ASSIGNMENT 2

Understanding the Data Set-

The dataset used in the report is a real-world financial domain dataset collected in banks of 77 districts spread across 8 regions of Czech Republic. It consists 8 tables providing information about the bank's clients, their accounts, transactions made from and to those accounts, permanent orders depicting characteristics of a payment, granted loans, credit cards issued, and information about the district in which those accounts were opened.

The dataset includes 4500 accounts manipulated by 5369 clients. A disposition relation depicts whether a client is an Owner or a Disponent of the account being manipulated. One client can have more than one account, also one account can be handled by more than one client. This indicates a M: N Relationship between client and account which is shown in disposition.asc file.

There are some additional services provided by the bank which are shown in files named loan.asc and card.asc. The loan and account relations show a 1:1 relation because at most one loan can be granted to an account. The relation 'demographic data' which is provided in the file named as district.asc gives insight about the districts in which accounts are opened. Information like district name, district code, population of district, average salary of the district etc, is given in that relation.

Overview of the Report-

This report is designed on the foundation of Graph Data Modelling. We are forming a graph out of the given 8 relations to find meaningful insights from the dataset given. Also, we have devised an experiment to see the performance of different graphical representations. The experiment is carried out on a bipartite graph using disposition table for algorithm analysis. From this experiment we'll know which representation is best for our main graph, and then we'll use that representation for further data analysis.

After the cost analysis we'll use graph modelling concepts to compute the Number of Loans per region and see which region's accounts have been issued most loans in the given time period. After knowing the number of loans per region, we'll use the temporal modelling concept learnt in previous units to see the average amount of loan issued every year. Also using the same temporal concept we'll observe a pattern in issuance of credit cards every year. After that using the main graph and some relational modelling on card relation, we'll find out the top 10 most active accounts of our dataset.

We are using Python and Tableau as the major tools for the report. Modules of python used here are Pandas, NetworkX, Matplotlib, Time.

NetworkX is a python package used for creation, manipulation and study of network graphs. Matplotlib is a python library used for creation of interactive visualizations.

Graph Cost Modelling-

We know that the three major representations of a Graph are edge list, adjacency list, and adjacency matrix. We'll represent a bipartite graph in each rendition and perform degree distribution calculations on that representation to see which of them takes least time.

We'll divide the relation in the steps of 100 rows and keep on creating subgraphs until the full relation is used to form graph. These subgraphs will be used to calculate the degree distribution and then we'll plot these degree distributions on a line chart to see the effect of increasing the number of nodes on degree distribution. While this algorithm runs, we'll keep track of time and see which representation is most efficient for us to use further.

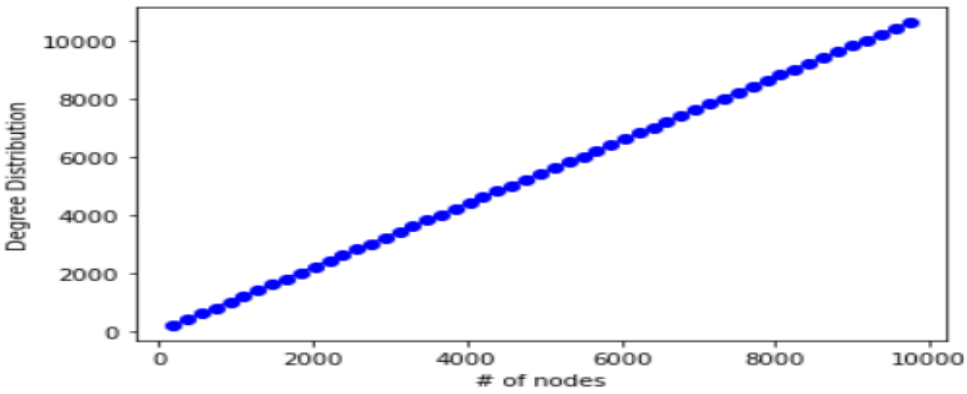
Edge-list:

First is the edge list representation of a graph. In this we created the list of all edges of graph and then calculated degree distribution using that.

Below is the algorithm used for edge-list of creations of subgraph and then degree distribution calculations of all subgraphs.

```
import pandas as pd
import matplotlib.pyplot as plt
from networkx import nx
import time

start = time.time()

def degree_dist(G):
    degrees = [G.degree(n) for n in G.nodes()]
    DD_edgeList.append(sum(degrees))

DD_edgeList = []
df_disp= pd.read_table('disp.asc', delimiter=';')
for i in df_disp['client_id']:
    df_disp['client_id'].replace(i,f"{i}.c", inplace = True)

for i in df_disp['account_id']:
    df_disp['account_id'].replace(i,f"{i}.a", inplace = True)

NoOfNodes = []
for i in range(91,df_disp['disp_id'].count()+1,91):
    df1 = df_disp.head(i)
    edge = list(zip(df1['account_id'], df1['client_id']))
    G = nx.Graph()
    G.add_nodes_from(df1['account_id'], bipartite=0)
    G.add_nodes_from(df1['client_id'], bipartite=1)
    G.add_edges_from(edge)
    degree_dist(G)
    NoOfNodes.append(G.number_of_nodes())

plt.plot(NoOfNodes, DD_edgeList, '-bo')
plt.ylabel('Degree Distribution')
plt.xlabel('# of nodes')

end = time.time()
print(f" Runtime of edgeList program = {end - start}")
```

From above algorithm we got below Plot of Degree Distribution vs the number of nodes.



