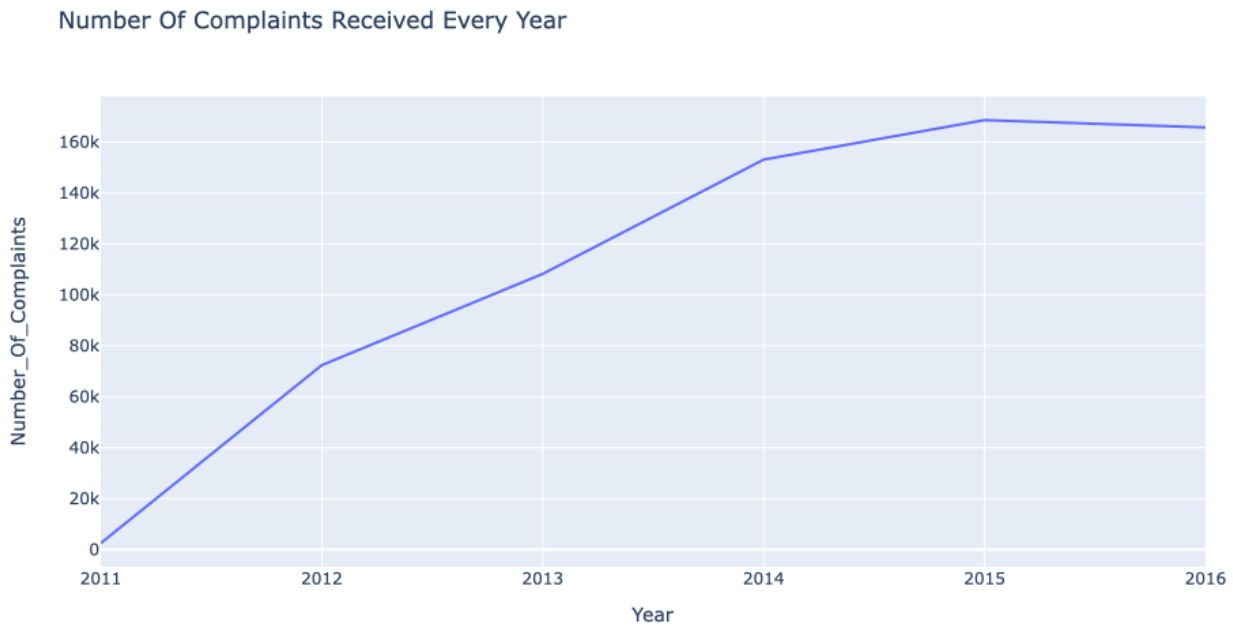


This table show the cumulative complaints over the years.

```
df_date = df1.groupby(["Year"]).size().reset_index(name="Number_Of_Complaints")
df_date.head()
```

	Year	Number_Of_Complaints
0	2011	2540
1	2012	72403
2	2013	108226
3	2014	153088
4	2015	168600

```
lineplot = px.line(data_frame= df_date, x ='Year', y = 'Number_Of_Complaints',
                   title = 'Number Of Complaints Received Every Year')
lineplot.update_layout(showlegend=True)
```

