2021

Big Data Architecture & Governance

Northeastern University

Assignment Name:

US Permanent Visas

Student Names:

Prathamesh Verlekar

Yash Pandya

Saurabh Satra

1. Contents

2. Assignment....2
2.1. Case....2
2.2. Assignment Goals....2
2.2.1. Visualization Deliverables....3
2.2.2. OTHER DELIVERABLES....3
3. Documentation....4
3.1. Vision Diagram....4
3.2. Data Wrangling and Cleansing....4
3.3. Database Installation....11
3.4. Data Mapping and Integration....12
3.5. Data Validation and Data Visualization....18
3.6. System Integration and User Acceptance Testing....25
3.7. Challenges Encountered....27
3.8. End User Instructions....28

2. Assignment

2.1. 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.

.

2.2. 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:
    - Database Creation and load.
    - Visualization interpretation - describe information regarding your findings.

2.2.1. 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:

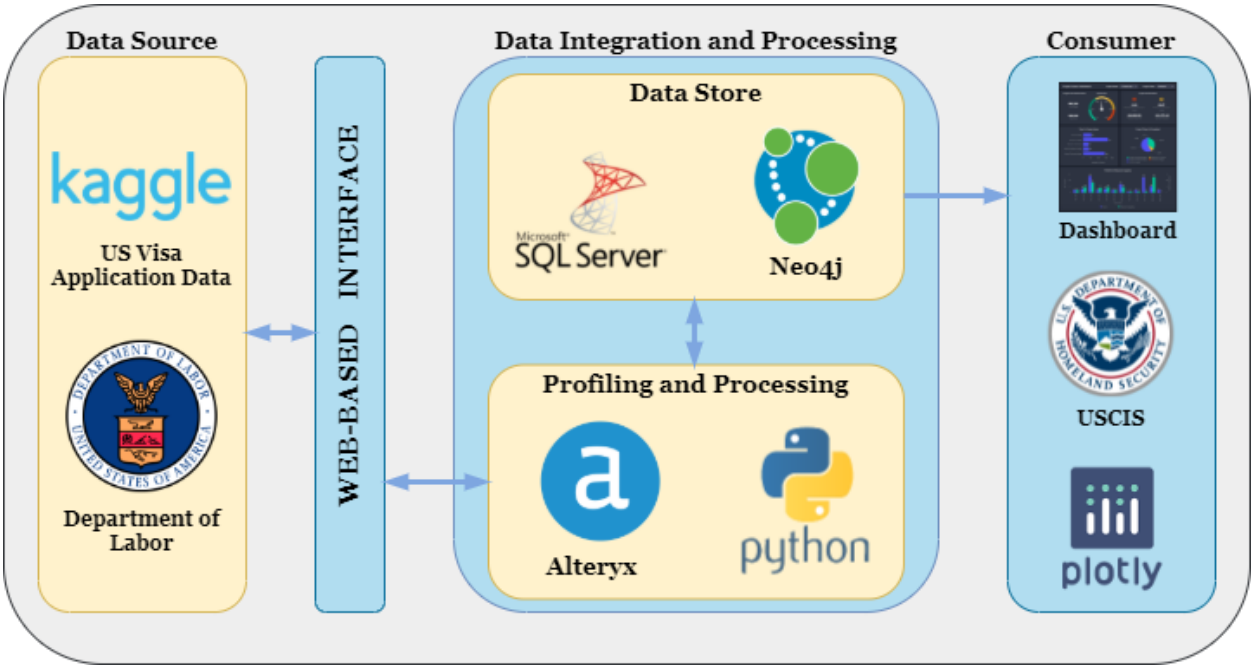
- Columns used for dimensions, and columns that are used for measurement.
- How would you generate new dimensions?
- Who would use this dashboard and how they benefit from your dashboard?
- What value would be generated using this dashboard?

2.2.2. 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.

3. Documentation

3.1. Vision Diagram



3.2. Data Wrangling and Cleansing

About Dataset

Data covers 2011-2016 visa applications and includes information on employer, position, wage offered, job posting history, employee education and past visa history, and final decision.

This data was collected and distributed by the US Department of Labor.

Important Columns in the dataset

Case Number	Unique identifier assigned to each application submitted for processing
Application Year	Year during which the application was submitted for processing
Application Month	Month during which the application was submitted for processing

Application Day	Day during which the application was submitted for processing
Application Decision	Status associated with the last significant event or decision. Example: Approved or Denied
Country of Citizenship	Country of citizenship of the foreign worker being sponsored
Country Latitude	Derived Column showing Latitude Coordinates for each country
Country Longitude	Derived Column showing Longitude Coordinates for each country
Visa Type	Class of immigration visa the foreign worker held. Example: F1, H1-B
SOC Code	Standard Occupational code associated with the job being requested for permanent labor certification
SOC Title	Standard Occupational Code Title of the permanent job.
Job City	City in which the foreign worker is expected to be employed
Job State	State in which the foreign worker is expected to be employed
Employer Name	Name of employer requesting permanent labor certification
Employer Address	Address information of the employer requesting permanent labor certification.
Employer City	City name of the employer requesting permanent labor certification.
Employer State	State name of the employer requesting permanent labor certification.
Employer Postal Code	Postal Code of the employer requesting permanent labor certification.
Wage Amount	Prevailing wage for the job being requested for permanent labor certification
Wage Level	Level of the prevailing wage determination. Valid values include “Level I,” “Level II,” “Level III,” and “Level IV”
Wage Range	Derived Column where Wage Amount is split into different Ranges
Job Title	Industry title associated with the North American Industrial Classification System (NAICS) code
Job Sector	Major economic sector associated with the NAICS code of the employer.

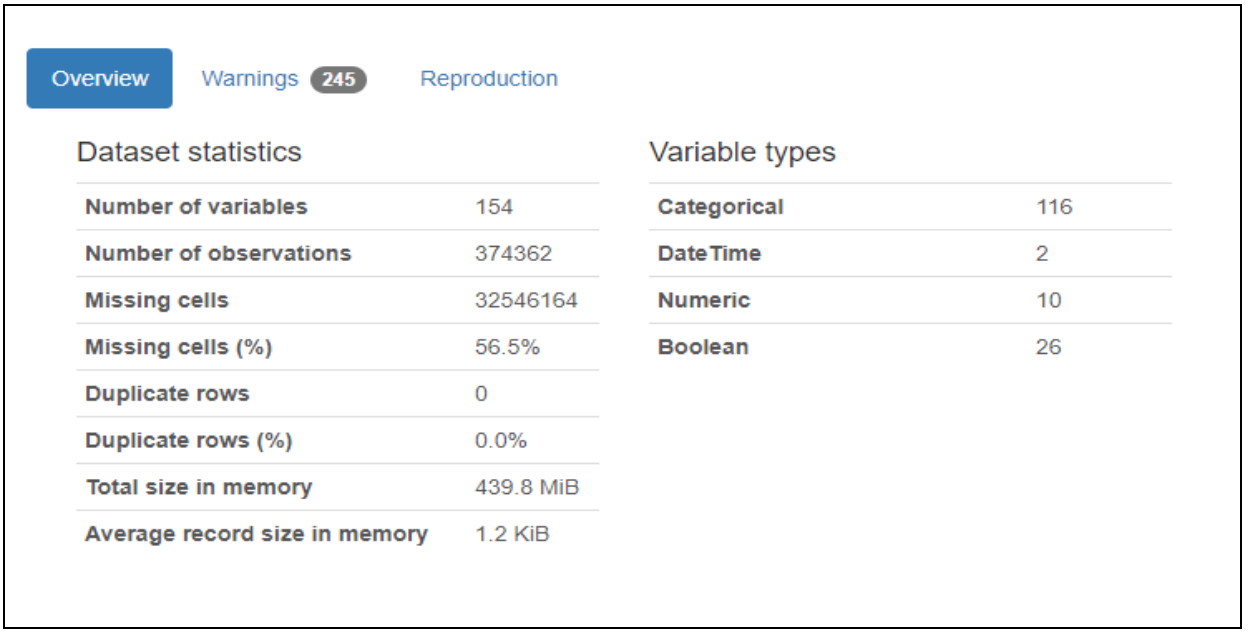
Data Profiling

Pandas Profiling

Pandas profiling is an open-source Python module with which we can quickly do an **exploratory data analysis** with just a few lines of code.

it also generates interactive reports in web format that can be presented to any person, even if they do not know programming.

Using Pandas Profiling on our dataset



From the overview, we can analyze that the dataset has **374362** observations out of which **373025** are unique observations. The dataset has 154 variables out of which only 21 variables have more than 330000 non-missing observations.

The higher Missing cells % is because of the remaining variables which have more than 50% of missing values.