Now, Time taken by above algorithm to execute is.

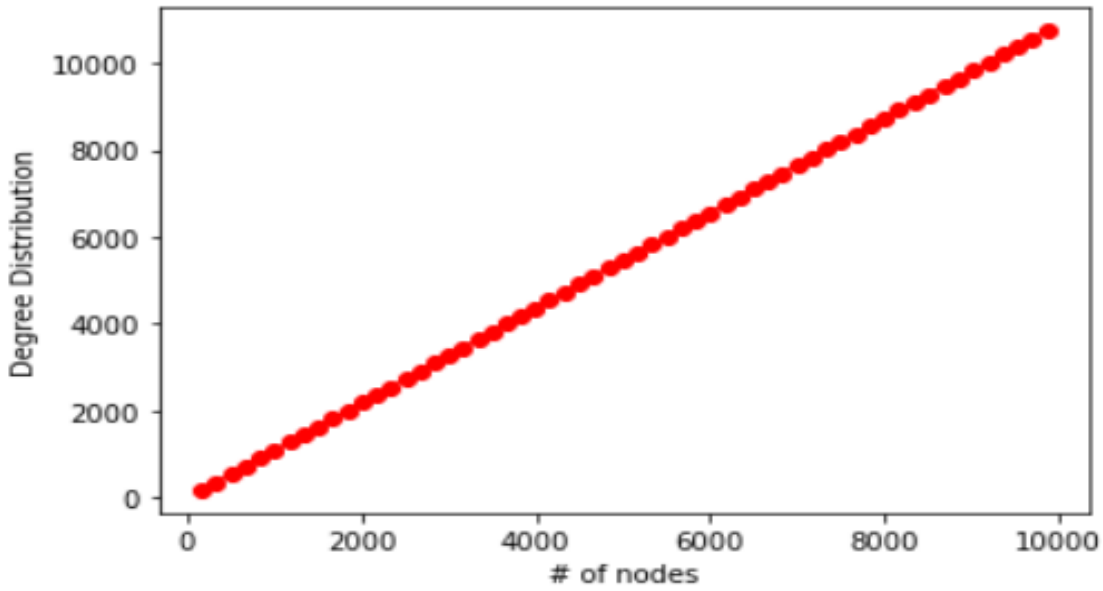
```
In [188]: runfile('C:/Users/risha/.spyder-py3/untitled0.py', wdir='C:/Users/risha/.spyder-py3')
 Runtime of edgeList program = 4.392711877822876
```

Adjacency-List:

Second representation of graph is the Adjacency-List. It is a collection of unordered lists used to represent a graph. Each list indicates the neighbours of that node of a graph. Below is the algorithm used for degree distribution calculation with the help of adjacency-list.

```
import pandas as pd
import matplotlib.pyplot as plt
from networkx import nx
import time

start = time.time()

def degree_dist(adjlist):
    li = []
    for n in G.nodes():
        li.append(len(adjlist[n]))
    DD_List.append(sum(li))

Adj_List = []
DD_List = []
df_disp= pd.read_table('disp.asc', delimiter=';')

for i in df_disp['client_id']:
    df_disp['client_id'].replace(i,f"{i}.c", inplace = True)

for i in df_disp['account_id']:
    df_disp['account_id'].replace(i,f"{i}.a", inplace = True)

NoOfNodes = []
for i in range(91,df_disp['disp_id'].count()+1,91):

    df1 = df_disp.head(i)
    G = nx.Graph()
    G.add_nodes_from(df1['account_id'], bipartite=0)
    G.add_nodes_from(df1['client_id'], bipartite=1)
    Adj_List = {key: [] for key in G.nodes()}
    for i in list(zip(df1['account_id'], df1['client_id'])):
        Adj_List[i[0]].append(i[1])
        Adj_List[i[1]].append(i[0])
    degree_dist(Adj_List)
    NoOfNodes.append(G.number_of_nodes())

plt.plot(NoOfNodes, DD_List, '-ro')
plt.ylabel('Degree Distribution')
plt.xlabel('# of nodes')

end = time.time()
print(f" Runtime of Adjacency-List program = {end - start}")
```

We have used above algorithm to find the degree distribution for multiple subgraphs and full graph of the relation with the help of Adjacency List representation and got below Plot of Degree distribution vs the number of nodes.



