DATA WAREHOUSING ON TELECOM CHURN RATE DATA USING AWS REDSHIFT

1. Introduction

Customer churn refers to a process in which an existing customer who utilized the organization's services ceased using them or switched to utilizing alternative services that generated less income for the company. Therefore, a company's revenue can be directly impacted by customer churn. The churn issue is not exclusive to the telecom industry. Both the gambling and tourist industries struggle with high rates of consumer attrition. In the gaming industry, customers are likely to stop playing a game if it is too challenging. On the other hand, if the game is too simple, the player will find it too dull, which will eventually lead to churn from that game. One of the most often used systems for data warehousing, which is a popular technique for managing enormous volumes of data, is AWS Redshift. The purpose of this project is to create a data warehouse for analysis of telecom churn rates using AWS Redshift. The project's objective is to provide strategies that might be applied to reduce churn rates and provide a detailed knowledge of the factors that influence them. The project involves gathering and integrating data from many sources, formatting it for straightforward analysis, and using analytical techniques to get results. The article describes the analytical tools used to assess the data once it has been loaded into AWS Redshift, including SQL queries and data visualization tools. The report includes information on the customer demographics, usage patterns, and customer service experiences that influence turnover rates. The article also provides strategies for reducing churn rates, including improving customer service, offering customized discounts, and anticipating high-risk customers.[1]

2. Telecom churn analysis

The telecommunications industry is one of the key sectors accelerating the growth of numerous countries. Therefore, customer churn is a constant problem for businesses operating in this sector, both new and established.

The issue of customer turnover has been worse recently and is directly related to the emergence of competing telecom companies offering better services. Let's examine the causes of client turnover in the modern telecommunications industry: (1) Tougher Telecom Environment: The market is saturated due to the dominance of a small number of well-rounded firms, which sparks fierce pricing competition.

(2) Smarter and more demanding clients: Clients who compare prices are less loyal and seek better services at reduced prices. The churn in the telecom industry can take many different forms, some of which include: (1) Churn in Tariff Plans: When a subscriber or client switches from a Rs 60 to a Rs 20 plan, or vice versa, this form of churn typically takes place. One likely explanation is that the business isn't giving subscribers excellent value for their subscription money. (2) Service Churn: This kind of churn happens when a subscriber switches from a weekly plan to an annual plan, or from a monthly plan to a weekly plan, or vice versa. (3) Product Churn: This is the process through which a consumer switches from a post-paid plan to a prepaid plan or vice versa. Product churn can occur for a number of different causes. One explanation might be that post-paid subscription offerings are superior to those on prepaid markets.[1]

3. Literature review

Snowflake :-

Snowflake is a cloud-based software as a service (SaaS) data platform. Data sharing that is safe The degree of scalability is limitless. slick cloud-to-cloud contact. The platform is built on a model of a virtual warehouse and makes use of cloud computing services from third parties including Amazon Web Services, Microsoft Azure, and Google Cloud Platform. Organizations that have the choice to use high-performance cloud platforms benefit from real-time auto-scaling. Boost the effectiveness of workloads. Large numbers of requests may be handled by the elastic cloud. With a special SQL query engine and a three-layer architecture, the Snowflake analytics platform can handle real-time analytics of streaming big data. Without having to learn new programming languages, users may use its flexible design to construct their own analytical apps.[2]

Both Snowflake and Amazon Redshift are strong cloud-based data warehousing systems, however there are some significant differences between them. When compared to Redshift, Snowflake has the following advantages:

1. Architecture : Snowflake's innovative design isolates computation from storage resources, enabling businesses to expand resources separately and steer clear of performance problems brought on by conflict. Redshift, in contrast, combines computing and storage in a single cluster, which might cause performance problems as the cluster's size increases.

2. Query Performance : Snowflake's design enables it to deliver almost immediate query speed, even with huge and complicated datasets. In order to increase speed, Snowflake may also automatically optimize queries and distribute data across clusters. Redshift is renowned for its high performance, however further fine-tuning may be needed to improve query speed for particular workloads.

3. Support for a wide range of data formats, including unstructured, semi-structured, and structured data. Snowflake is capable of handling a wide range of data kinds. Organizations now find it simpler to store and evaluate data from many sources. Redshift may not be as effective at managing semi-structured or unstructured data because it is primarily intended for structured data.

4. Security: Snowflake offers numerous levels of protection, including access restrictions, encryption both at rest and in transit, and audit trails. In addition, Snowflake complies with several industry norms and laws, including SOC 2, HIPAA, and GDPR. Redshift offers security capabilities as well, but may need additional administration and configuration to guarantee compliance with particular rules.

5. Ease of Use :- Simple SQL syntax, an intuitive user interface, and built-in support for typical data warehousing processes like ETL and data sharing make Snowflake straightforward to use and administer. Redshift is likewise user-friendly, however managing complicated activities or unique processes could call for additional knowledge.

Overall, Snowflake is a popular option for businesses wishing to store and analyze huge amounts of data in the cloud because of its distinctive design, performance, support for a variety of data types, security, and ease of use.[2]

AWS Redshift :-

Using cloud-based computing nodes, AWS Redshift is a data warehousing system that enables massive data processing and storage. The platform employs column oriented databases to link business intelligence tools with SQL-based query engines. The platform combines PostgreSQL with Massively Parallel Processing (MPP) on dense storage nodes to offer quick query outputs on large data sets. Redshift offers a number of cluster management options in addition to quicker query processing. Here are a few examples: interactively using AWS CLI or

Amazon Redshift Console Query API for Amazon Redshift in the AWS Software Development Kit

Businesses can efficiently query and gather petabytes of data utilizing the fully managed warehousing platform Amazon Redshift. Organizations may get new insights from each data point in the application/system with the help of the cache included in Accelerator (AQUA), which speeds up query operations by up to 10 times. The user-friendly UI of AWS Redshift makes analytics and querying simple. a fully managed platform that requires little administration, upkeep, or maintenance. It functions in concert with the ecosystem of AWS services. [3]

3.1 Why are we using AWS Redshift for the given data

The advantages of the AWS Redshift when compared to snowflake are:-

1. Cost: For some workloads, notably those with fewer queries or simpler data sets, Redshift may be more cost-effective than Snowflake. Redshift price is determined by the cluster's number of nodes, whereas Snowflake pricing is determined on compute and storage use. Redshift could occasionally be more economical for lighter workloads.

2. Familiarity with the AWS ecosystem: Redshift may be simpler for organizations to connect with their current infrastructure and workflows if they currently use AWS services. It is simpler to move data across services and manage data pipelines because of Redshift's comprehensive integration with other AWS services like S3, Glue, and EMR.

3. speed for some workload types: Redshift may be a superior option for workloads with recurring queries or those requiring complicated transformations, notwithstanding Snowflake's reputation for quick query speed. Additionally, Redshift supports a number of SQL extensions, which is advantageous for businesses that need advanced analytics or data processing.

4. Redshift offers greater customization and tuning flexibility, which might be advantageous for businesses with particular performance or data management needs. Redshift users have access to a variety of

settings, such as table compression, sort keys, and distribution keys, which can assist optimize query speed for particular workloads.

5. Maturity: Redshift has a bigger user base and has been on the market longer than Snowflake, which might make it simpler to access support and resources. Additionally, Redshift has a strong ecosystem of third-party tools and integrations, which can be helpful for businesses that need a certain functionality.[5]

Due to the above reasons we are using the AWS Redshift over the snowflake for the given Telecom churn dataset. Since the amount of data is relatively small and cost effective AWS Redshift is preferred.

4. Methodology

The dataset was first collected from kaggle and it consisted of 7042 rows and 21 columns. With that dataset our goal was to data warehouse the dataset using AWS Redshift and which will help in handling the dataset in case of adding new data and improving the query processing time. AWS Redshift is also a cost effective method when compared to Snowflake. But for the amount of data we have, AWS Redshift was the best choice.

Once the data was collected it was loaded into the AWS Redshift by using S3. Here the data was first uploaded into s3 bucket and then IAM roles were created to create the AWS Redshift cluster and then cluster was then created. Once created we had to load the dataset into AWS Redshift using the query editor v2. There using the COPY command the data was loaded to analyze and check query processing. All of these processes are explained in detail with screenshots in the next section.
After this our predictive models were developed to predict the telecom churn rate using the dataset. The data was also analyzed in the beginning for the null values and missing values. Once the preprocessing of the data was done ML models were developed to predict the churn rate. And the highest accuracy was acquired using the XGBoost algorithm with 84% accuracy. The findings from this project were the factors that were involved in Telecom Churn rate and how the telecom companies use data to keep hold of their customers. And the necessary parameter involving customer churn rate.

5. Data Warehousing

Large volumes of data may be processed quickly and effectively for analytical purposes using Amazon Redshift, a data warehouse service. Redshift speeds up searches and the processing of massive datasets by utilizing columnar storage, data compression, and parallel computing. In this tutorial, we'll go through how to leverage data from Amazon S3 to build up a data warehouse in Redshift.