The Dataset has,

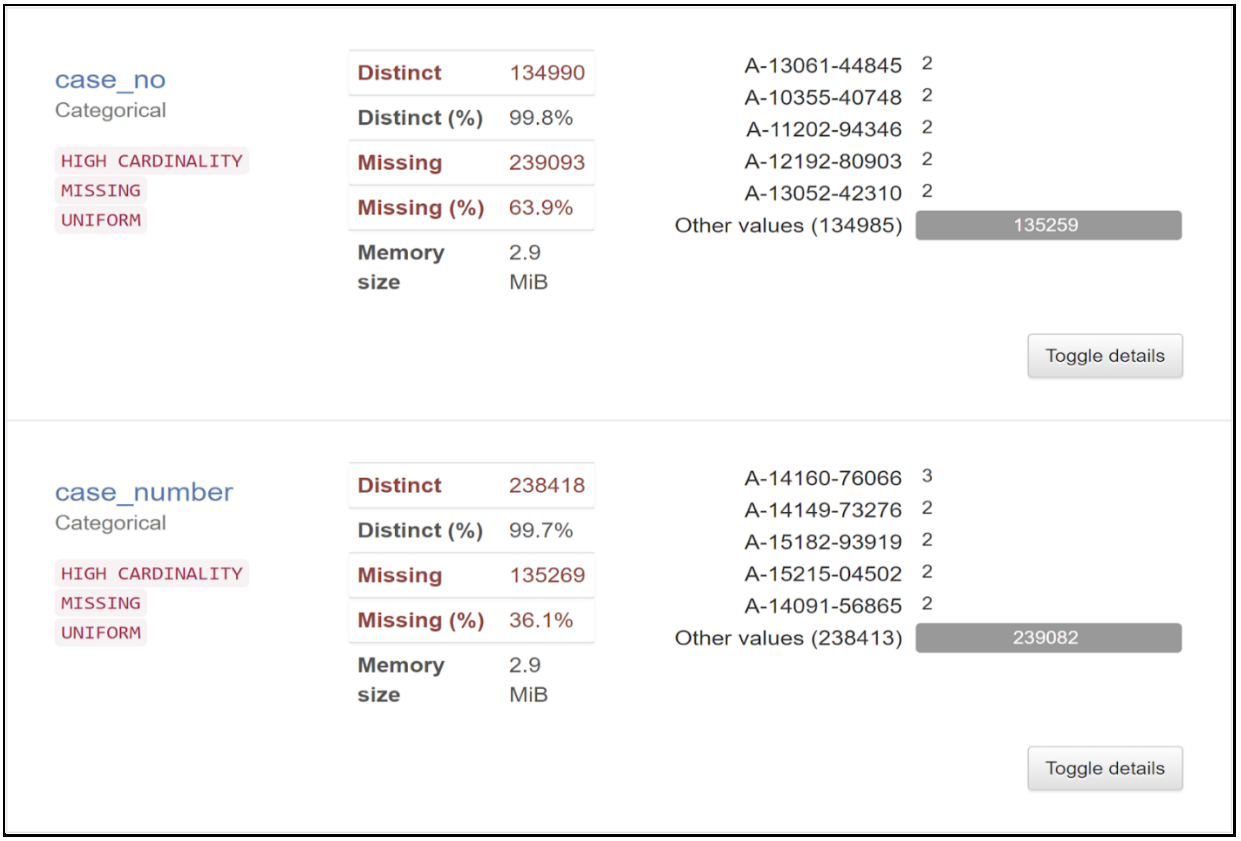
116 Categorical values

2 Date Time values

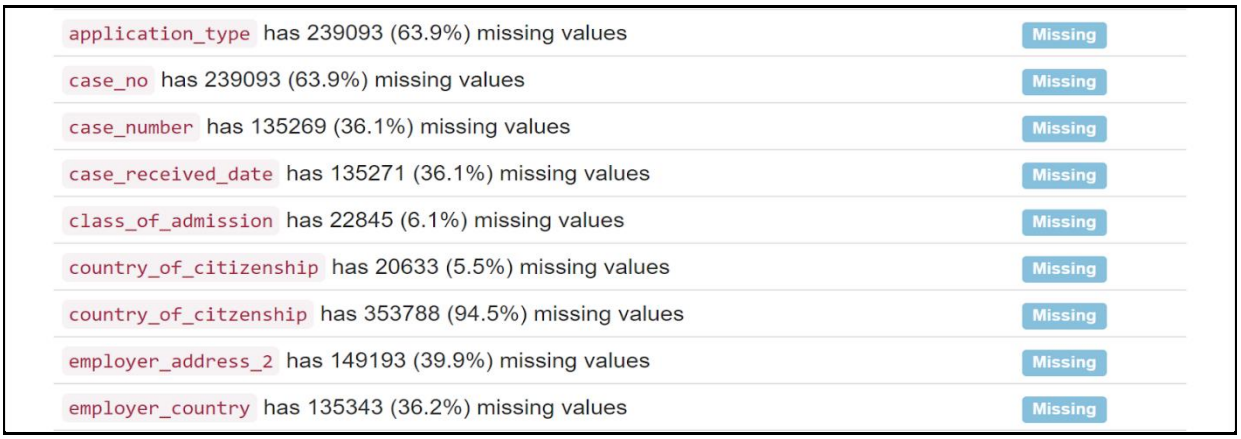
10 Numerical values

26 Boolean values

There are two columns one **case_no** and other **case_number**, we have data for either one which means these two columns are the same.

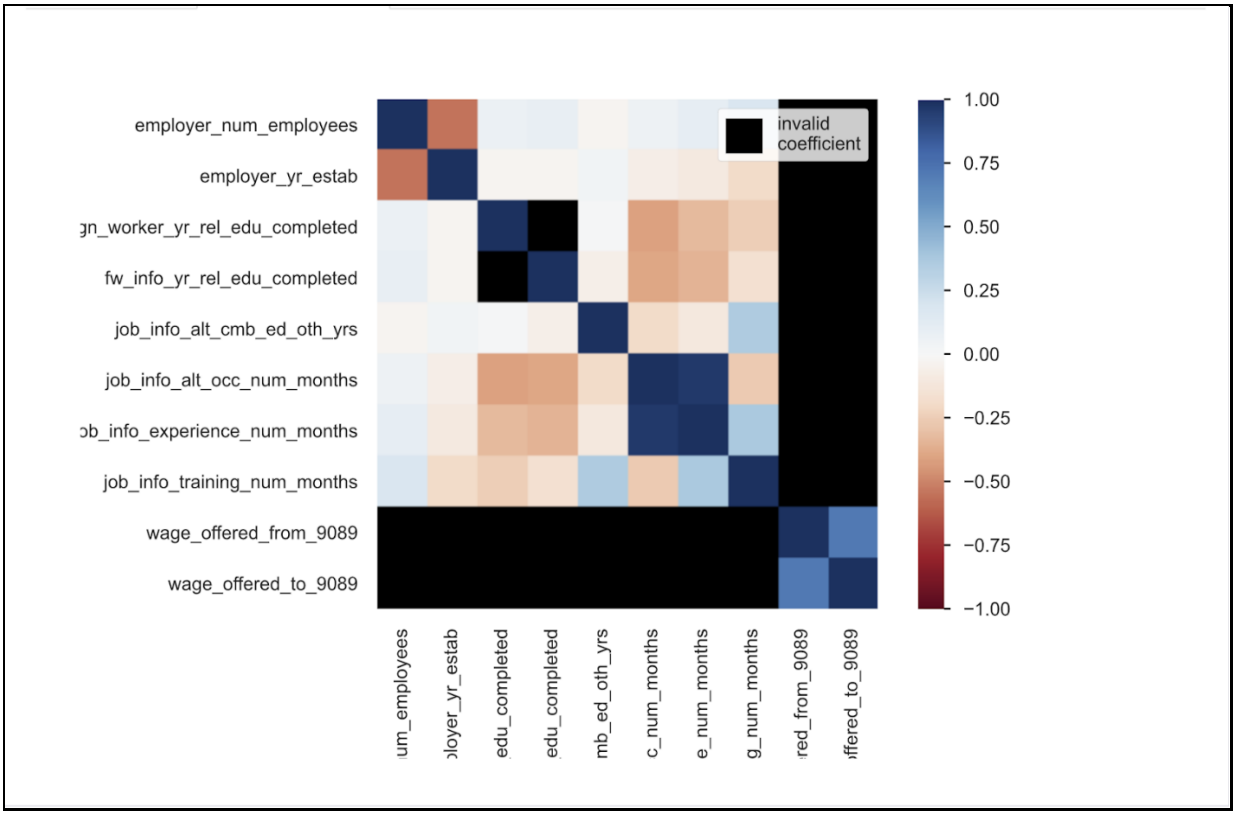


Percentage of Missing values in the columns selected for analysis



Correlation

The correlation between columns shows that most of the columns have invalid coefficient for correlation.



Important Column to consider for analysis case_status, which can be Approved, Denied or Withdrawn.

Number of unique observations

```
Approved     330519
Denied        25649
Withdrawn     18194
Name: case_status, dtype: int64
```

Data Preprocessing

We created a new column **casenumber** by merging two columns **case_no** and **case_number**. The new created column **casenumber** has **373025** unique observations.

For handling Missing values in the dataset for selected columns, we have replaced them with a string value **"Not Provided"**.

We used permanent wage amount column and permanent wage unit of pay column to derive a new column that has annual salary values

Example

	pw_amount_9089	pw_unit_of_pay_9089
0	75629.0	yr
1	37024.0	yr
2	47923.0	yr
3	10.97	hr
4	94890.0	yr
5	37024.0	yr
6	47083.33	yr
7	36733.0	yr
8	44824.0	yr
9	12.86	hr

	pw_amount_9089	pw_unit_of_pay_9089
0	75629.00	Year
1	37024.00	Year
2	47923.00	Year
3	21940.00	Year
4	94890.00	Year
5	37024.00	Year
6	47083.33	Year
7	36733.00	Year
8	44824.00	Year
9	25720.00	Year

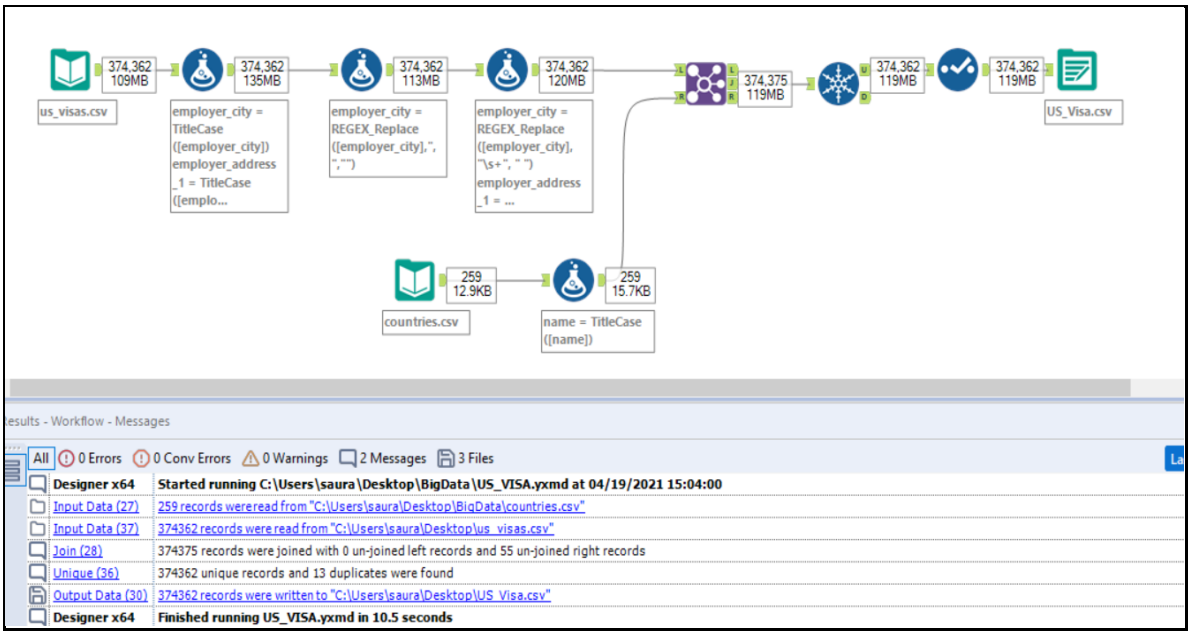
Then, we created a derived column **salary range** for analysis purpose.

```
0-30k         22035
30-60k        50217
60-90k       139278
90-120k      117579
120-150k      31931
150-180k       7420
180-210k       4007
210-240k        638
240-270k         30
270k+          1226
Name: Salary_range, dtype: int64
```

Alteryx

Alteryx is designed to make advanced analytics accessible to any data worker. We used Alteryx for Data Preprocessing.