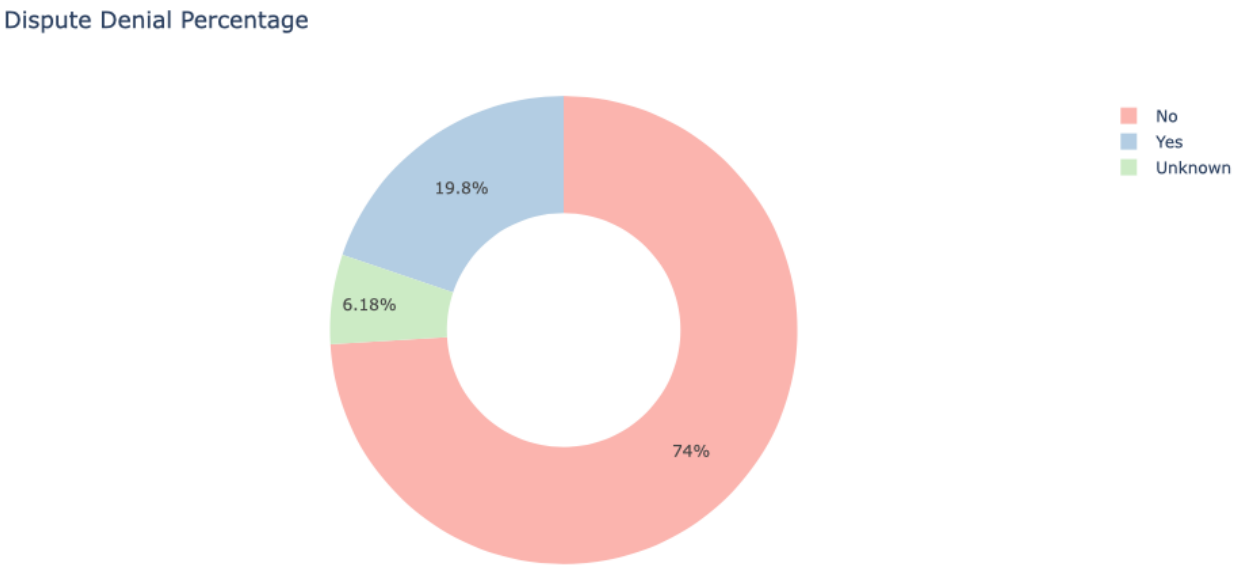


This plot shows the dispute denial percentage

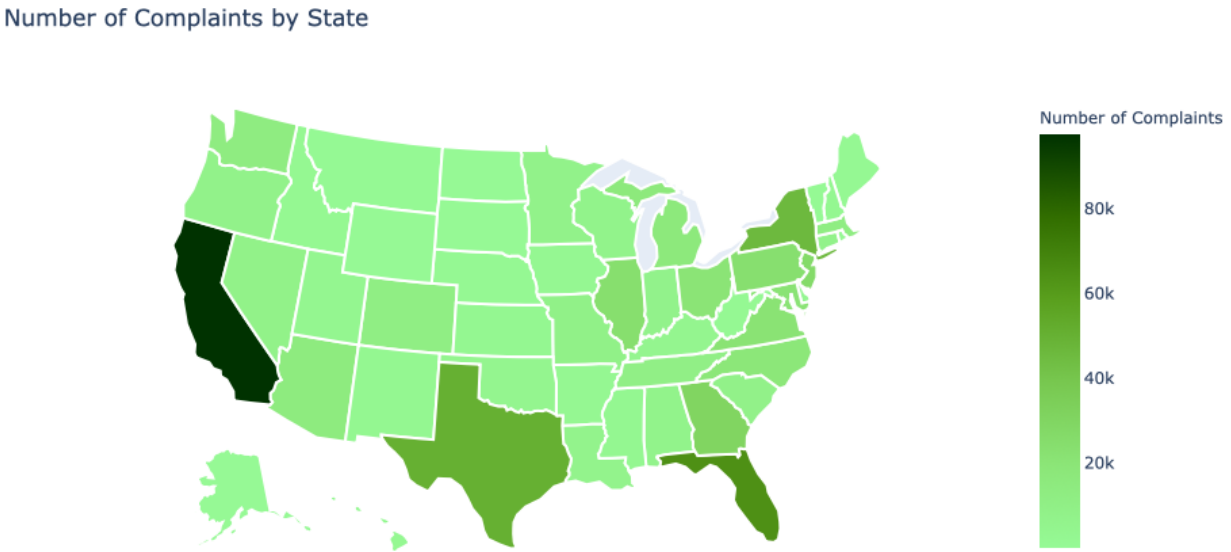
```
df_disputed = df1.groupby(["Consumer_Disputed"]).size().reset_index(name="Number_Of_Complaints")
df_disputed.head()
```

	Consumer_Disputed	Number_Of_Complaints
0	No	496466
1	Unknown	41419
2	Yes	132713

```
pie_chart_4 = px.pie(data_frame= df_disputed, values = 'Number_Of_Complaints', names = 'Consumer_Disputed',
                     title = 'Dispute Denial Percentage',color_discrete_sequence=px.colors.qualitative.Pastel1,
                        hole=0.5)
pie_chart_4.update_layout(showlegend=True)
```



This plot shows the map for most complaints filed from state.



7. System Integration and User Acceptance Testing

System integration was tested prior to ingesting data in the database.

The following piece of code was run to validate system connection. The py2neo is the package. It is open source and available to download using the PIP package manager.

The neo4j is running on localhost and the authentication is passed as parameter.

```
from py2neo import Graph
graph = Graph("bolt://localhost:7687", auth=("neo4j", "neo4j"))
try:
    graph.run("Match () Return 1 Limit 1")
    print('Connection successful')
except Exception:
    print('Connection unsuccessful')
```

We connect to the database and send a query which if runs without error means the database is connected.

Upon no error we print **Connection Successful**

If the code returns some error, we say the **Connection Unsuccessful**

The output of the above code concludes system integration testing.

For integration testing, we test the contents of both the input file and the data inserted in the database.

For Data validation we check the count of the rows of data in the neo4j database and the python script used for EDA.

The following is the output for the neo4j row count query.

The match statement is used to match all the rows in the Complaint column, and we return the count of rows.

```
MATCH (:Complaint)
RETURN count(*) as count
```

	count
1	670598

```
df1= pd.read_csv ("Post_EDA.csv")
df1.shape[0]
```

```
670598
```

Count of all rows was cross verified with the input csv file using Python.

This concludes integration testing.

User acceptance testing:

User Acceptance Testing, also known as UAT testing means **A process of verifying that a solution works for the user.**

The ideal workflow will ideally be a user entering the data and it will successfully get inserted into the database.

The issue that could arise in this is the data is either invalid or has inconsistencies.

To validate the user can check the data that is to be inserted in the database using the python script to count the number of rows. Then we use a second script to check the amount of rows in the neo4j.

If the output of both the query and the python code match then we can conclude the application is working perfectly and this concludes user acceptance testing.