Step 1:- Create an Amazon S3 bucket.
We first create an S3 bucket by logging into the aws account, search for S3 and create a S3 bucket. Once the bucket is created we have to upload the data folder into the bucket.

Step 2 :- Create a Redshift Cluster
We create a redshift cluster by using the IAM role we created for this process and then choose d2.large database with 2 nodes. Lider and computer nodes.

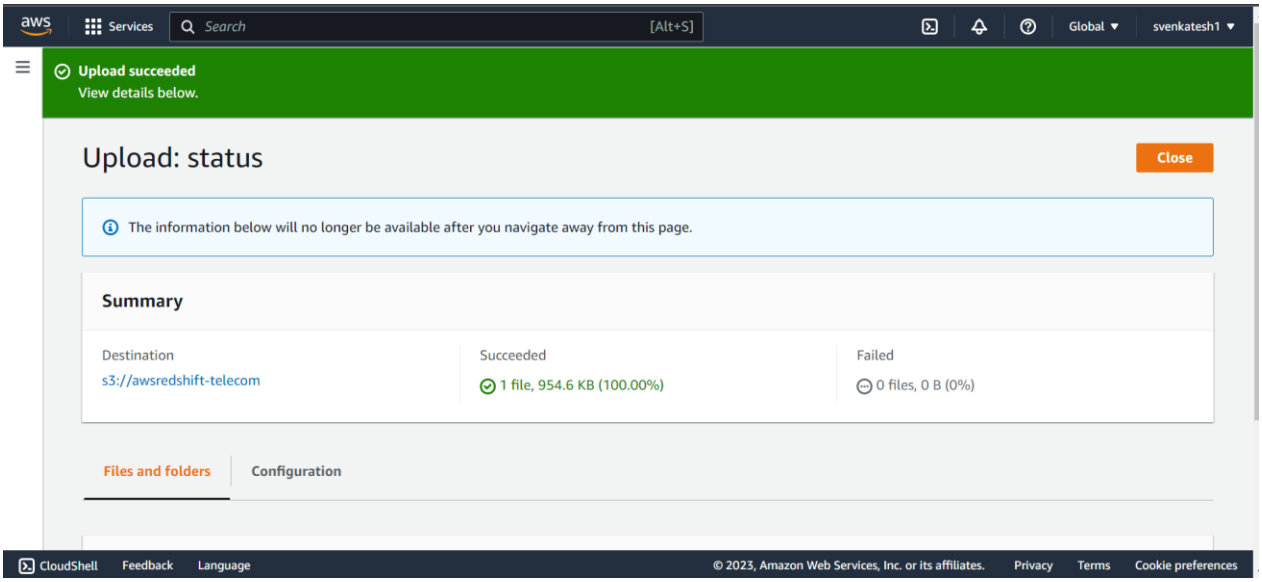
Step 3 :- Create Schema and tables
We then create schema and tables in the database. During this process we can also use SQL commands to create and define tables with appropriate names and datatypes.

Step 4 :- : Load data in Redshift
We then load the dataset in the Redshift in the Redshift query editor v2. Where the COPY command is used and then the dataset is loaded using the s3 bucket.

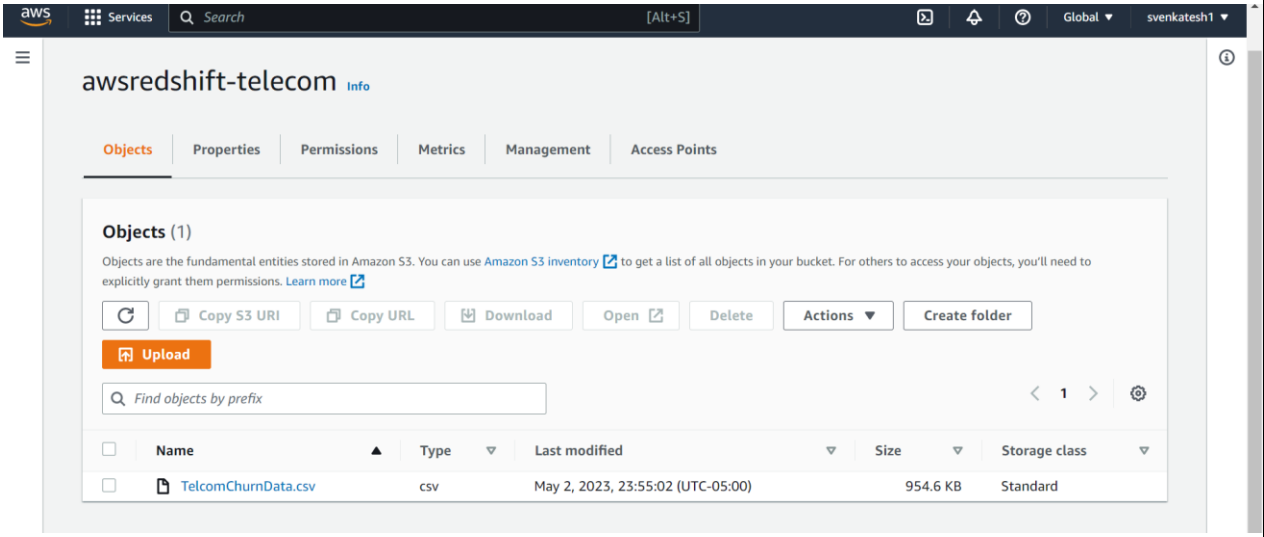
Step 5 :- Analyze and query the dataset
Once the dataset is loaded into the redshift we can analyze the dataset and check the query processing time of each SQL command.

All the above steps have been performed successfully, as shown below with screenshots of each process. [9]

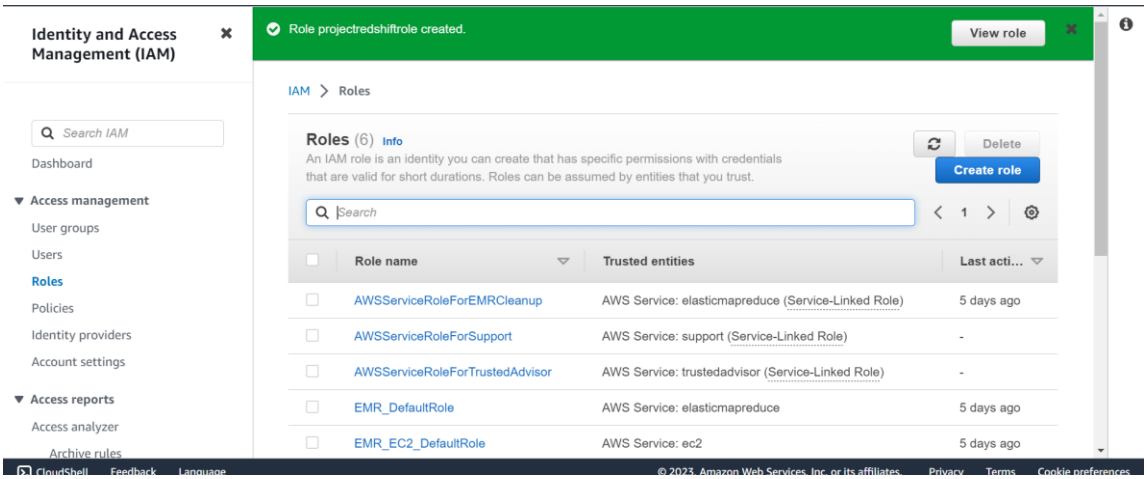
Step 1 :- We create a S3 bucket.



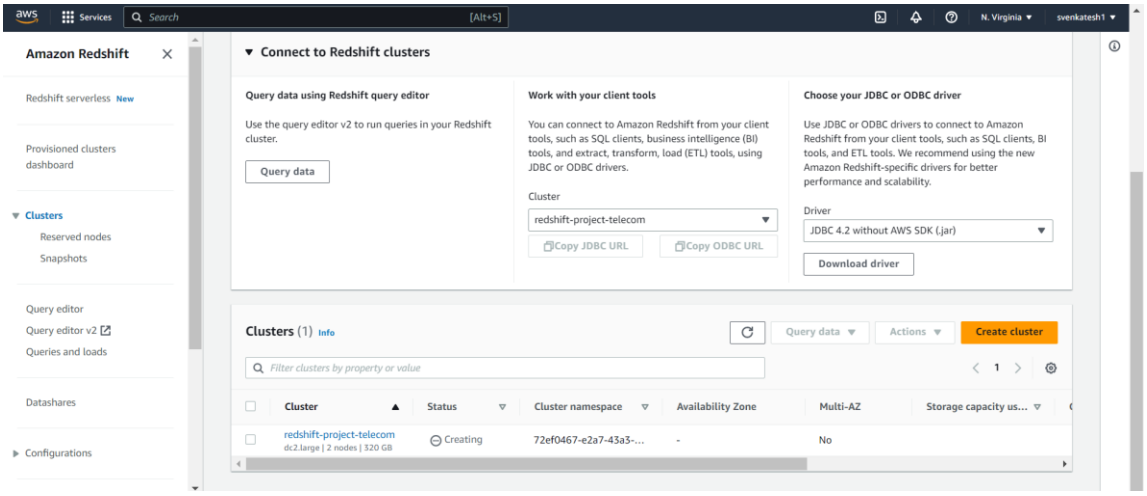
Step 2 :- Upload the CSV file into the S3 bucket.