Alteryx Workflow for Cleaning the dataset which includes removing blank spaces, extra commas, and Capitalization.



Explanation

1) The first formula tool takes care of Capitalization. It converts all the columns in the Title Case format.

2) The second formula tool replaces all the commas with a blank character.

3) The third formula tool removes all the extra whitespaces. The**" \s+"** pattern matches one or more whitespace characters, which all get replaced by a single space.

4) Finally using the join tool, we added 2 columns latitude and longitude by joining another csv file based on country name.

3.3. Database Installation

Neo4j Database

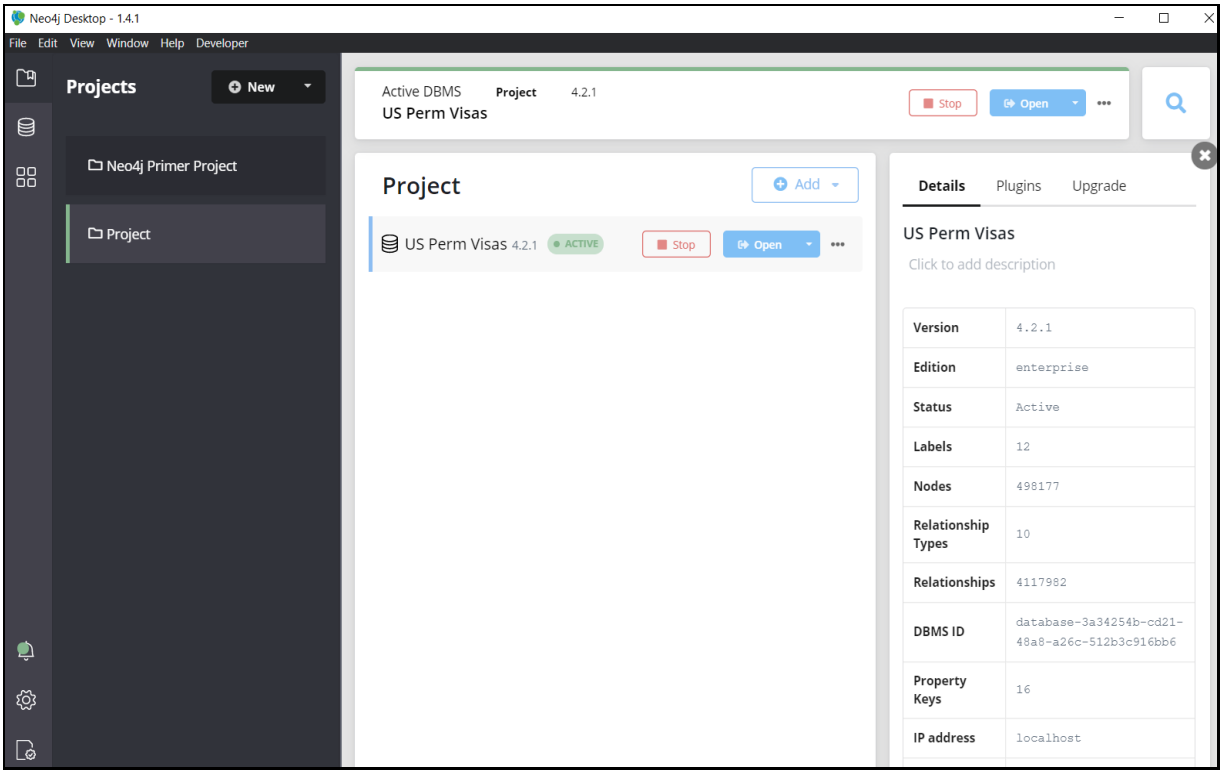
The Neo4j database is a graph database and is used to represent the data in the form of graphs. It offers data integrity and is ACID (Atomic, Consistent, Isolated, Durable) compliant. Just like RDBMS has a language called SQL to access data, the Graph database has a language called Cypher.

Neo4j can be downloaded by clicking on this link: **Download Neo4j**

Starting the Server using Neo4j Desktop

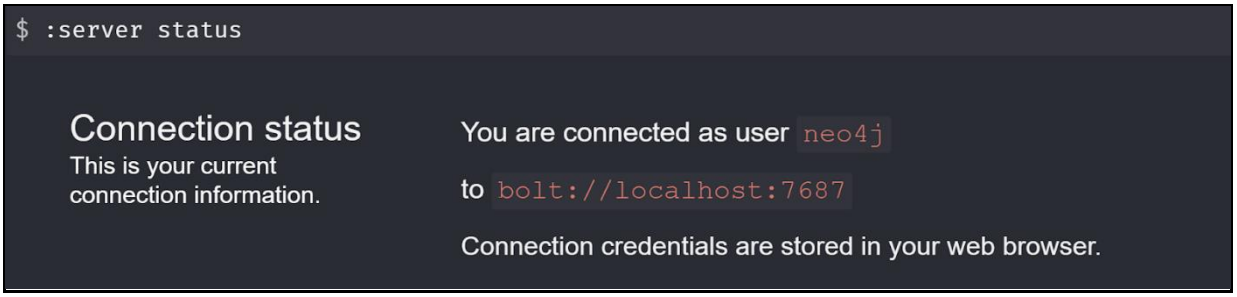
To start using **Neo4j database**, open the Neo4j Desktop installed on the system.

Now, create a database by clicking the **"Add"** button and set a password for your database. It is also possible to change the password by going into the administration tab and setting a new password. Add all the relevant files related to the database in the files section.



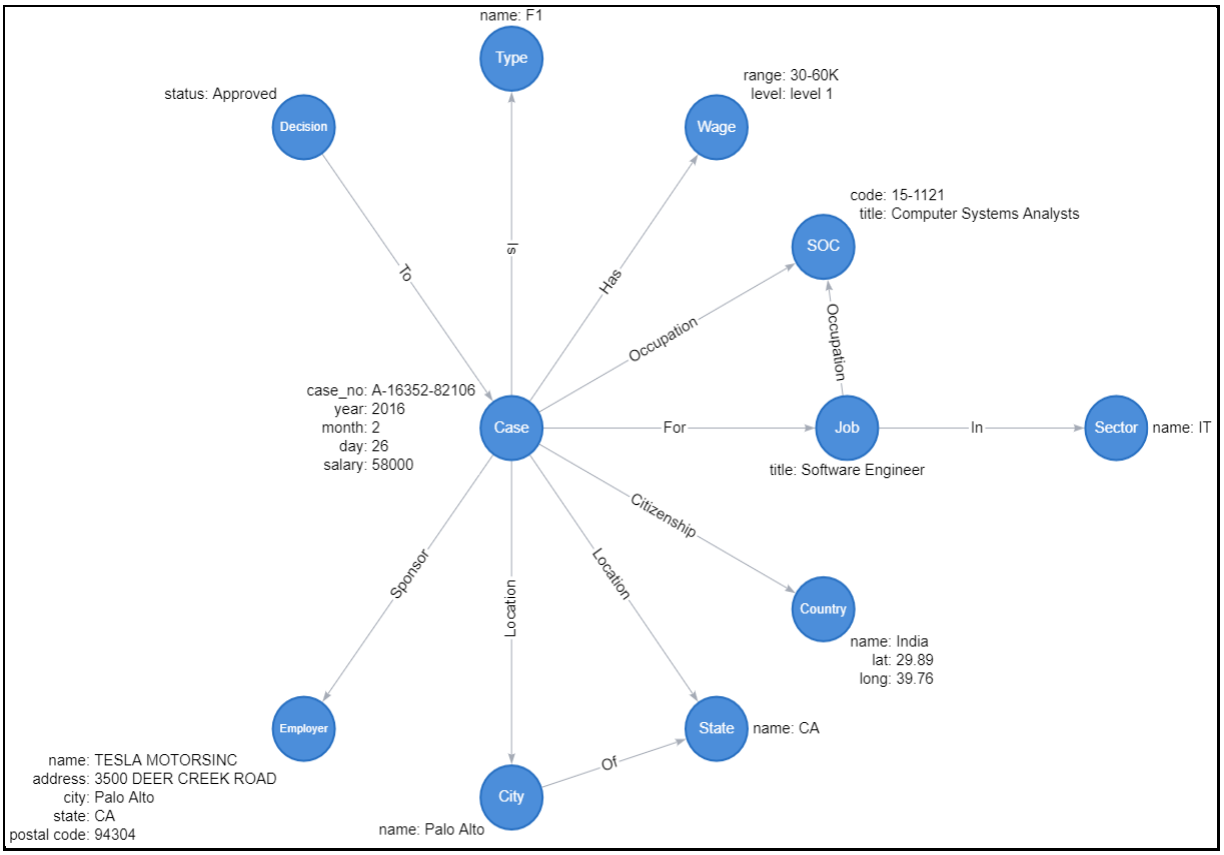
Start the server by clicking the play button on the window, stop the server or restart the server when the database is not needed. Now, the database is ready and query the data using the Neo4j Browser **"Open browser"** or through command line **"Open Terminal"**.

You can also open a new window in your preferred browser and type **http://www.localhost:7474** into the URL. To connect, you will need to enter the password you entered for your database and click **"Connect"**.