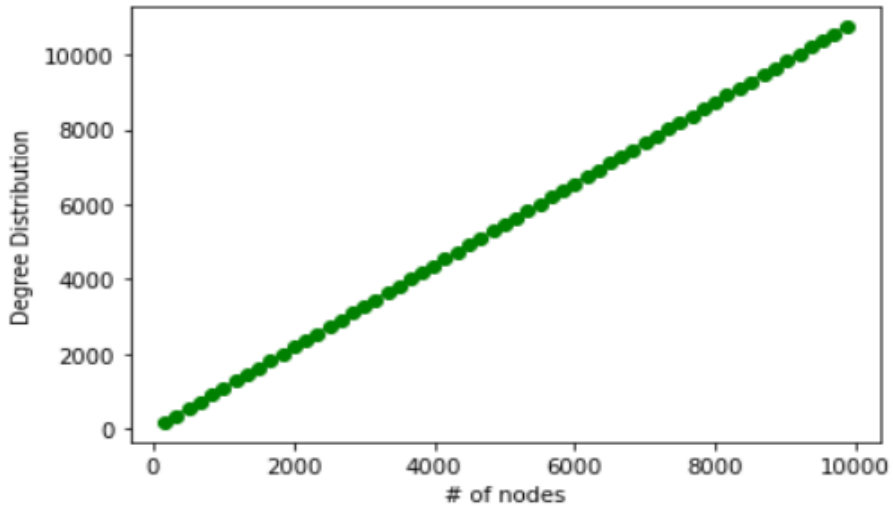
Now, Time taken by above algorithm to execute is.

```
In [4]: runfile('C:/Users/risha/.spyder-py3/Adj_list_cost_analysis.py', wdir='C:/Users/risha/.spyder-
py3')
 Runtime of Adjacency-List program = 4.564667224884033
```

Adjacency Matrix:

Third representation is Adjacency Matrix. It's a square |V| x |V| Boolean matrix indicating which edges exist in the graph. 1 represents that the nodes are adjacent to each other, 0 represents otherwise.

Below is the plot of Degree Distribution vs Number of Nodes obtained from the algorithm using Adjacency Matrix.



Below is the algorithm used.

```
import pandas as pd
import matplotlib.pyplot as plt
from networkx import nx
import time
start = time.time()
def degree_dist(adjmat):
    li = []
    for n in G.nodes():
        count = 0
        for m in G.nodes():
            if adjmat[m][n] == 1:
                count= count + 1
        li.append(count)
    DD_List.append(sum(li))

DD_List = []
df_disp= pd.read_table('disp.asc', delimiter=';')
for i in df_disp['client_id']:
    df_disp['client_id'].replace(i,f"{i}.c", inplace = True)

for i in df_disp['account_id']:
    df_disp['account_id'].replace(i,f"{i}.a", inplace = True)

NoOfNodes = []

for i in range(91,df_disp['disp_id'].count()+1,91):
    df1 = df_disp.head(i)
    G = nx.Graph()
    G.add_nodes_from(df1['account_id'], bipartite=0)
    G.add_nodes_from(df1['client_id'], bipartite=1)
    Adj_Mat= pd.DataFrame(index = G.nodes(), columns = G.nodes())
    Adj_Mat.fillna(0, inplace = True)
    for i in list(zip(df1['account_id'], df1['client_id'])):
          Adj_Mat[i[0]][i[1]] = 1
    degree_dist(Adj_Mat)
    NoOfNodes.append(G.number_of_nodes())

plt.plot(NoOfNodes, DD_List, '-go')
plt.ylabel('Degree Distribution')
plt.xlabel('# of nodes')

end = time.time()
print(f" Runtime of Adjacency-Matrix program = {end - start}")
```

Now, time taken by above algorithm to execute.

```
In [47]: runfile('C:/Users/risha/.spyder-py3/untitled2.py', wdir='C:/Users/risha/.spyder-py3')
 Runtime of Adjacency-Matrix program = 68638.362538576126
```

Summarizing Experimental Analysis:

We can clearly see from above snapshots that 'Adjacency Matrix' representation of graph is the worst of all the three as it's using maximum time to get executed (**68638.36353 units**) which is an impractical time for a simple degree distribution calculation algorithm to take.

Also, we observed that,

Time taken by 'Adjacency List' algorithm to execute = 4.56466 units

Time taken by 'Edge List' algorithm to execute = 4.39271 units

These two representations have a similar wall time of execution which shows the efficiency of these two representations over the 2D matrix representation. The minute difference in the time of Adjacency list and Edge list might be due to the latency of system.

Hence, we'll use Edge List or an Adjacency List representation in our report to find answers for our questions.

Theoretical Cost Analysis of Degree Distribution Algorithms:

As we have learned in class the different costs of calculating Degree Distribution for different representations, Let's see how that shapes out for our bipartite graph between accounts and clients.