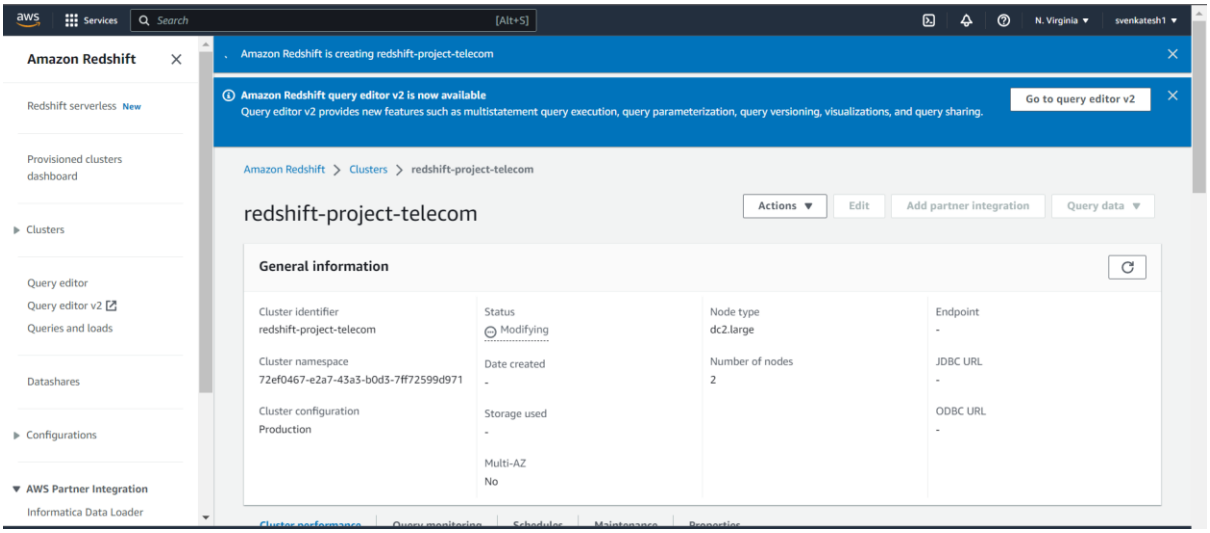
Step 3 :- Modifying and adding new IAM roles for the creation of the AWS Redshift cluster.

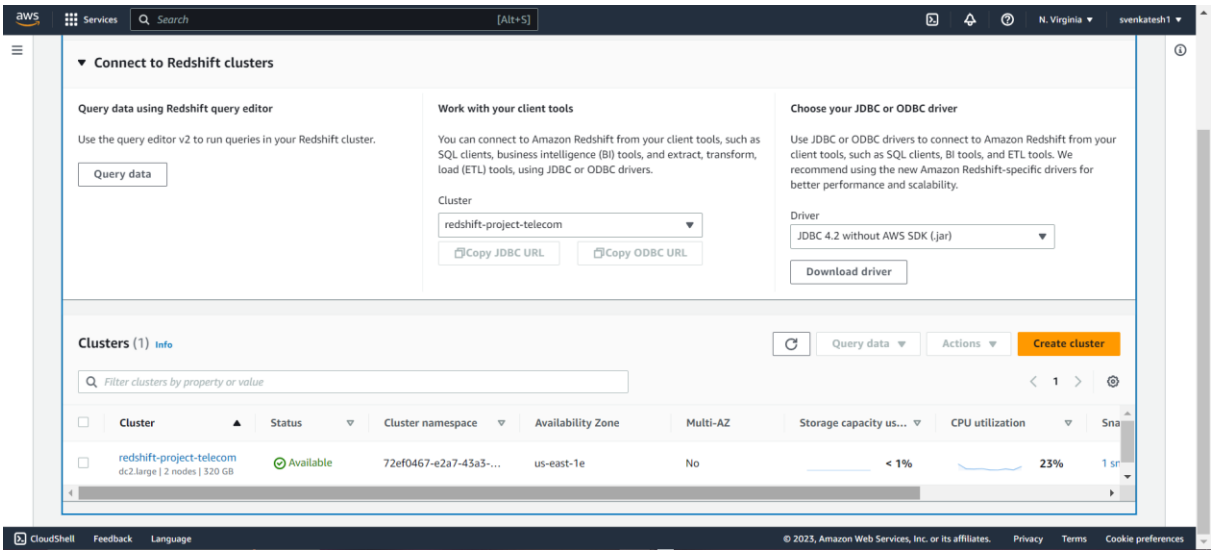


Step 4 :- Create the AWS Redshift cluster using the dc2.large database and the IAM roles specified in the previou step.

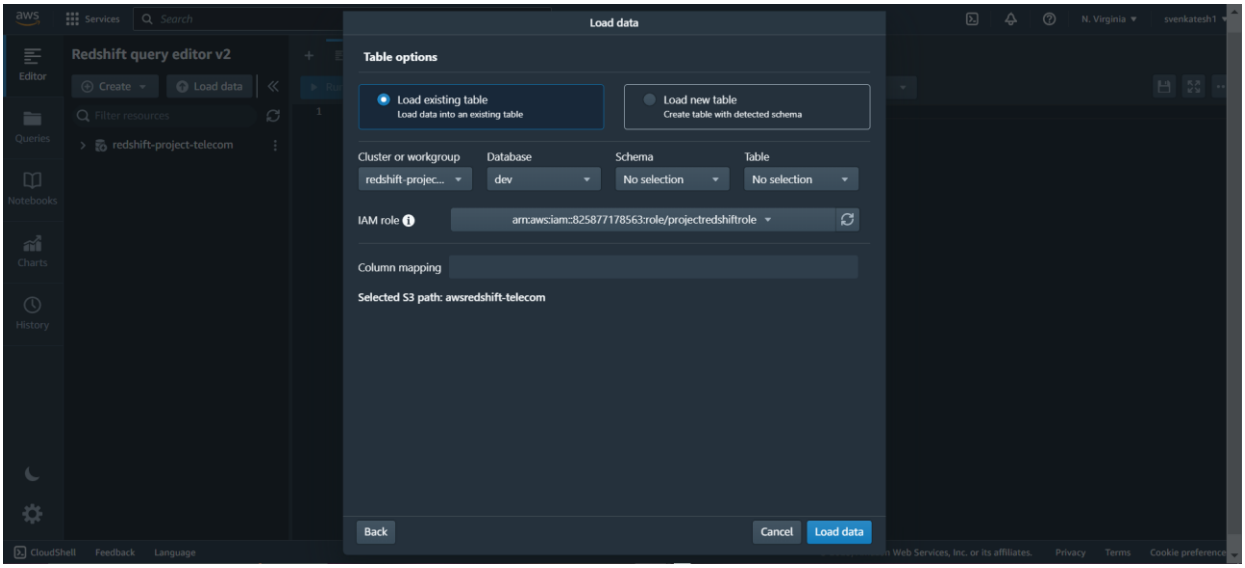


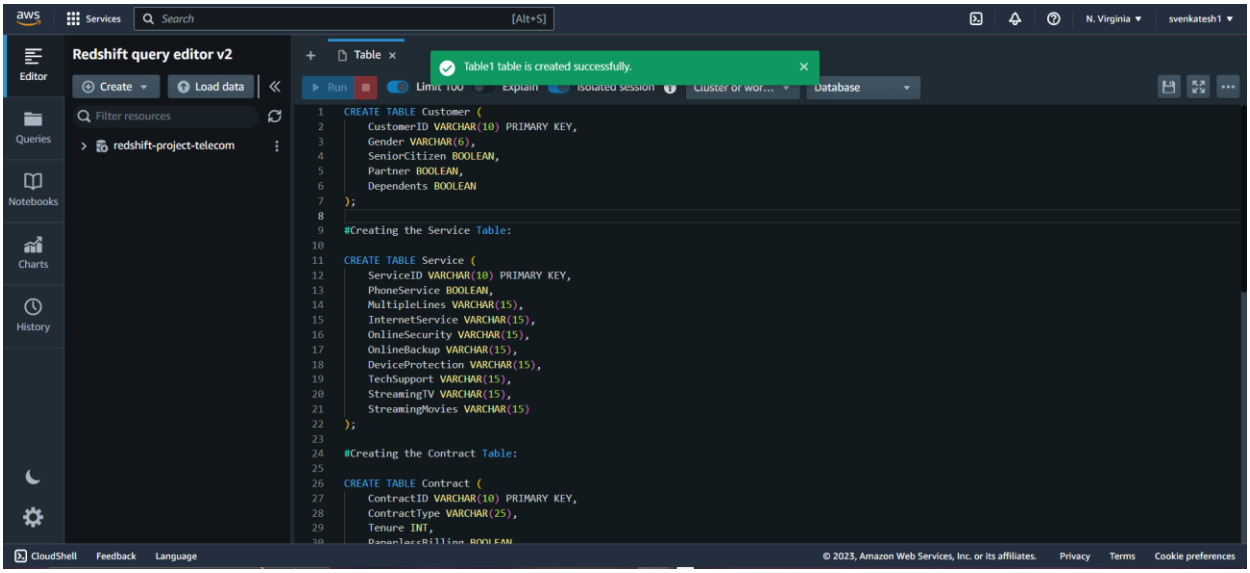
Step 5 :- We wait until the cluster becomes available, the database health is good, and the endpoint is generated.