3.4. Data Mapping and Integration

Data Model



Loading US Permanent Visas dataset into Neo4j database

#Defining constraints for the database

```
CREATE CONSTRAINT ON (c:Case) ASSERT c.case_no IS UNIQUE;
CREATE CONSTRAINT ON (co:Country) ASSERT co.name is UNIQUE;
CREATE CONSTRAINT ON (t:Type) ASSERT t.name IS UNIQUE;
CREATE CONSTRAINT ON (d:Decision) ASSERT d.status IS UNIQUE;
CREATE CONSTRAINT ON (j:Job) ASSERT j.title IS UNIQUE;
CREATE CONSTRAINT ON (s:SOC) ASSERT s.code IS UNIQUE;
CREATE CONSTRAINT ON (st:State) ASSERT st.name IS UNIQUE;
CREATE CONSTRAINT ON (ci:City) ASSERT ci.name IS UNIQUE;
CREATE CONSTRAINT ON (e:Employer) ASSERT e.name IS UNIQUE;
CREATE CONSTRAINT ON (w:Wage) ASSERT w.range IS UNIQUE;
CREATE CONSTRAINT ON (se:Sector) ASSERT se.name IS UNIQUE;
```

#Loading Case and Country Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line, SPLIT(line.decision_date, '/') AS date

MERGE (case:Case {case_no: line.casenumber})
SET case.salary=TOINTEGER(line.pw_amount_9089)
SET case.month = TOINTEGER(date[0])
SET case.day = TOINTEGER(date[1])
SET case.year = TOINTEGER(date[2])
MERGE (country:Country {name: toUpper(line.country_of_citizenship) })

CREATE (case)-[:Citizenship]->(country)
```

#Loading Decision and Status Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})

MERGE (decision:Decision {status: toUpper(line.case_status) })
MERGE (type:Type {name: toUpper(line.class_of_admission) })

CREATE (decision)-[:To]->(case)
CREATE (case)-[:Is]->(type)
```

#Loading SOC Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})

MERGE (soc:SOC {code: toUpper(line.pw_soc_code)})
ON CREATE SET soc.code=toUpper(line.pw_soc_code), soc.title=toUpper(line.pw_soc_title)
ON MATCH SET soc.title=toUpper(line.pw_soc_title)

CREATE (case)-[:Occupation]->(soc)
```

#Loading Employer Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})

MERGE (employer:Employer {name: toUpper(line.employer_name)})
ON CREATE SET employer.name=toUpper(line.employer_name),
employer.address=toUpper(line.employer_address_1),
employer.city=toUpper(line.employer_city),
employer.state=(line.employer_state),
employer.postal_code=toUpper(line.employer_postal_code)
ON MATCH SET employer.address=toUpper(line.employer_address_1),
employer.city=toUpper(line.employer_city),
employer.state=(line.employer_state),
employer.postal_code=toUpper(line.employer_postal_code)

CREATE (case)-[:Sponsor]->(employer)
```

#Loading Job Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})
MERGE (job:Job {title: toUpper(line.job_info_job_title) })

CREATE (case)-[:For]->(job)
```

#Loading City and State Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})

MERGE (city:City {name: toUpper(line.job_info_work_city) })
MERGE (state:State {name: toUpper(line.job_info_work_state) })

CREATE (case)-[:Location]->(city), (city)-[:Of]->(state)
CREATE (case)-[:Location]->(state)
```

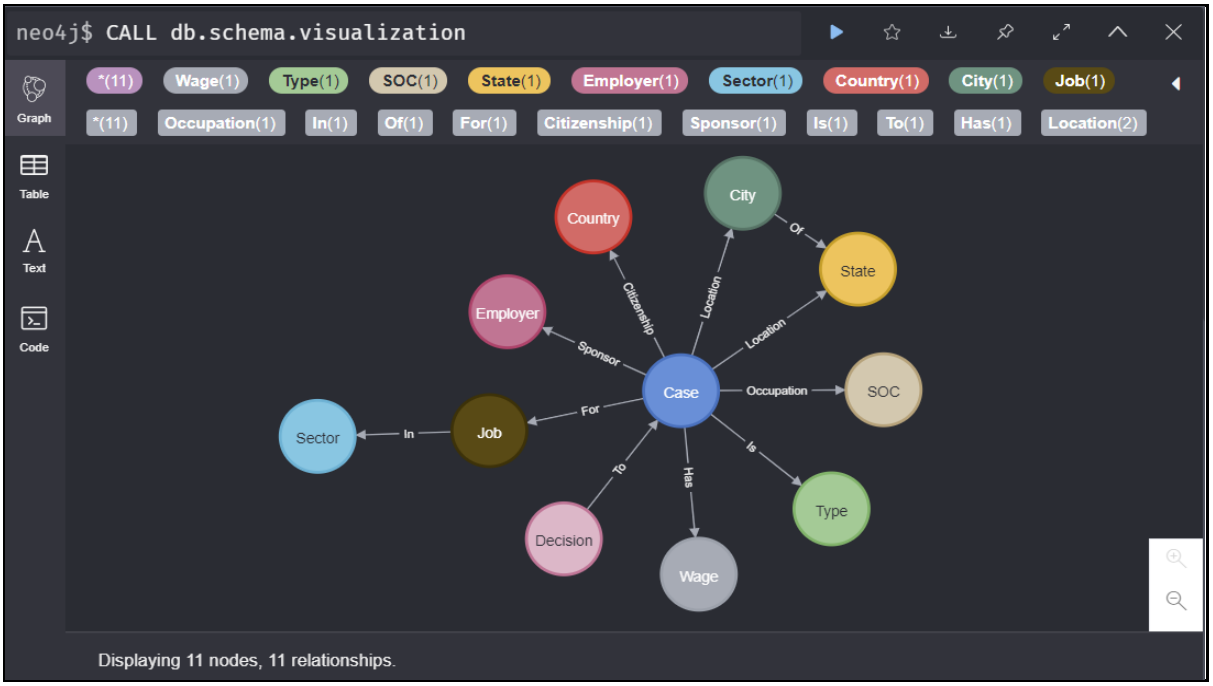
#Loading Wage Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (case:Case {case_no: line.casenumber})
MERGE (wage:Wage {range: toUpper(line.Salary_range) })
ON CREATE SET wage.range=toUpper(line.Salary_range),
wage.level=toUpper(line.pw_level_9089)
ON MATCH SET wage.level=toUpper(line.pw_level_9089)

CREATE (case)-[:Has]->(wage)
```

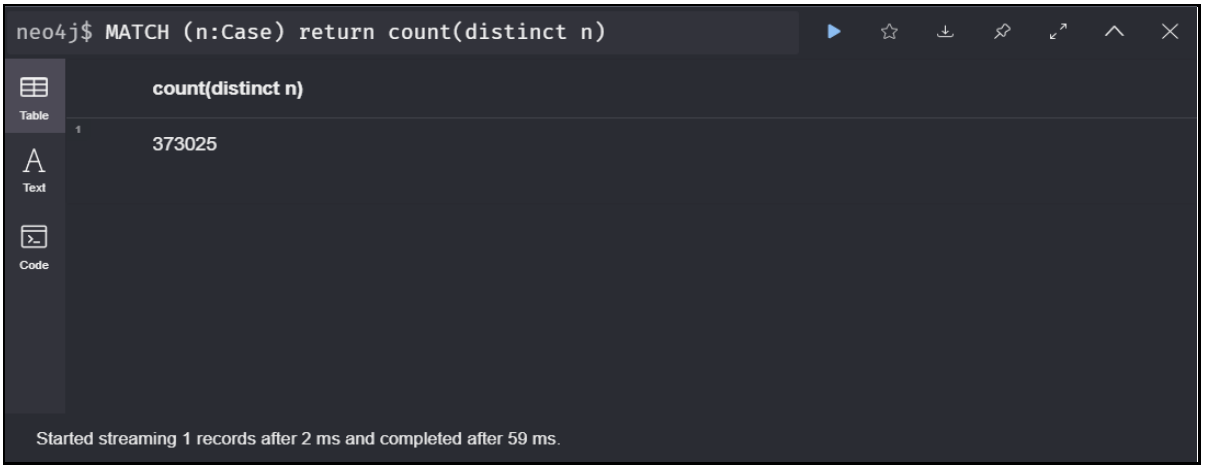
#Loading Sector Nodes

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///US_Permanent_Visas.csv' AS line
WITH line

MATCH (job:Job {title: toUpper(line.job_info_job_title) })
MERGE (sector:Sector {name: toUpper(line.us_economic_sector) })

CREATE (job)-[:In]->(sector)
```

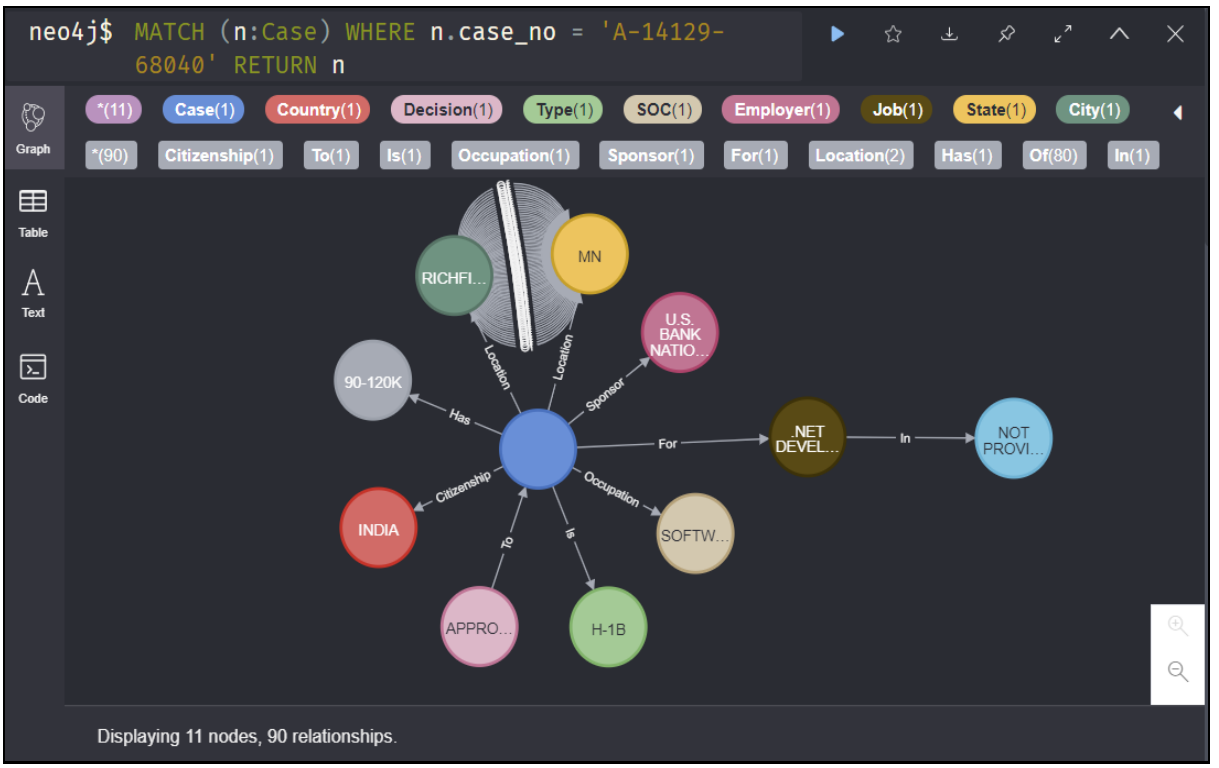
Database Schema in Neo4j



Distinct Nodes created which are equal to the number of unique records in our dataset.