Total No. of Nodes in the Graph |V| = Distinct Clients in disp.asc + Distinct Accounts in disp.asc

= 5369 + 4500

|V| = 9869

Total No. of Edges in the Graph |E| = 2 * Total # of rows in disp.asc

= 2 * 5369

|E| = 10738

For Adjacency Matrix, the cost of calculating Degree Distribution = $O(|V|^2)$

= 9869 * 9869

Cost of Adjacency Matrix = 97,397,161

For Adjacency List, the cost of calculating Degree Distribution = O (|V| + |E|)

= 9869 + 10738

Cost of Adjacency List = 20,607

For Edge List, the cost of calculating Degree Distribution = O (|V| + |E|)

= 9869 + 10738

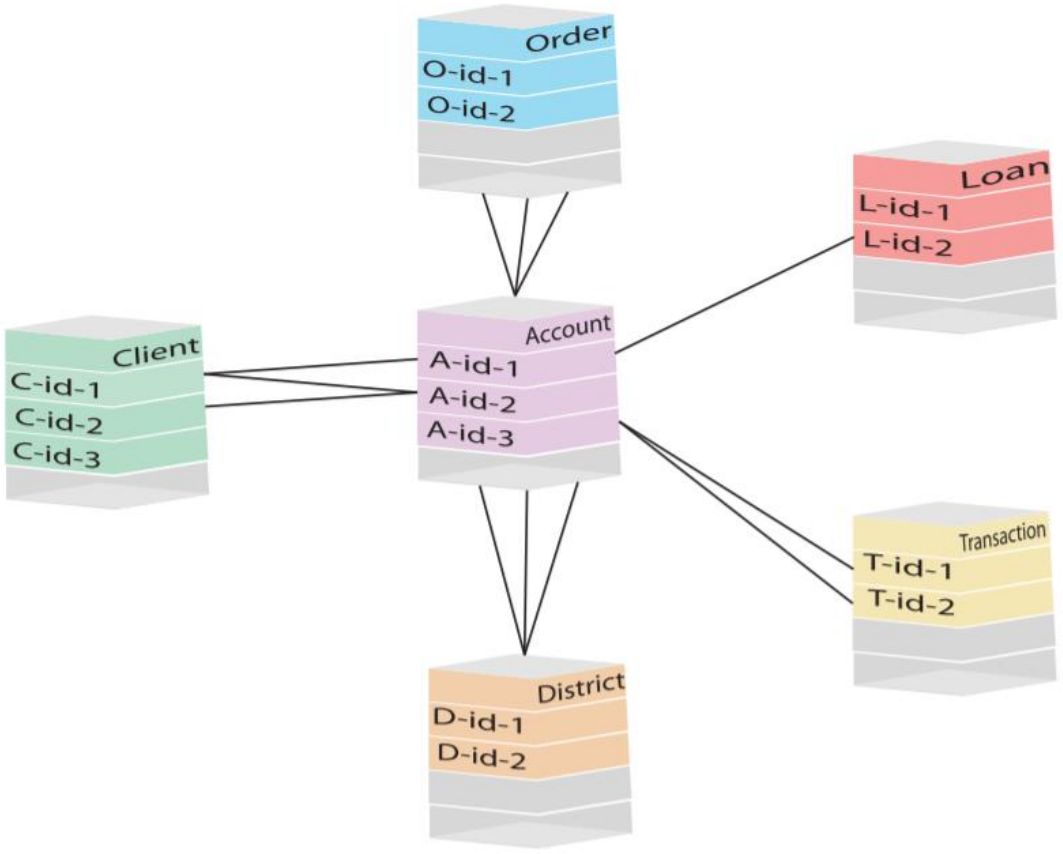
Cost of Edge List = 20,607

We can depict from above analysis that cost of an algorithm of calculating degree distribution for Adjacency List and Edge List is same and very less from the Cost of Adjacency Matrix. The difference between the cost of Lists and Matrix is uncanny. We can see that the list representations are almost 5000times more efficient than Matrix representation for our graph and algorithm. This is also very clearly indicated from the experimental algorithmic analysis done above with all three representations. Hence, we can conclude that Adjacency List and Edge List are much more efficient than Adjacency Matrix and therefore we will use them in further data analysis.

Graph Modelling integrated with Relational and Temporal concepts:

Now since we have established the efficient way to represent a graph for algorithm, we'll create a graph out of the relations Account, Client, Transactions, Loan, Order, and District.

Below is the graph we are using.



In the graph we have labelled nodes separately for each relation. For instance, Account relations, we have total 4500 nodes labelled as 'Account_id.A'. Similarly, for all the relations the identifiers are the labelled nodes of the graph. This will help us in extracting any row of a particular relation with precision, and with zero error.

From above graph we are clearly showcasing the relationship between the tables of the dataset.