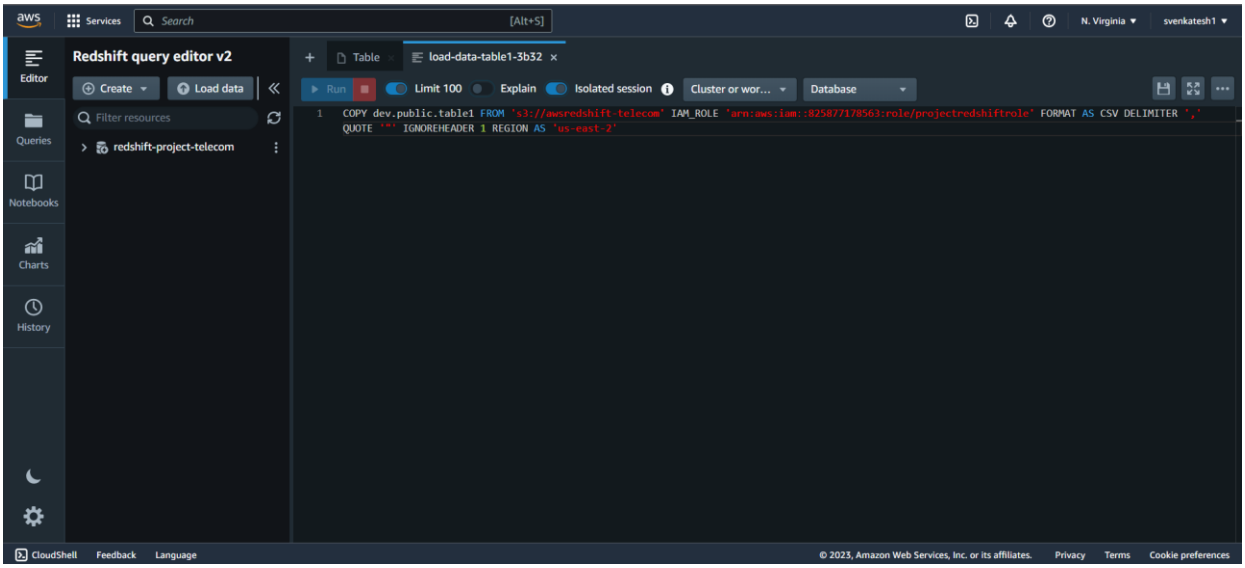


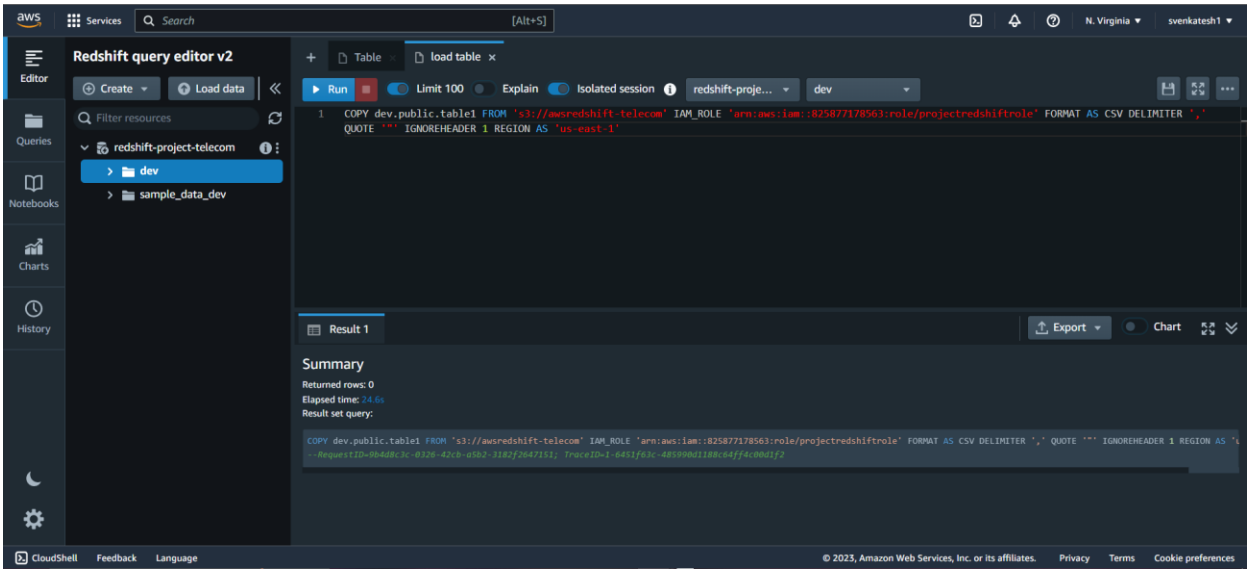
Step 6 :- Now we load the S3 into the redshift query editor.