Example Case Number: A-14129-68040



Connecting to Neo4j for Data Visualization

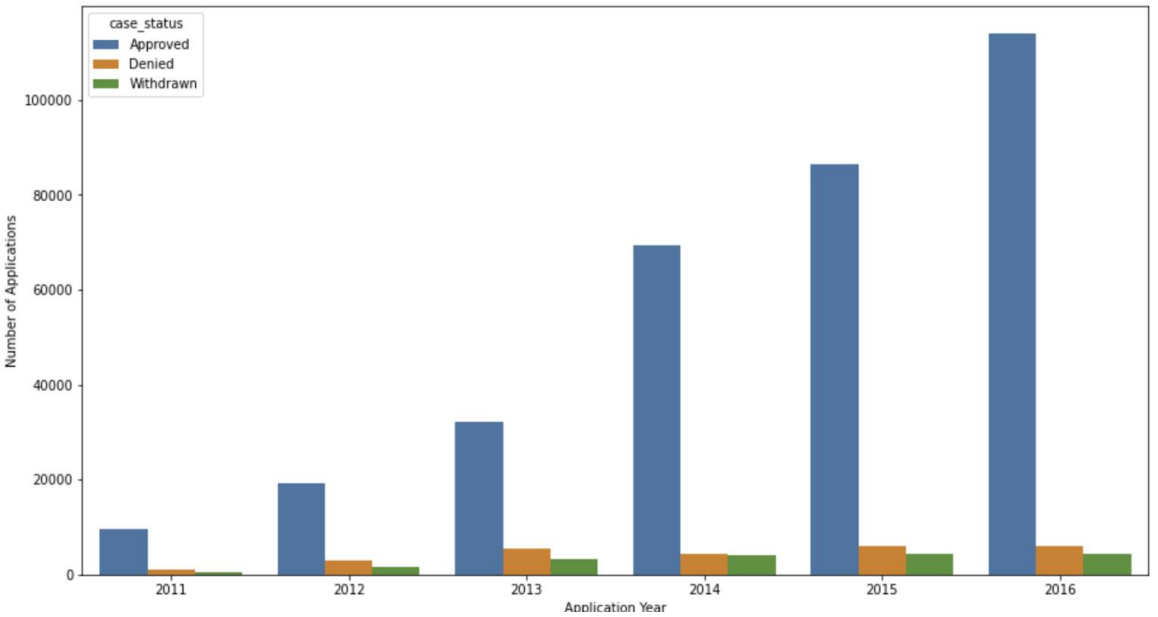
```
from neo4j import GraphDatabase
class Neo4jConnection:

    def __init__(self, uri, user, pwd):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__driver = None
        try:
            self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
        except Exception as e:
            print("Failed to create the driver:", e)

    def close(self):
        if self.__driver is not None:
            self.__driver.close()

    def query(self, query, db=None):
        assert self.__driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.__driver.session(database=db) if db is not None else self.__driver.session()
            response = list(session.run(query))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response
```

```
conn = Neo4jConnection(uri="bolt://localhost:7687", user="neo4j", pwd="groupproject")
```

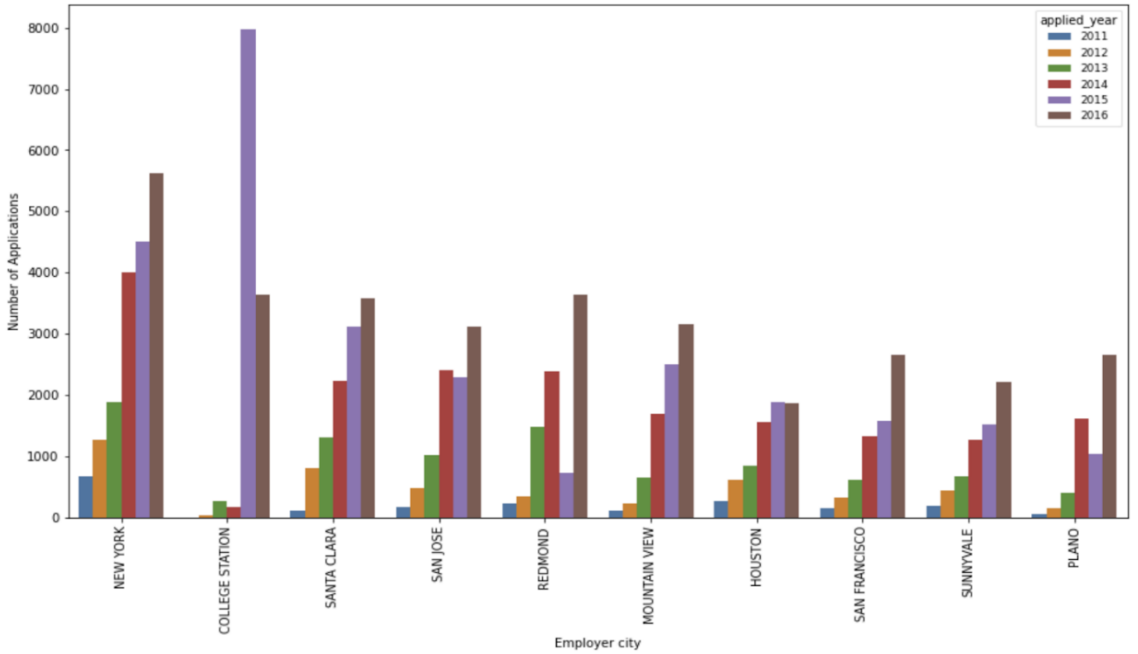
3.5. Data Validation and Data Visualization

Bar graph to analyze, understand and validate the data

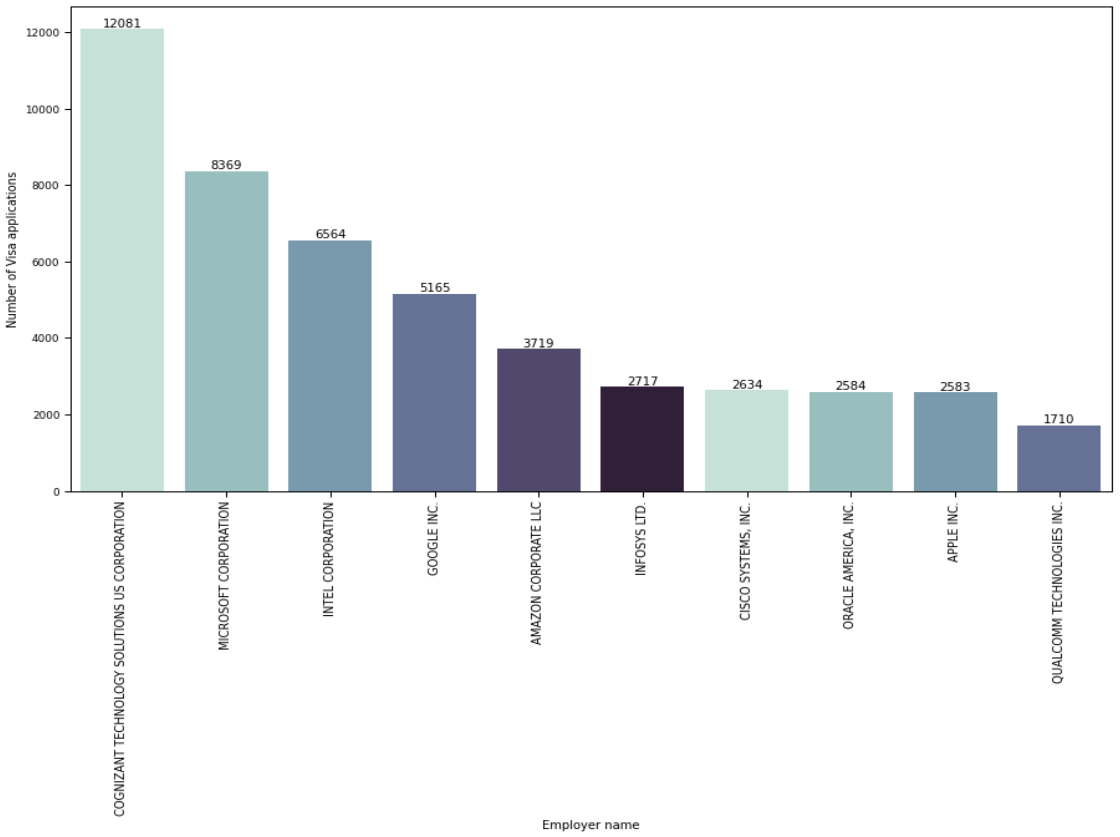
Plot to analyze number of applications every year



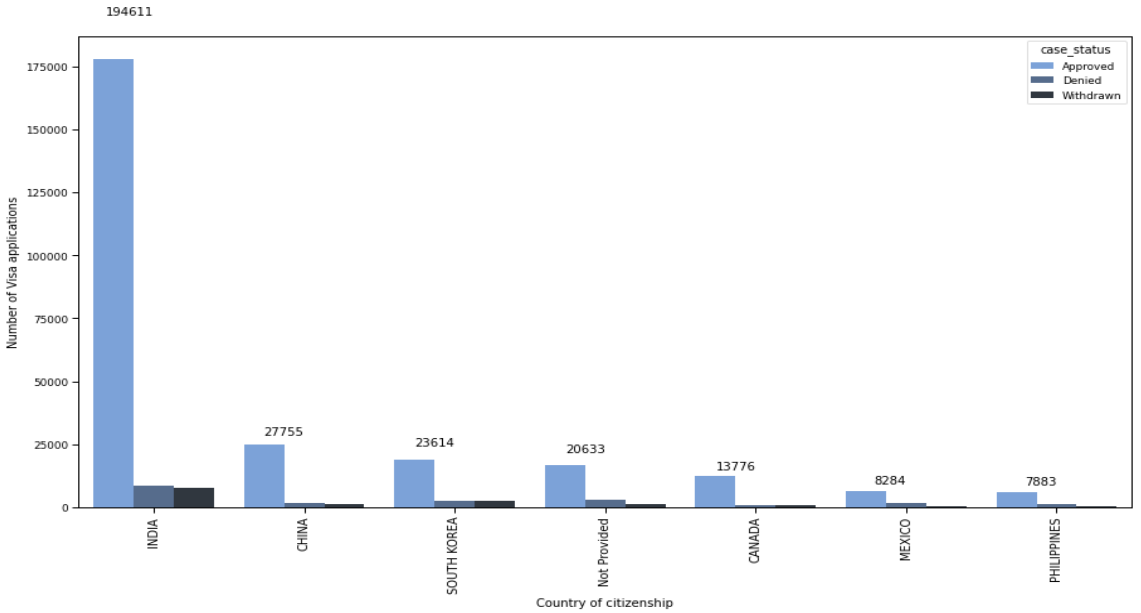
Plot to identify city with the greatest number of visa applications