Relation-1	Relation-2	Relationship
Account	**Client**	**M: N**
Account	**Order**	**1: N**
Account	**District**	**N: 1**
Account	**Transaction**	**1: N**
Account	**Loan**	**1: 1**

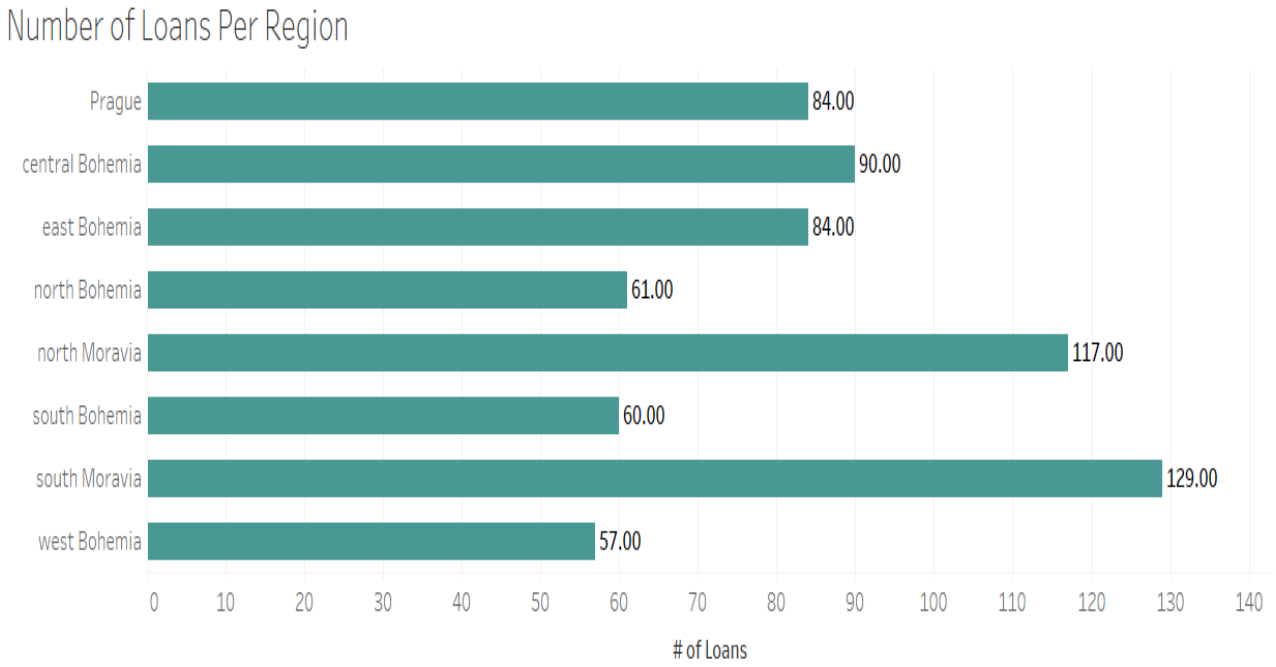
We have a path from a node in a relation to a node to every other relation via Account relation by traversing on the edges. Now using above graph let's find out Loans issued per region.

To find out the loans per region, we'll traverse the graph in the following way:

LOAN_ID ---------- > ACCOUNT_ID ---------- > DISTRICT_ID

We have created edge list representation of the graph. Now, to traverse from Loan_ID node to an Account_ID node, we just need to find the edges that does the same transition. After hopping once from Loan_ID to Account_ID to know which account has taken a loan, we'll perform another hop from Account_ID to District_ID to see which account with a loan belong to what district. Using this graph hopping technique we'll get ourselves the number of loans in a particular district. Now, applying relational modelling concepts to the district relation, we'll map the District_IDs to the 8 regions given in the relation. We'll create a new relation table which will indicate the number of loans in that particular region. Finally, using relational and graph modelling concepts we attain a relation table which tells us the number of loans per region.