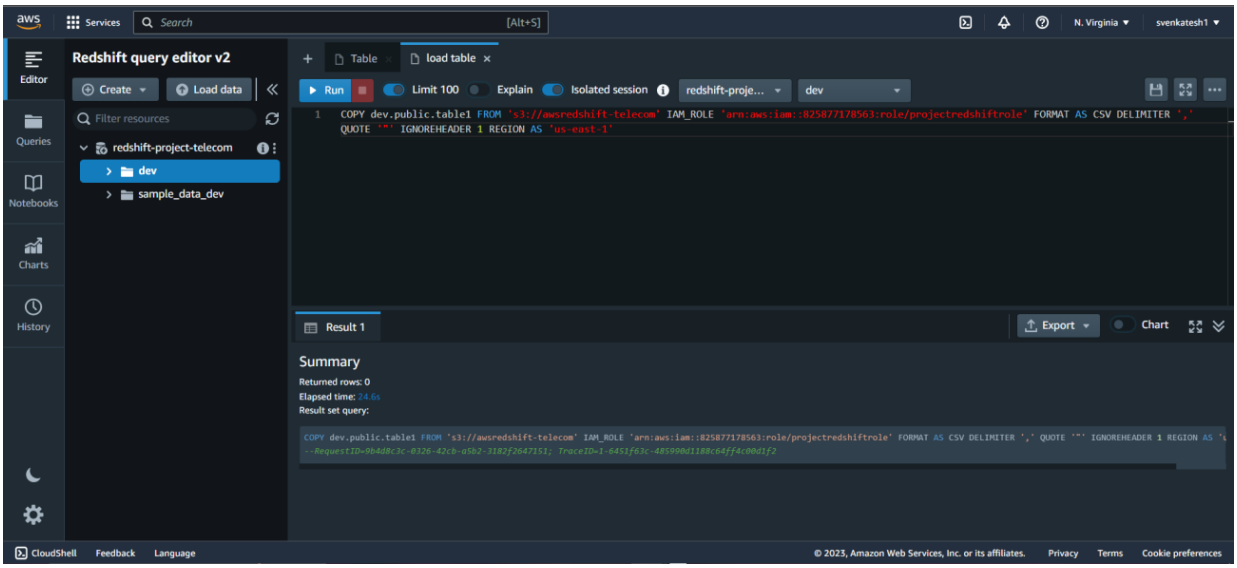




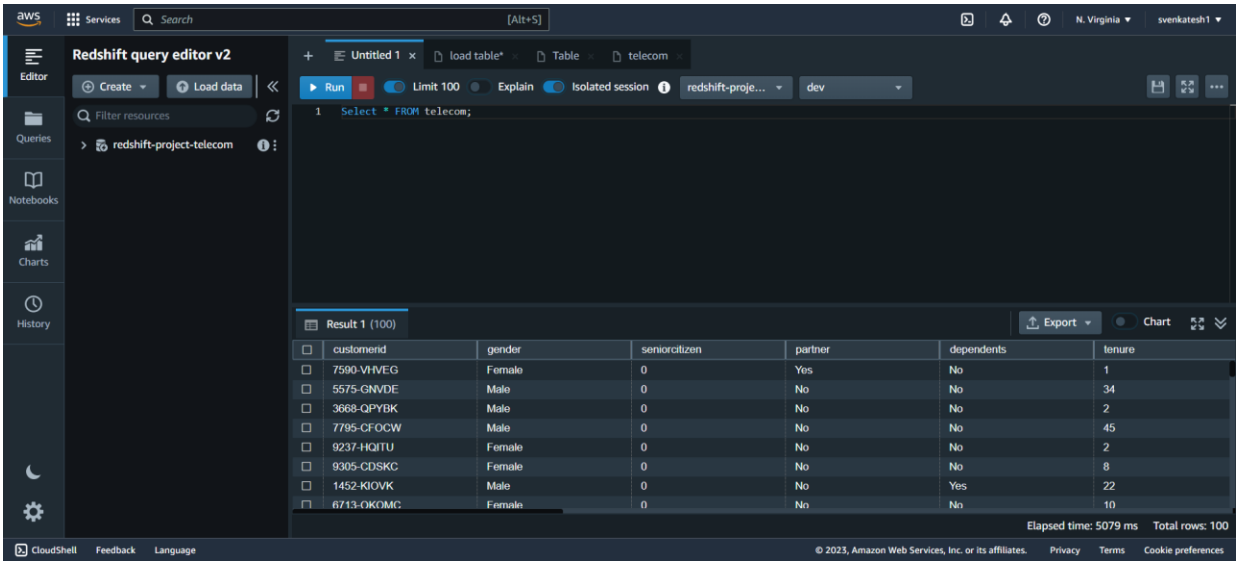
Step 7 :- Using the copy command, we connect the S3 bucket to the AWS Redshift using the IAM role.

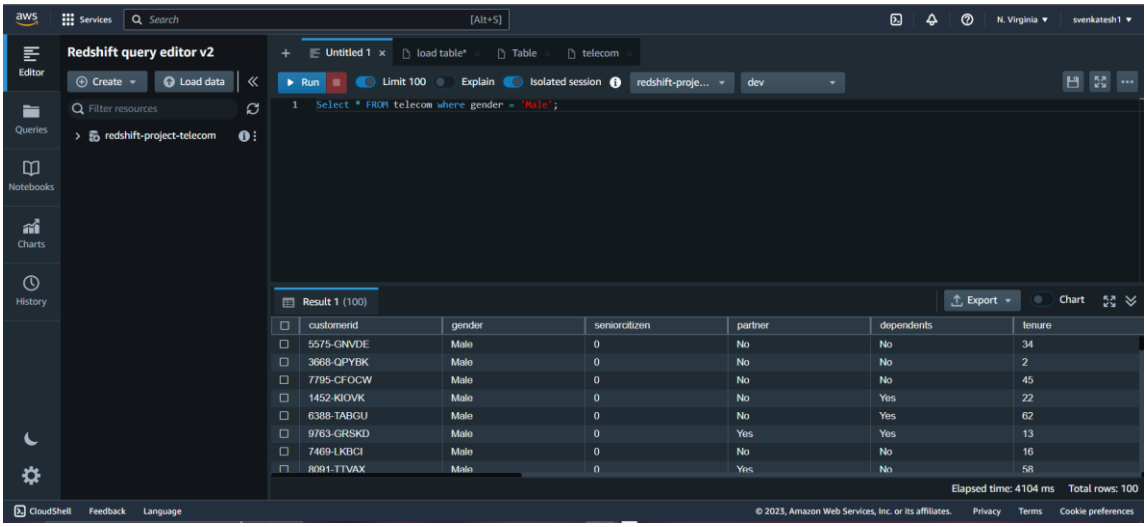

Redshift query editor v2
Create
Load data
Table
load-data-table1-3b32
Run
Limit 100
Explain
Isolated session
Cluster or wor...
Database
Filter resources
redshift-project-telecom
1 COPY dev.public.table1 FROM 's3://awsredshift-telecom' IAM_ROLE 'arn:aws:iam::825877178563:role/projectredshiftrole' FORMAT AS CSV DELIMITER ','
QUOTE '"' IGNOREHEADER 1 REGION AS 'us-east-2'

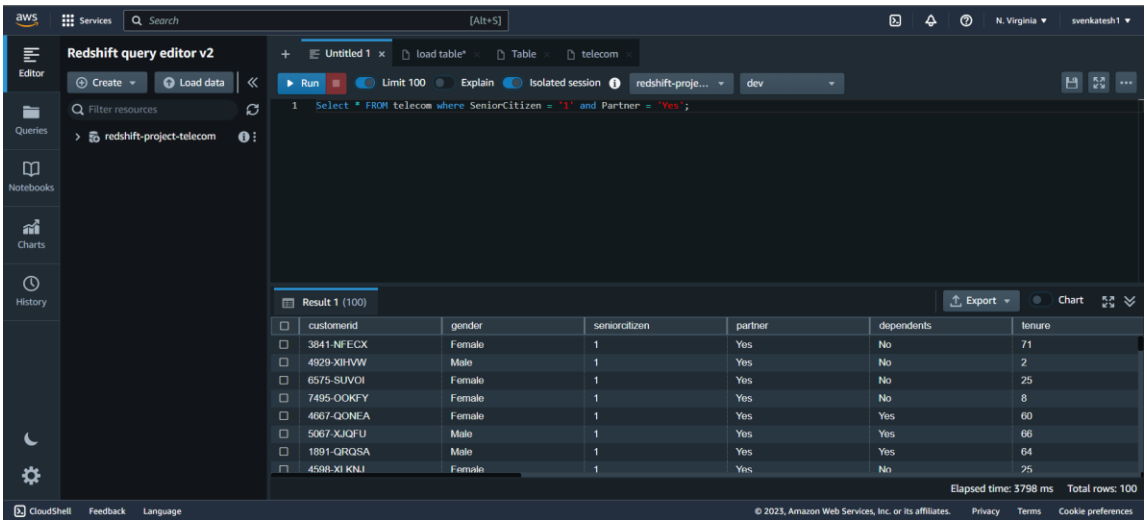

Redshift query editor v2
Create
Load data
Table
load table
Run
Limit 100
Explain
Isolated session
redshift-proje...
dev
Filter resources
redshift-project-telecom
dev
sample_data_dev
1 COPY dev.public.table1 FROM 's3://awsredshift-telecom' IAM_ROLE 'arn:aws:iam::825877178563:role/projectredshiftrole' FORMAT AS CSV DELIMITER ','
QUOTE '"' IGNOREHEADER 1 REGION AS 'us-east-1'
Result 1
Export
Chart
Summary
Returned rows: 0
Elapsed time: 24.6s
Result set query:

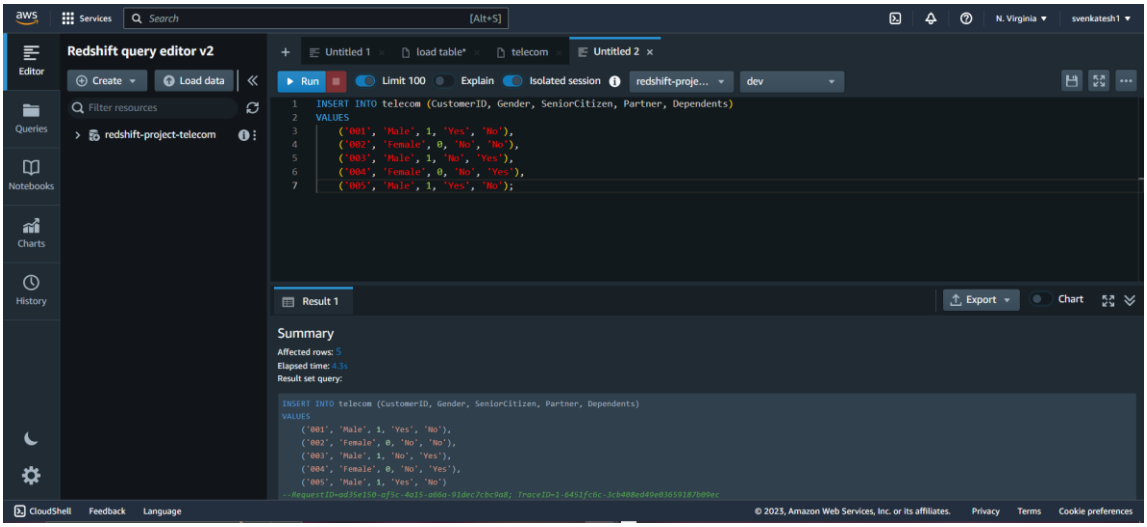


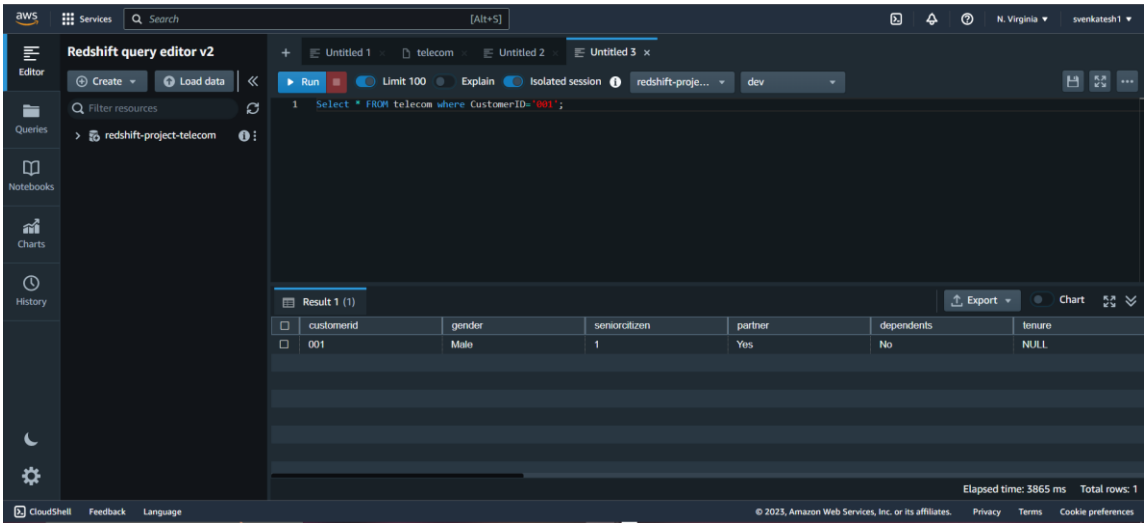
Step 8 :- Running SQL queries in Redshift query editor v2.




Redshift query editor v2
Untitled 1
load table*
Table
telecom
Run
Limit 100
Explain
Isolated session
redshift-proje...
dev
1 Select * FROM telecom where gender = 'Male';
redshift-project-telecom
Result 1 (100)
Export
Chart
customerid gender seniorcitizen partner dependents tenure
5575-GNVDE Male 0 No No 34
3668-QPYBK Male 0 No No 2
7795-CFOCW Male 0 No No 45
1452-KIOVK Male 0 No Yes 22
6388-TABGU Male 0 No Yes 62
9763-GRSKD Male 0 Yes Yes 13
7469-LKBCI Male 0 No No 16
Elapsed time: 4104 ms
Total rows: 100


Redshift query editor v2
Untitled 1
load table*
Table
telecom
Run
Limit 100
Explain
Isolated session
redshift-proje...
dev
1 Select * FROM telecom where SeniorCitizen = '1' and Partner = 'Yes';
redshift-project-telecom
Result 1 (100)
Export
Chart
customerid gender seniorcitizen partner dependents tenure
3841-NFECX Female 1 Yes No 71
4929-XIHVW Male 1 Yes No 2
6575-SUVOI Female 1 Yes No 25
7495-OOKFY Female 1 Yes No 8
4667-QONEA Female 1 Yes Yes 60
5067-XJQFU Male 1 Yes Yes 66
1891-QRQSA Male 1 Yes Yes 64
Elapsed time: 3798 ms
Total rows: 100


Redshift query editor v2
Untitled 1
load table*
telecom
Untitled 2
Run
Limit 100
Explain
Isolated session
redshift-proje...
dev
1 INSERT INTO telecom (CustomerID, Gender, SeniorCitizen, Partner, Dependents)
2 VALUES
3 ('001', 'Male', 1, 'Yes', 'No'),
4 ('002', 'Female', 0, 'No', 'No'),
5 ('003', 'Male', 1, 'No', 'Yes'),
6 ('004', 'Female', 0, 'No', 'Yes'),
7 ('005', 'Male', 1, 'Yes', 'No');
redshift-project-telecom
Result 1
Export
Chart
Summary
Affected rows: 5
Elapsed time: 4.3s
Result set query:
INSERT INTO telecom (CustomerID, Gender, SeniorCitizen, Partner, Dependents)
VALUES
('001', 'Male', 1, 'Yes', 'No'),
('002', 'Female', 0, 'No', 'No'),
('003', 'Male', 1, 'No', 'Yes'),
('004', 'Female', 0, 'No', 'Yes'),
('005', 'Male', 1, 'Yes', 'No')



The above given snippets show the implementation of AWS Redshift query editor to show the query performance and to optimize query performance.

6. Database Design

Customer Table:

This table contains information about customers such as their ID, gender, senior citizen status, and whether they have a partner or dependents.

CustomerID (Primary key)
Gender
SeniorCitizen
Partner
Dependents
Service Table:

This table contains information about the services that customers have subscribed to such as phone service, multiple lines, internet service, and various streaming services.

ServiceID (Primary key)
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup

DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract Table:

This table contains information about the contracts that customers have with the company such as the contract type, tenure, payment method, and billing details.

ContractID (Primary key)
ContractType
Tenure
PaperlessBilling
PaymentMethod
MonthlyCharges
TotalCharges
Churn
Customer_Service Bridge Table:`

This table is a bridge table that links the Customer and Service tables, allowing customers to have multiple services and services to be subscribed to by multiple customers.

CustomerID (Foreign key referencing Customer table)
ServiceID (Foreign key referencing Service table)
Customer_Contract Bridge Table:

This table is a bridge table that links the Customer and Contract tables, allowing customers to have multiple contracts and contracts to be assigned to multiple customers.

CustomerID (Foreign key referencing Customer table)
ContractID (Foreign key referencing Contract table)
Referral Table:

This table contains information about the customers who referred others to the company and the customers they referred.

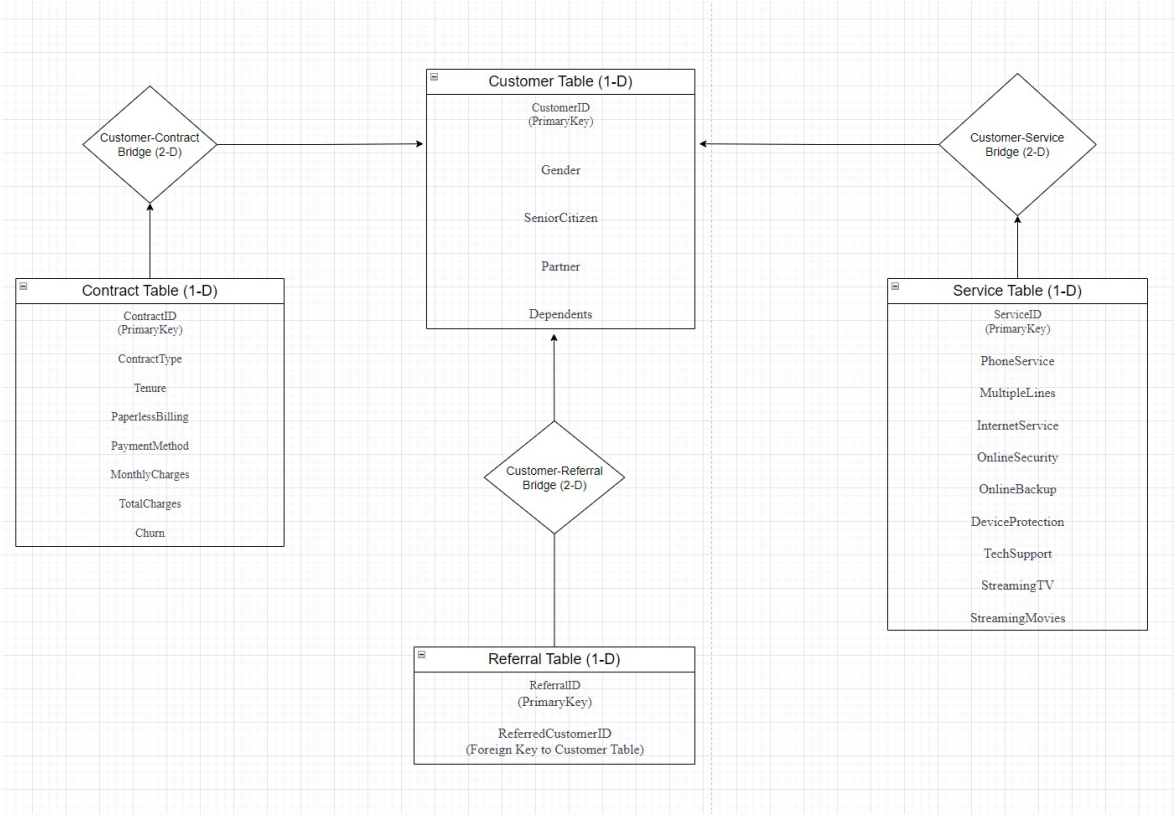
Attributes:

ReferralID (Primary key)
ReferredCustomerID (Foreign key referencing Customer table)
Customer_Referral Bridge Table:

This table is a bridge table that links the Customer and Referral tables, allowing customers to be referred by multiple other customers and for customers to refer to multiple other customers.