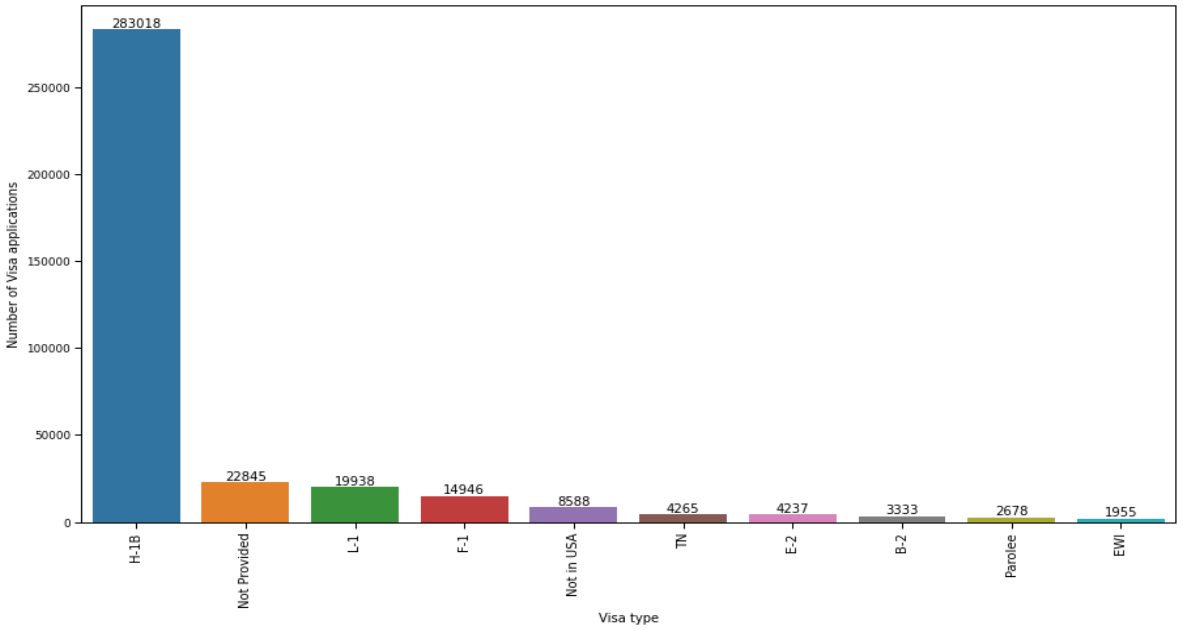
Plot to identify employers with the greatest number of visa applications



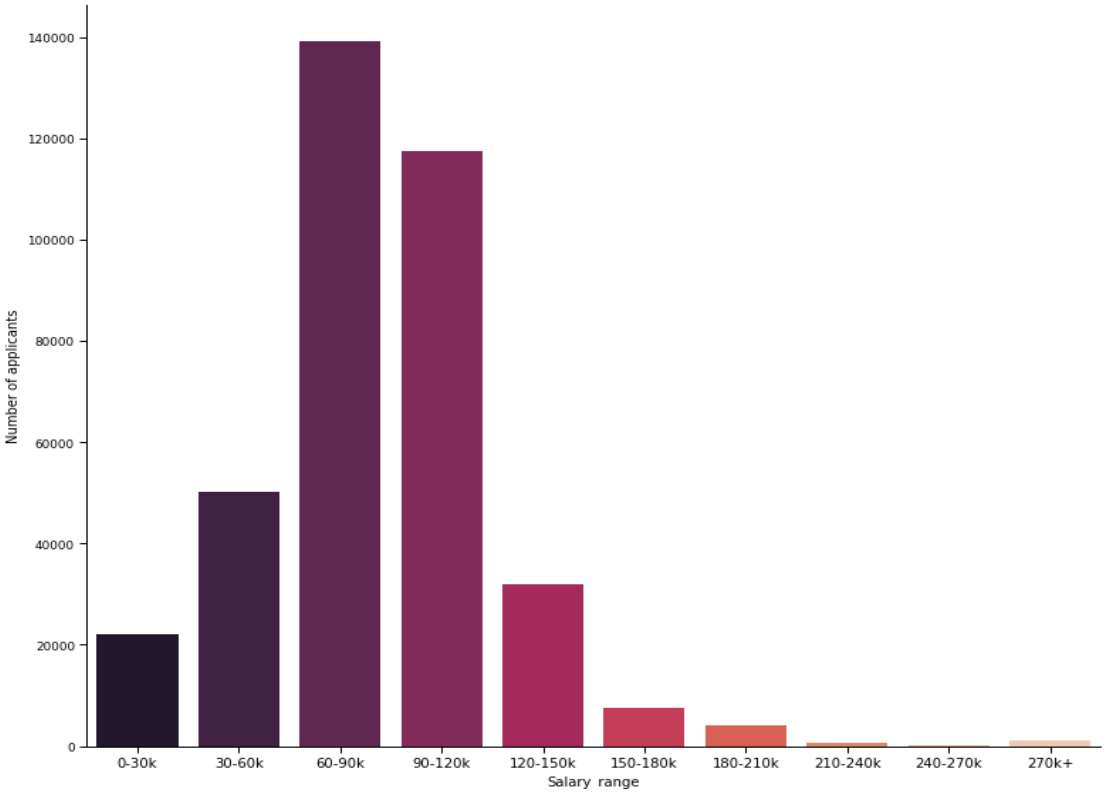
Plot for number of applications with respect to country of citizenship



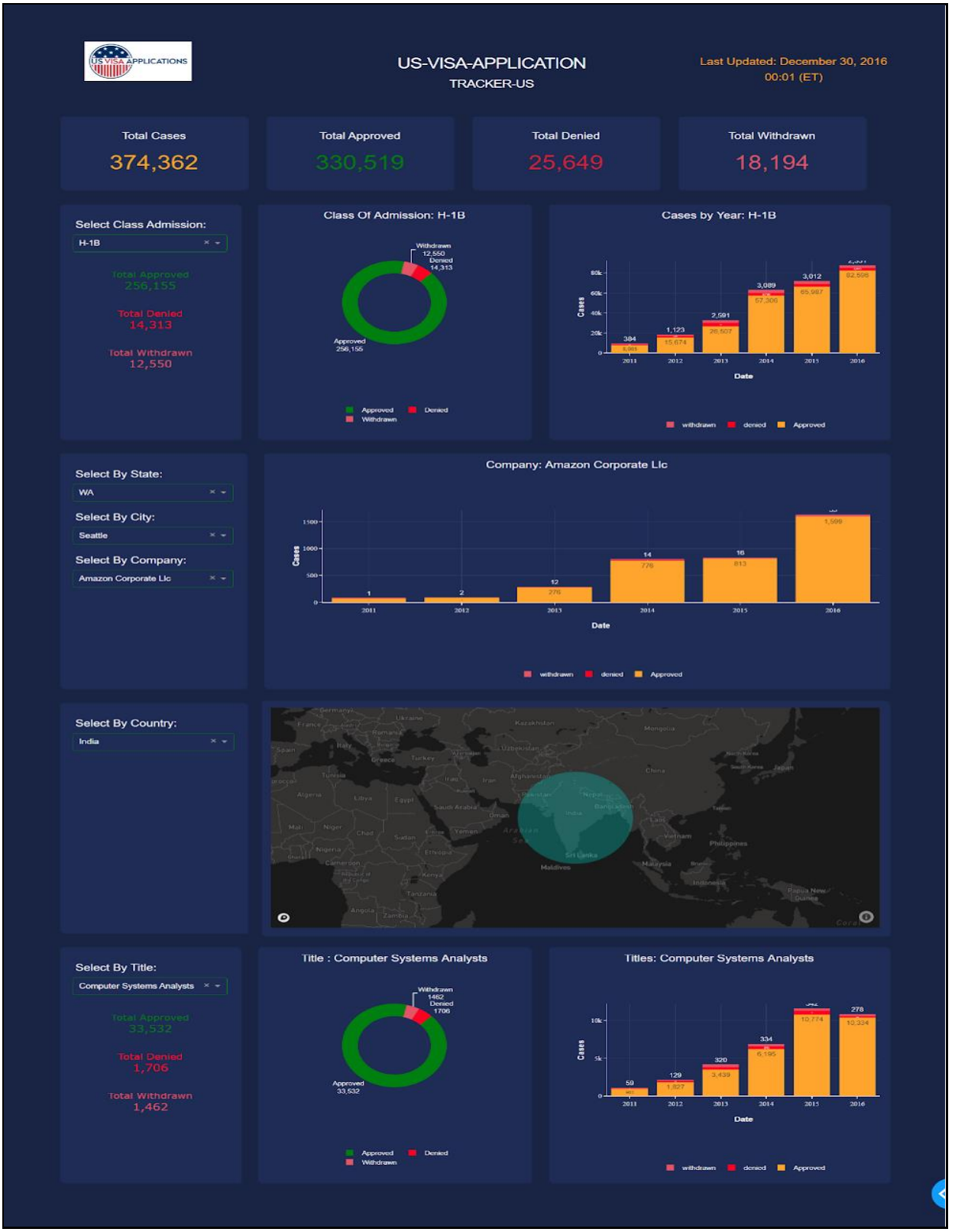
Plot for number of applications with respect to Visa Type



Plot for number of applications with respect to the Salary Range



Dashboard for US Visa Application Tracking



Target Audience

US Citizenship and Immigration Services

- Total number of Approved, Denied, and Withdrawn cases in the US.
- They can benefit from cases segregated based on visa admission, countries, different companies of the US located in different states.
- Understand the data of visa cases in each frequency.