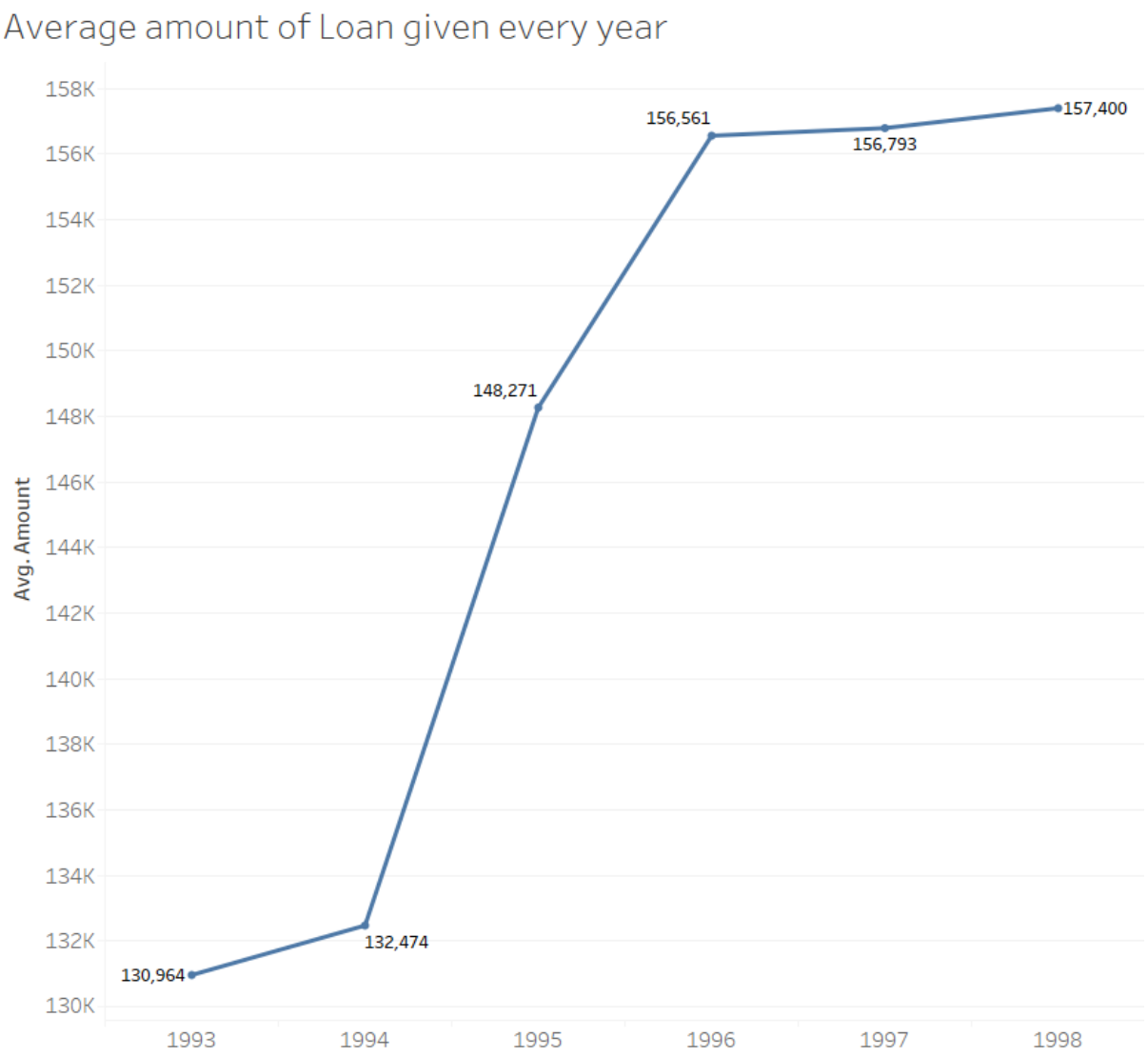
Below is the pictorial Bar chart representation of Number of Loans for all 8 regions within the given timeframe of around 5 years.



Now, after knowing about the loans in each region, we'll apply temporal modelling concepts to the relation we obtained from above operations to see the average amount of loan given each year.

In the obtained relation of Loan information, firstly we parsed the date into a relevant format for data visualization. Secondly, we collected all the loans issued in a particular year. Lastly, we calculated the average of all the loans issued in that year and saved the value into a pandas Data-Frame in order to visualize that in a line-chart using Tableau.

So, below is the Pictorial Line-Chart Representation of Average Amount of Loan issued every year in those 8 regions and 77 districts.