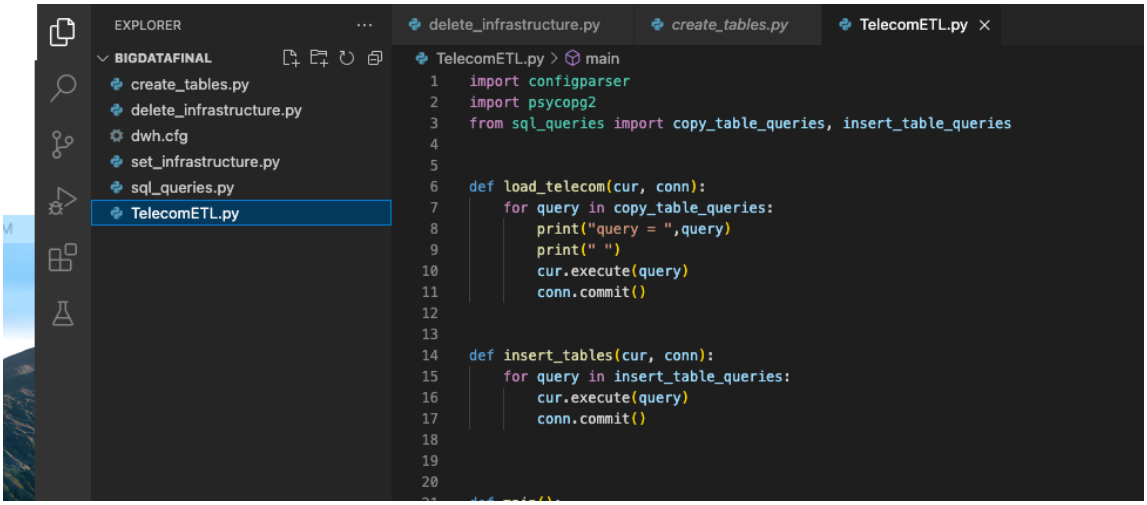
Attributes:

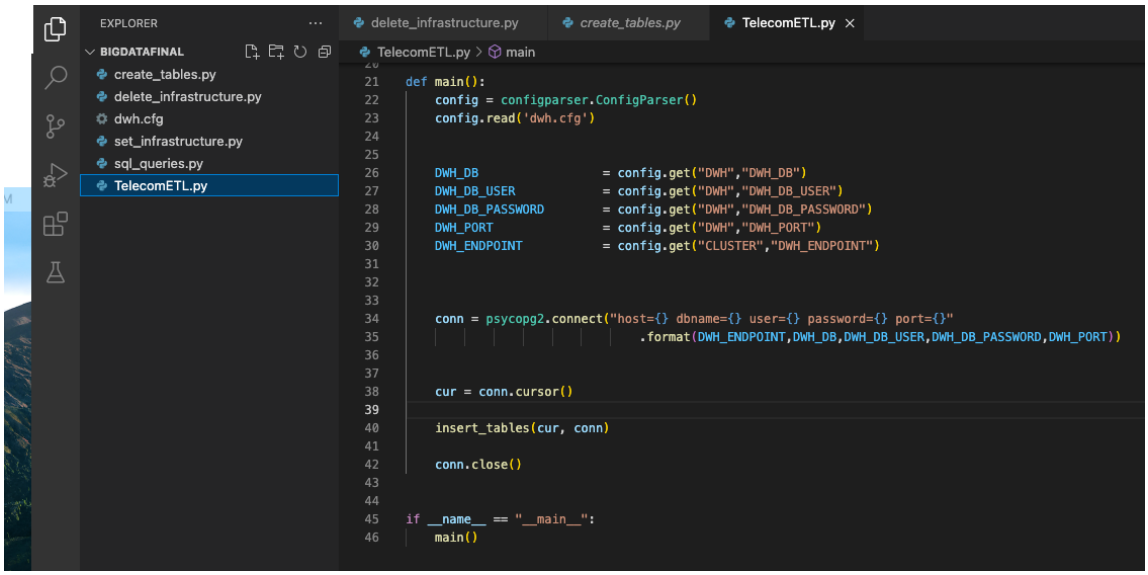
CustomerID (Foreign key referencing Customer table)
ReferralID (Foreign key referencing Referral table)