Corporates of different sectors

- Corporates can use this data at the time of visa application filling of their employees
- The dashboard will give them an edge for filling visa requests as they can compare their visa request data with another company's data.
- Corporates and diversify their employees using the dashboard insights

Immigrants applying for US Visa

- Immigrants can look insights according to their class of admission or visa application that they are applying for
- Different year insights of approve, deny, or withdraw data will help people to understand their chances of getting their visa application accepted

Benefits of using the US Visa Application tracker dashboard

- Drill down segregation of approved, denied, and withdrawn visa applications using State, City and then company for better insights.
- Understanding of which class of admission visa application is approved most and got rejected most
- The observation of with kind of companies are majorly getting visa application approved and applied
- Country wise Visa application tracker for better understanding of visa application status
- It helps to fill the critical gaps among the people who are applying for visa application and their doubts will get cleared

- Quick exploration around the dashboard will give idea about feature analysis of visa application
- Filter based on country, class of admission and employee job title will help better navigate through the dashboard

Steps to run the code for dashboard

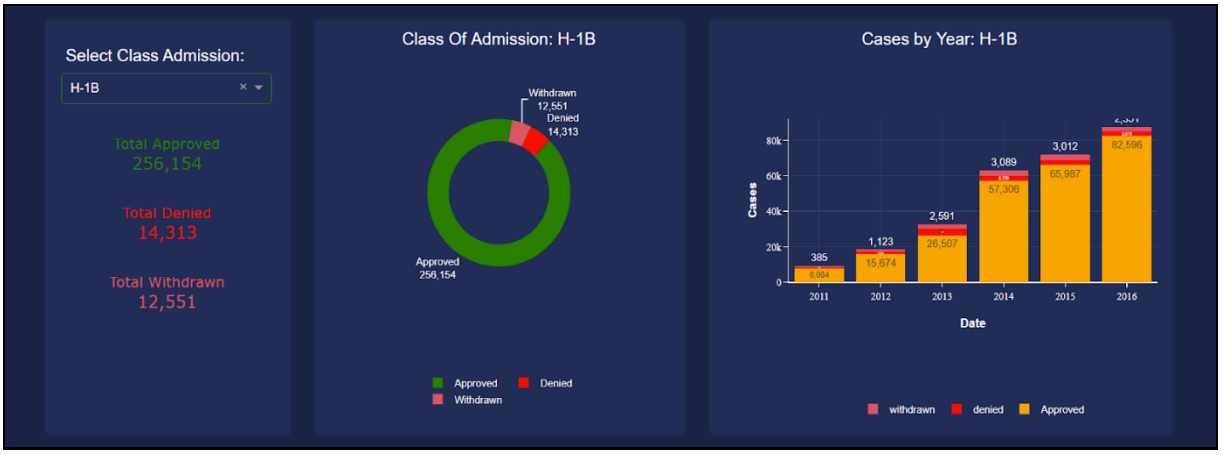
Dashboard is built on python Dash. Dash is a user interface library for creating analytical web applications.

To run the dashboard, use python, style and css file and run it on PyCharm or any python code editor. Once the code runs, we get a local IP address like **http://127.0.0.1:8050/** copy this link and run it in the chrome.

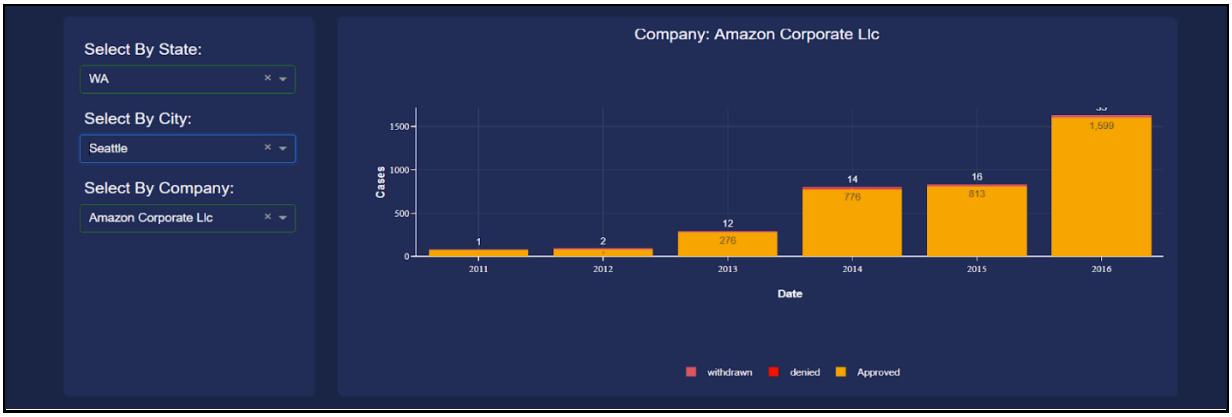
Dashboard Insights

Total Cases of approved, denied and approved also with last date of file to get updated.

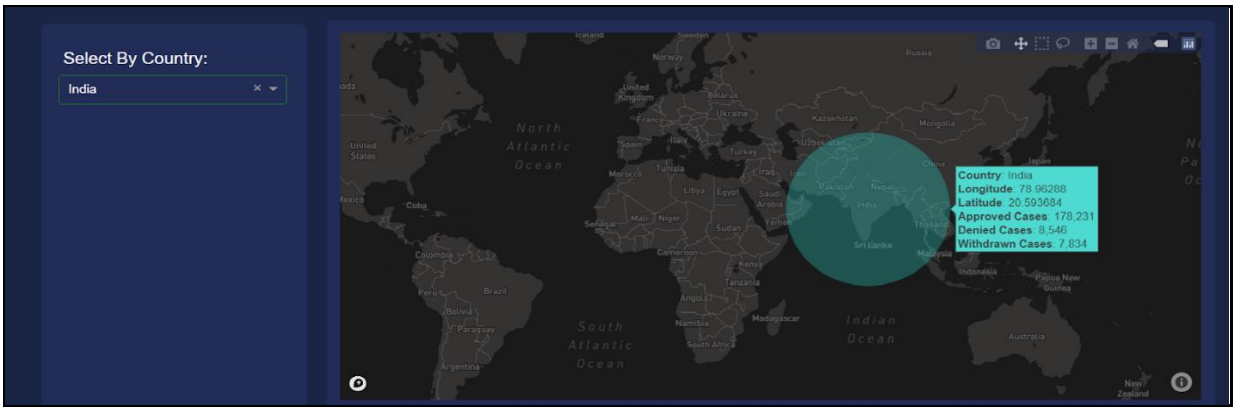
Most number of applications to get approved based on Class of Admission. We found that H-1B is the top visa application that is applied through the different companies and most approved visas.



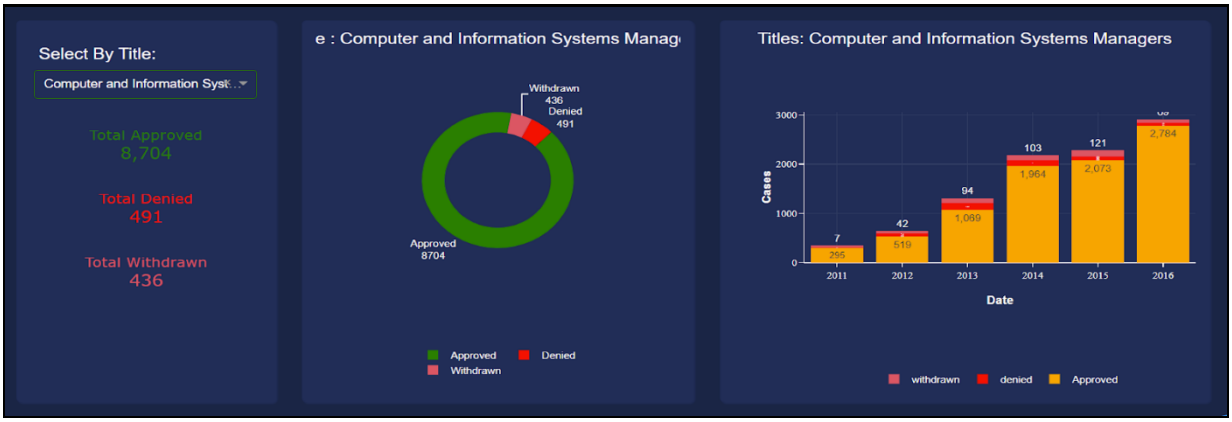
Amazon is amongst top 5 companies that file the highest number of visa applications and gets approved and after drilling down through state and city we found that Amazon campus at Seattle files the most of visa application and it is in increasing fashion.



India is the country with the greatest number of visa applications filed throughout the world and has the most approved cases. The cases appears when we hover over the country.



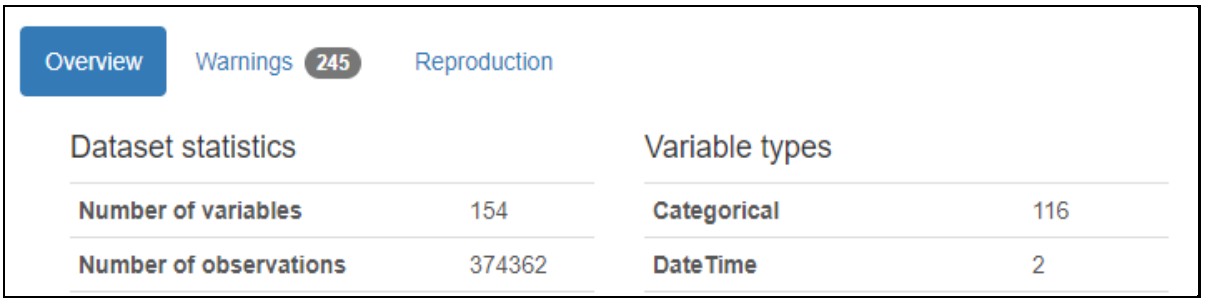
Computer Engineering is the hottest job for which companies are filling visa applications and has highest rate of approval.