8. Challenges Encountered

The following are the challenges we encountered

- Data mapping

  Here we had to collectively understand what the data provided to us contained. We had to understand the relationship between the columns.

- Data visualization

  For visualization, a new package had to installed to build interactive graphs and plots

  Zip code and State relationship had to be built to plot the data on an interactive map

- Neo4j

  The concepts and inner working of this new database had a learning curve.

  The working knowledge of how nodes and relations are built in Neo4j had to be figured out.

  New syntax had to me understood to efficiently query the database and insert data into it.

9. End User Instructions

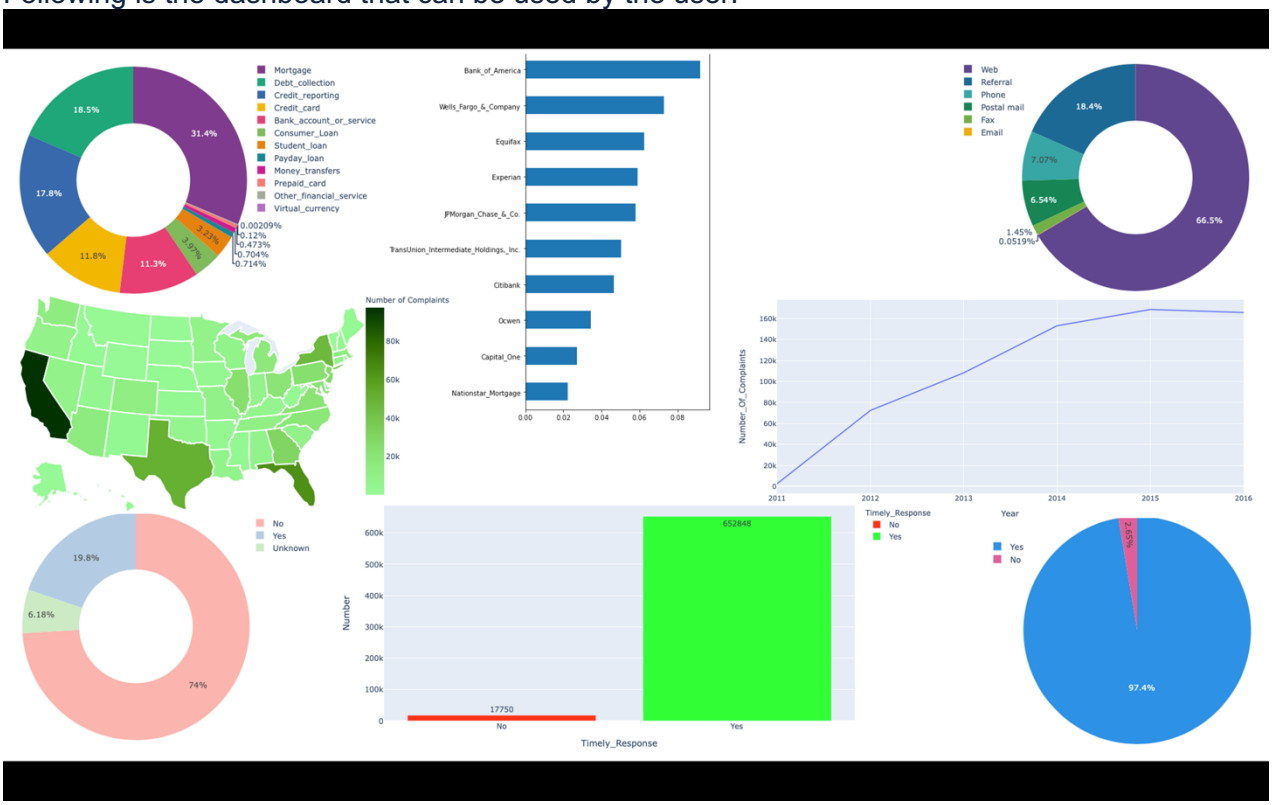
The end user for our project will be the **Consumer Financial Protection Bureau** or **CFPB.** The dataset and the dashboard created by us will be helpful for them to protect people from getting into trouble due to the financial decision that they make.

Database Creation and loading

- Open Neo4j and create a project, followed by creating a local database.
- Load the Post_EDA.csv file to the import folder.
- Start and then open the database.
- Create constraints followed by nodes and relationships.
- Visualize the schema.

Visualization interpretation

Following is the dashboard that can be used by the user:



The dashboard can be interpreted as:

- A personal from CFPB can look at the dashboard to identify the companies that have been receiving the maximum number of complaints and set up an investigation against them to improve the quality of services provided.

- One can also see the products that need careful consideration before investing the money into. Since, high number of complaints is not a good sign.
- The states from where, the complaints filed are maximum can be evaluated using the map. Further, the state government rules can be modified if the companies are exploiting the consumers.
- There are a few companies that have denied any dispute on the complaints being filed. The CFPB can get in touch with them to identify the problems and act if necessary.
- The dashboard shows that the number of complaints has been increasing every subsequent year. This can be an indication that there might be something wrong with the policies that allow companies to make rules that the consumer is not happy with.