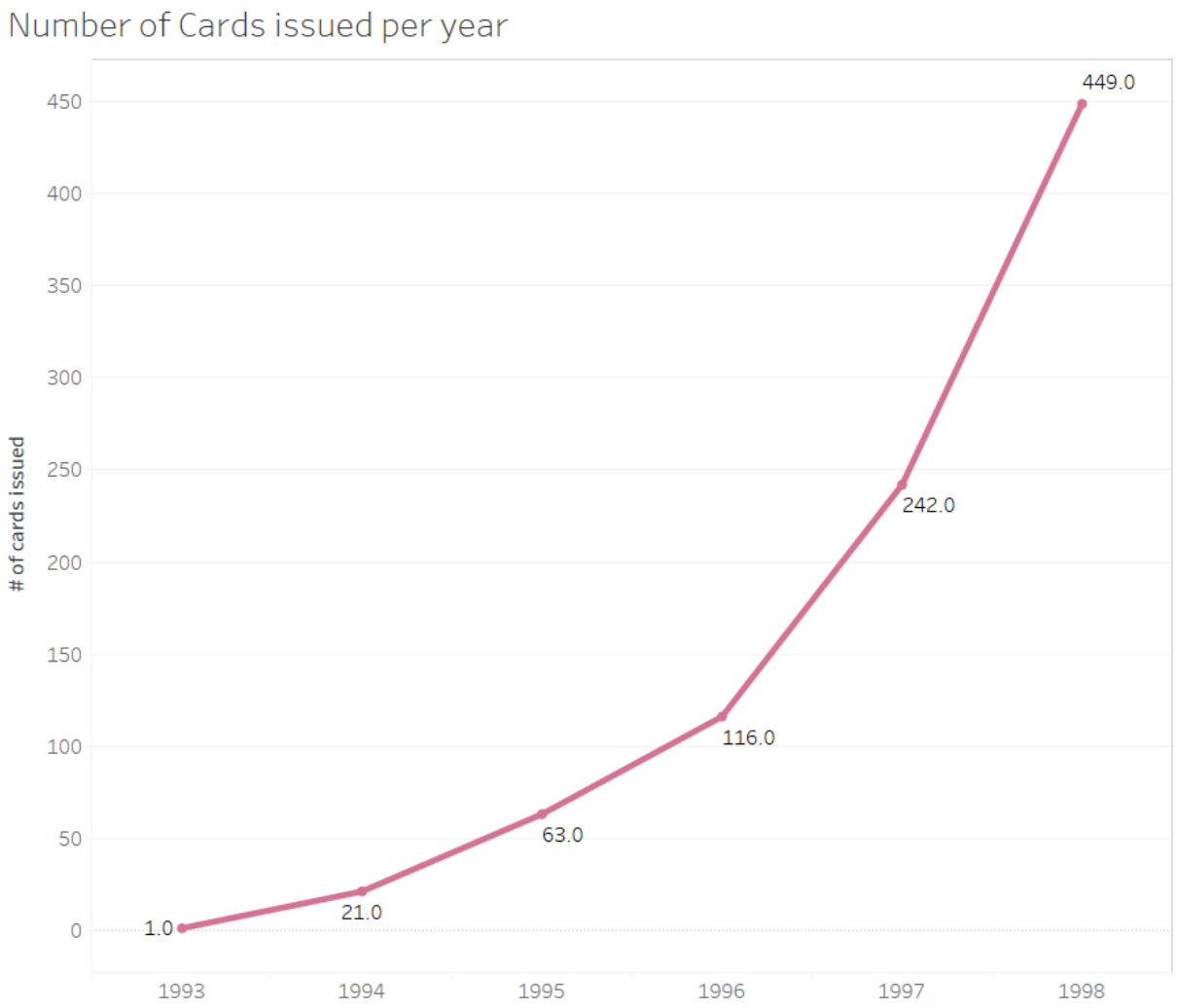


From above chart, we can see that the average loan amount increased every year. As with the growing modernization every year, the worth of things increased which tends to increase the need of necessary wealth for the people, hence the increasing trend line of the loan amount. Also, we can see a drastic increase in the amount from 1994 to 1996 as compared to the other time frames.

Now with the growing modernization, comes the introduction of credit cards. Many clients tend to incline from cash-based payment approach to a card-based transaction. As the world enters a digital age, people started realizing the benefits of carrying cards instead of carrying a bulk of cash with them. The benefits of cards over cash started luring people towards it and the clients starting approaching the bank for issuance of cards.

So, to support this theory, we used temporal modelling concepts on relation Card to see whether there's an increasing demand of credit cards in our 8 regions and 77 districts every year or these regions are deprived of the usage of credit cards.

Below is a Pictorial Line-Chart Representation of Number of Cards issued per year for the given timeframe.



As predicted, we can clearly observe the exponential increase in issuance of credit cards in the 8 regions. Also, the graph depicts that the first credit card in those 77 districts was issued in 1993 which marks the start of digitization of monetary transactions. It took almost 3 years for people in those districts to realize the benefits of the credit cards over cash and hence the number of cards issued in 1997 and 1998 saw a rapid surge as opposed to previous years.

Since, our whole graph is centred around 'Account' relation nodes, let's try to find out the **top 10 most active accounts** in our dataset.