3.6. System Integration and User Acceptance Testing

Validating the number of records after every process

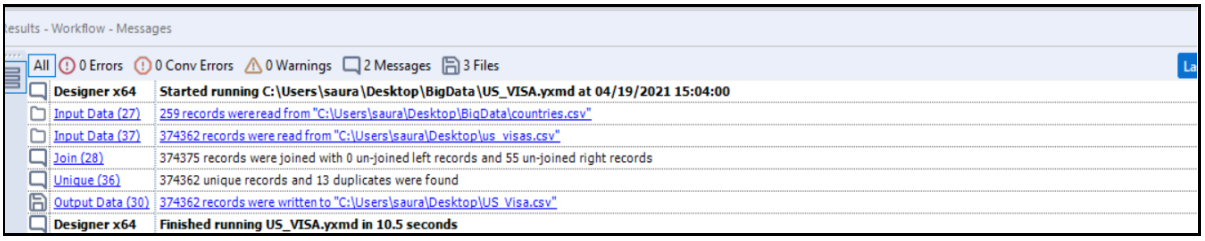
Number of records using Pandas Profiling



Number of records after preprocessing using Python libraries

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374362 entries, 0 to 374361
```

Number of records after processing the data using Alteryx



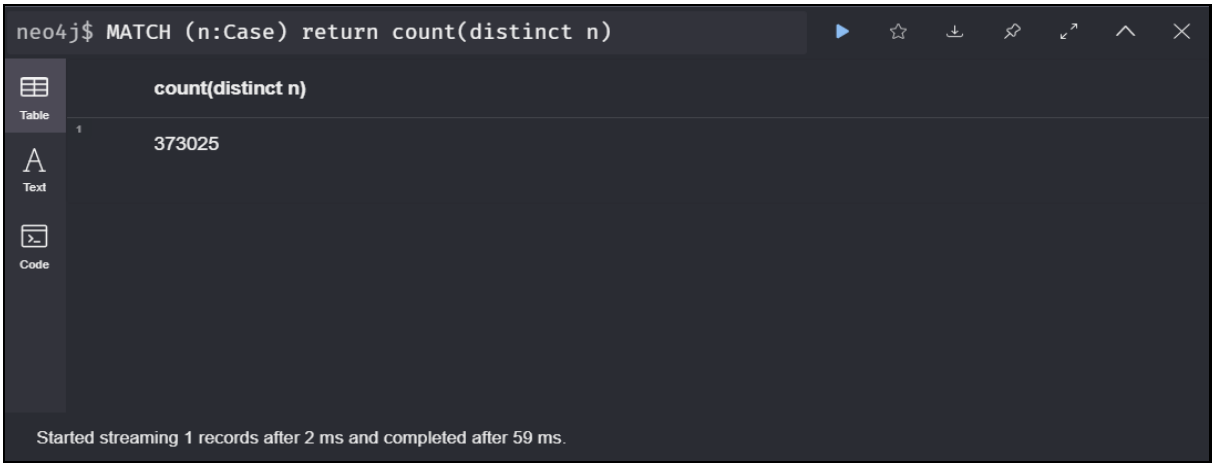
There is no loss of records after the preprocessing of data

Number of distinct case number in the original dataset – **373025**

```
#Counting distinct values for every case_number
print(df_perm_visas.casenumber.value_counts())
```

```
A-14160-76066    3
A-16181-27423    2
A-16004-56774    2
A-16216-38907    2
A-13106-56405    2
                ..
A-13119-59830    1
A-15068-57092    1
A-13311-12597    1
A-16113-00899    1
A-16127-06571    1
Name: casenumber, Length: 373025, dtype: int64
```

Number of distinct nodes created for every case



Validating data in the dashboard with respect to the original dataset for **User Acceptance Testing**

Data in original dataset

Total Cases: **374362**

Total Approved Cases: **330519**

Total Denied Cases: **25649**

Total Withdrawn Cases: **18194**

Data shown in the dashboard

3.7. Challenges Encountered

Lack of expertise in Neo4j and understanding cypher queries

Business is at risk if the lack of expertise brings about inefficiency and loss of operation and production of product.

The seriousness of the failure and its associated effects may defend on the causes of having expertise that brings about inefficiency and loss of operation and production.

Solution

Learning to use the database through documentation and online courses

Practice on the default Movies database for better understanding

Dashboard reporting needs meaningful metrics

First and foremost, a dashboard must measure something meaningful. This requires an understanding of the right metrics to select.

Solution

Broad metrics such as Total Cases will not provide much insight, but drilled-down, specific metrics that influence those broader metrics will.

Compatibility and interfacing issues

Data is not universal. Connectivity and compatibility are a common challenge among business dashboards. If a dashboard cannot connect with a critical business system, then the information it provides will be incorrect, outdated, or limited in its usefulness.

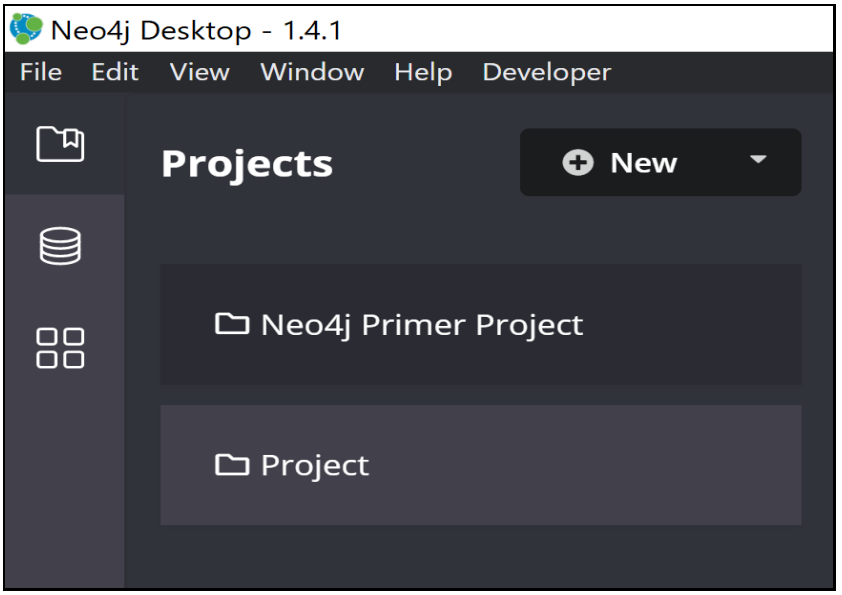
Solution

To bridge the gap, users must manually input data, a process that essentially defeats the purpose of the dashboard.

3.8. End User Instructions

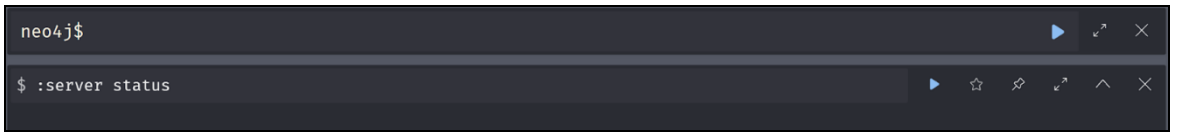
Steps for Database Creation and load

1) First, we create a Project from the Neo4j Desktop application.

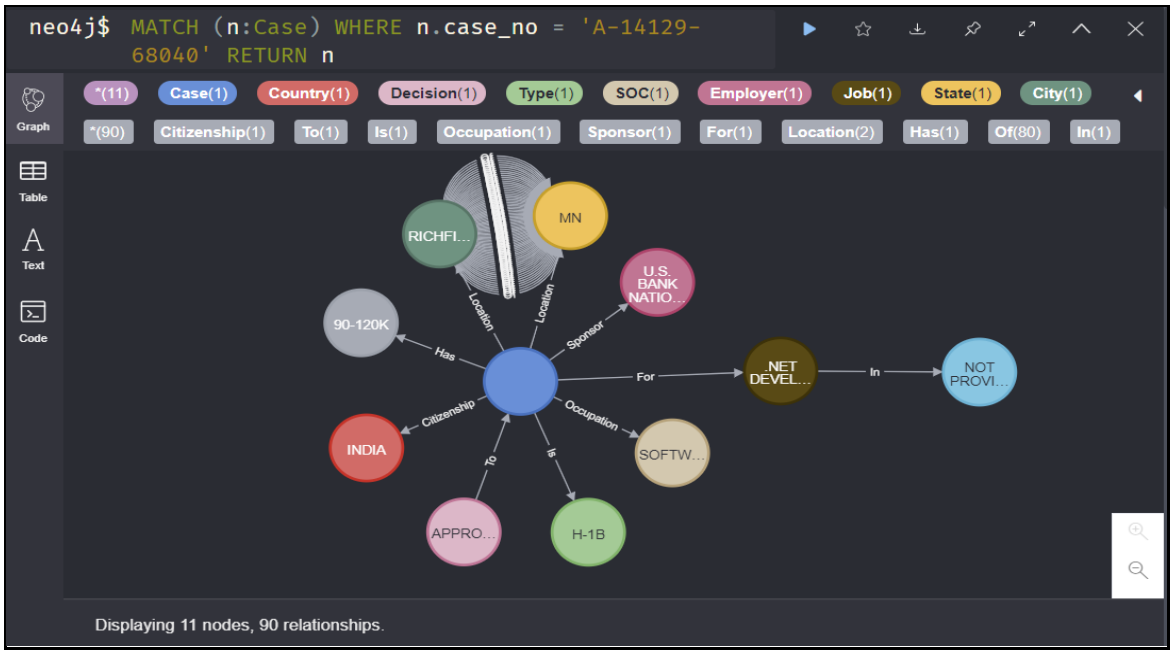


2) Then, we create a local DBMS for our project.

3) Once the DBMS is ready, we can start it by clicking on the start button and then open the server using the open button.
4) After opening the terminal, we can start creating the schema and loading the data into our Neo4j database from the query terminal.



5) For loading the nodes, we need to create all the constraints, it can be created using the code provided in the document.
6) Once we create the constraints for the database, we can start loading the nodes which are unique throughout our dataset i.e., for our dataset **Case**. Load all the nodes for Case along with the Country node and its relationship from the CSV.
7) Then using the MATCH function, we can load the remaining nodes and their relationship from the CSV.
8) Finally, when all the nodes are created with proper relationship, we can start querying, analyzing, and understanding the data using Graph interface.



Steps for running the dashboard

1) Run python file it through PyCharm or any python code editor.
2) Once the code runs, we get a local IP address **http://127.0.0.1:8050/**
3) Run it in the chrome.

Understanding the dashboard

1) For total cases, total approved, total denied and total withdrawn for a given dataset, look on the first container of dashboard.
2) To check cases for the class of admission, select class of admission from drop down list and then look at the percentage distribution at pie chart and again to check the cases distribution by year look for the bar chart.
3) To find the country that files the highest amount of visa application you can look at the country map.
4) Drill down function can be used to get detailed information of cases filled by different companies.