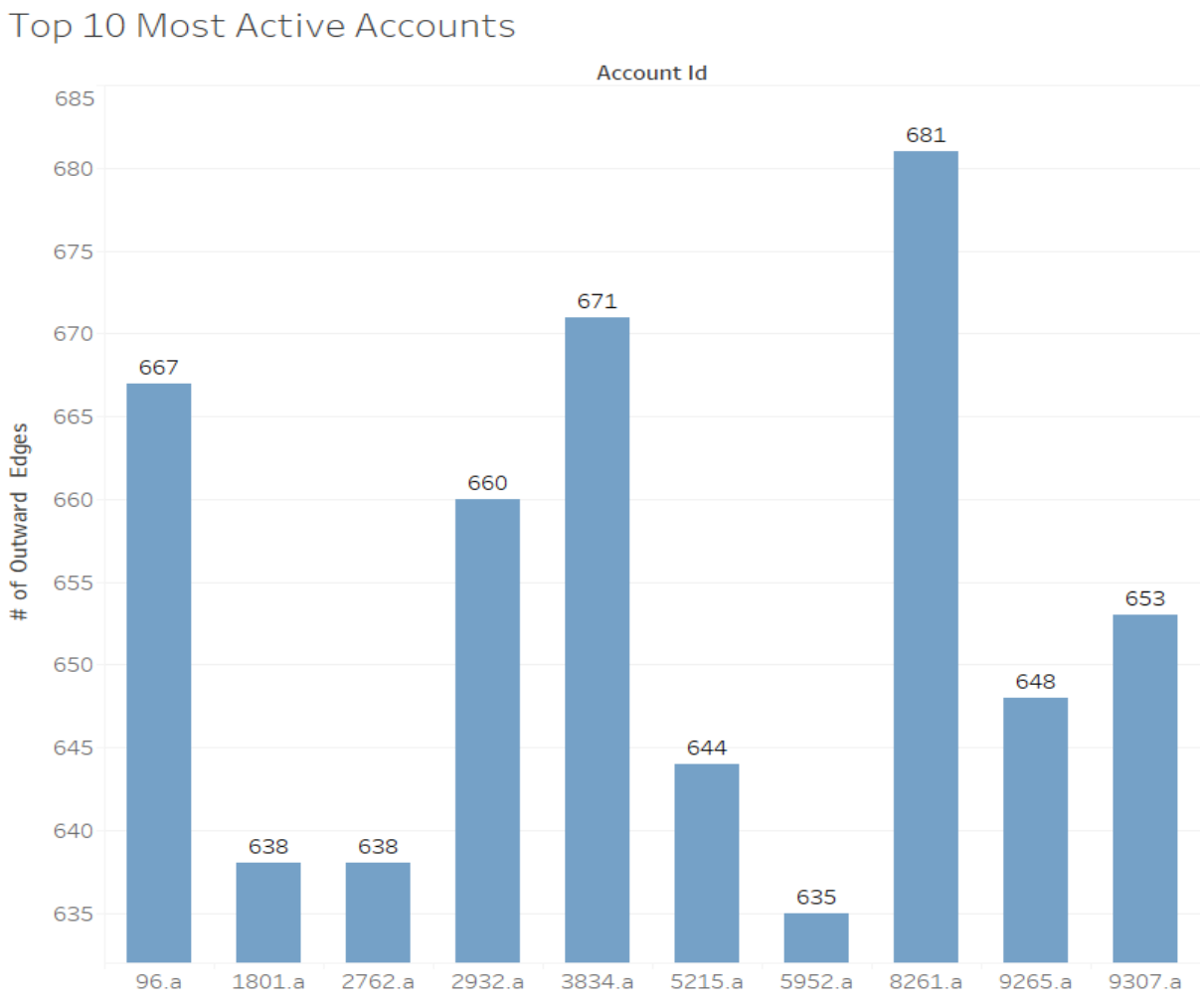
An account node will be considered most active if it has a large number of outward edges going towards Order relation, Transaction relation, Loan relation. It is obvious that an account will be active if there's a large number of transactions and payment orders being done from it. Hence, the condition of top 10 account nodes with the most outward edges will be considered as the most active accounts for a given timeline. We are not considering edges going from Account to Client or District relation because these just shows the static information of the account and we are analysing an account on the basis of its dynamic service utilizing characteristics.

Also, we have a relation called as Card which indicates the credit cards being issued to a specific client using a particular account. A card being issued under an account is also a sign of activeness of that account. But we have left this relation out of the graph so that we can integrate relational modelling concepts studied in previous units with the graph modelling concept to find the top 10 active accounts.

We created a Data-Frame which saved the number of edges of the Account nodes. Using graph concepts on the Edge-List, all edges were counted for every node and saved in the Data-Frame created above. All the edges incident from Transaction nodes, Order nodes, Loan nodes onto the account nodes were counted and segregated as per Account_ID. We used the degree distribution calculation algorithm to find degree of Account nodes and then subtracting the edges which were going towards Client and District relations which gave us accurate dynamic characteristic edge count for each node.

Now, to incorporate credit card issuance in this Data-Frame, we applied left join operation on Card relation and Disposition relation on the key 'disp_id'. This join operation gave us the Account_ID corresponding to each card issued. Now we added number of cards issued to each of the Account_ID to the Data-Frame of edges. After applying appropriate filters, we got the top 10 most active accounts of our given timeframe.

Below is the bar chart representation of Account_ID vs (Number of Edges + Cards issued).



Conclusion-

We have determined some decent insights from the dataset provided and used data modelling concepts to extract answers from the data for our questions. After seeing the dataset several questions came to our mind, like, which region has taken most loans, what is the total number of loans issued per region, what's the average amount of loan issued every year, if there's an increasing trend of credit cards issuance every progressing year, which account most used, which accounts are the most active, etc. In order to answer these questions, we applied graph, relational, spatial-temporal modelling concepts on the data.

Firstly, we found out the best representation for our graph modelling. An experiment was carried out to see which of the three representation is the best. We performed an algorithmic cost analysis on each degree distribution calculation algorithm to see which algorithm takes maximum time to execute. We found that Adjacency Matrix is the worst and most impractical for our graph modelling and Edge List is the best. We also saw that Edge List and Adjacency List representations had similar execution time for the algorithm. These results were backed by the theoretical analysis of the representations of graphical cost models of degree distribution. We learnt in the class about the cost of each representation while calculating degree distribution, so applied those concepts and the gained results were in accordance with the algorithmic results.

Secondly, we created a graph out of our dataset using Edge List representation. Using that graph we started answering the questions that came to us while understanding the data set. We use graph modelling concepts integrated with the relational and temporal models to find out answers to our problems.

Lastly, after getting answers in the form of tables and relations, we used Tableau to create some beautiful data visualizations which helps us to understand the answers pictorially and efficiently.

Hence, these analyzations display some elegant insights of the data with the help of the modelling techniques taught in the class.