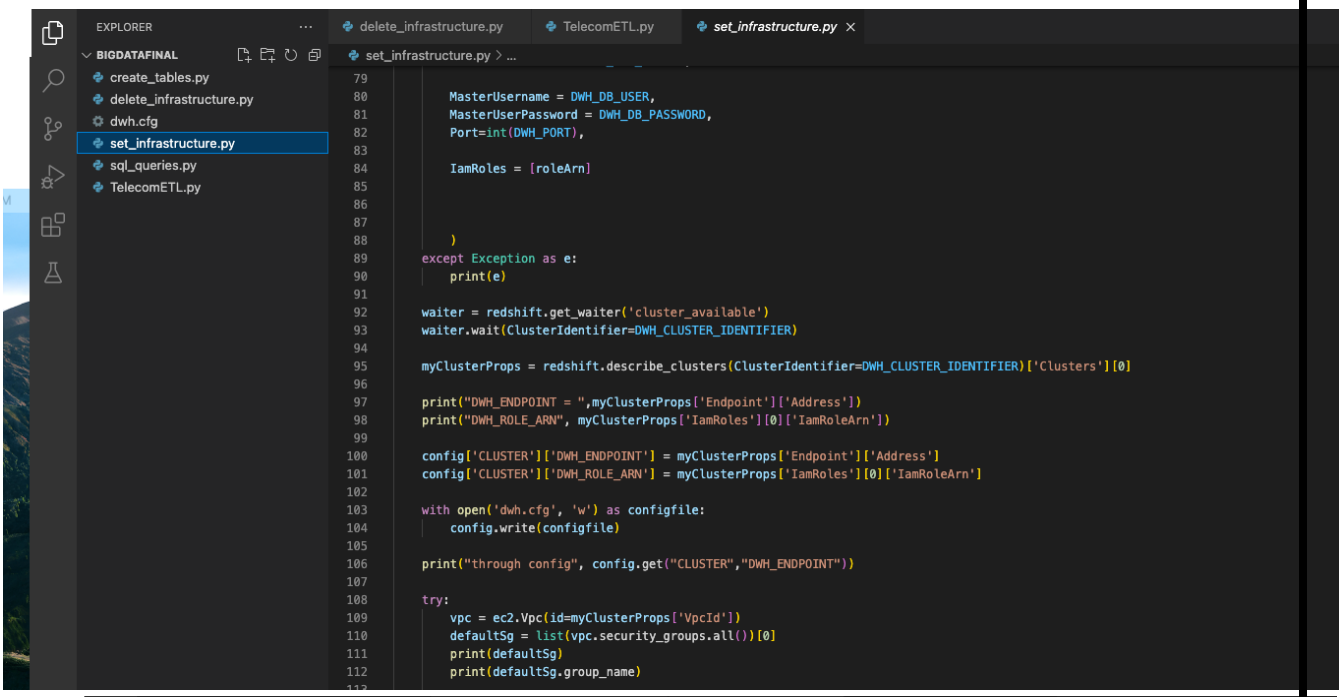


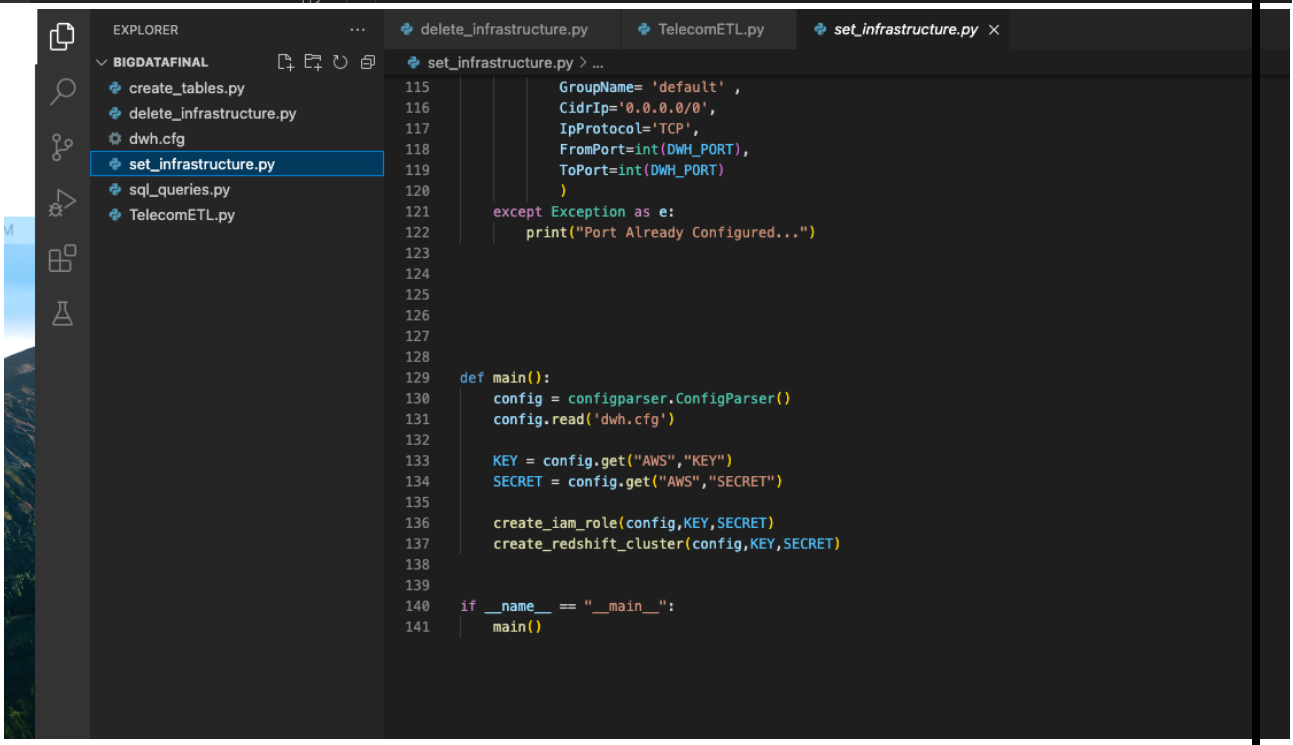
7. Implementation

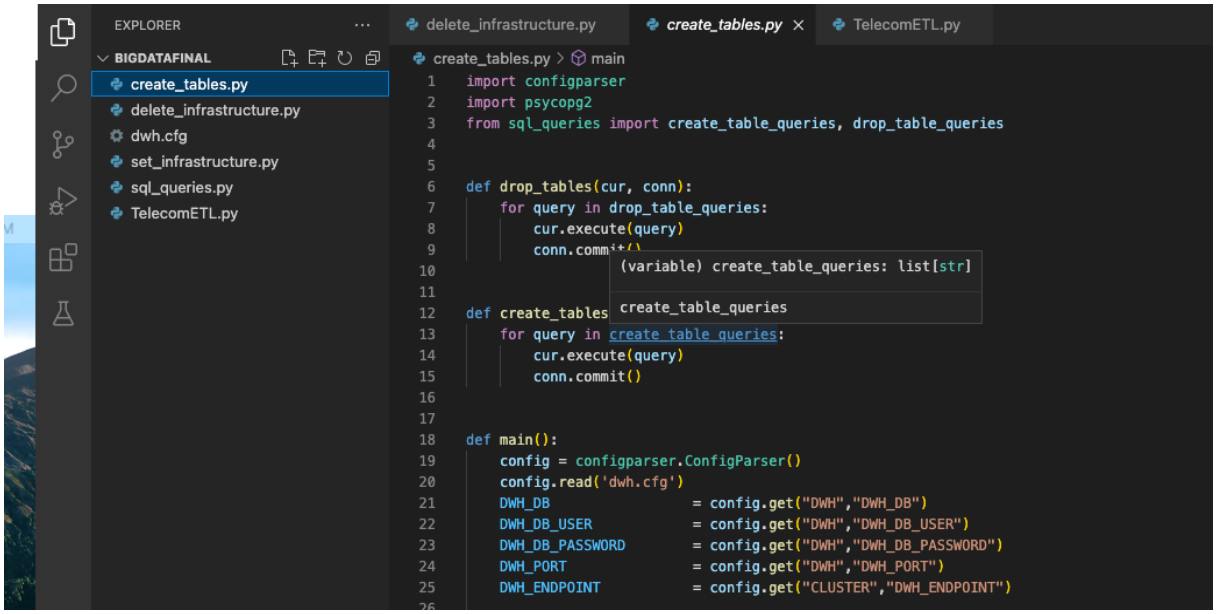
Here, in the beginning the ETL process is done which is shown in the below snippet where using the TelecomETL.py file we can complete the ETL process by running the TelecomETL.py file.

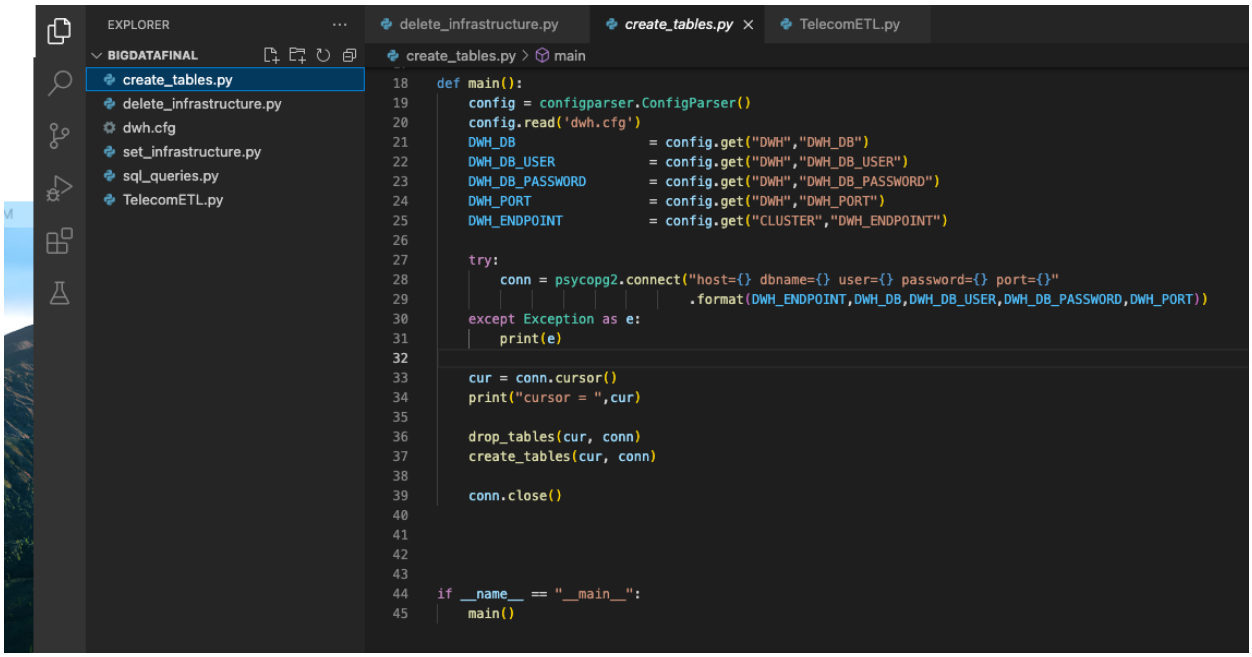





EXPLORER
BIGDATAFINAL
create_tables.py
delete_infrastructure.py
dwh.cfg
set_infrastructure.py
sql_queries.py
TelecomETL.py
delete_infrastructure.py
TelecomETL.py
set_infrastructure.py
set_infrastructure.py > ...
MasterUsername = DWH_DB_USER,
MasterUserPassword = DWH_DB_PASSWORD,
Port=int(DWH_PORT),
IamRoles = [roleArn]
)
except Exception as e:
print(e)
waiter = redshift.get_waiter('cluster_available')
waiter.wait(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)
myClusterProps = redshift.describe_clusters(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)['Clusters'][0]
print("DWH_ENDPOINT = ",myClusterProps['Endpoint']['Address'])
print("DWH_ROLE_ARN", myClusterProps['IamRoles'][0]['IamRoleArn'])
config['CLUSTER']['DWH_ENDPOINT'] = myClusterProps['Endpoint']['Address']
config['CLUSTER']['DWH_ROLE_ARN'] = myClusterProps['IamRoles'][0]['IamRoleArn']
with open('dwh.cfg', 'w') as configfile:
config.write(configfile)
print("through config", config.get("CLUSTER","DWH_ENDPOINT"))
try:
vpc = ec2.Vpc(id=myClusterProps['VpcId'])
defaultSg = list(vpc.security_groups.all())[0]
print(defaultSg)
print(defaultSg.group_name)


EXPLORER
BIGDATAFINAL
create_tables.py
delete_infrastructure.py
dwh.cfg
set_infrastructure.py
sql_queries.py
TelecomETL.py
delete_infrastructure.py
TelecomETL.py
set_infrastructure.py
set_infrastructure.py > ...
GroupName= 'default' ,
CidrIp='0.0.0.0/0',
IpProtocol='TCP',
FromPort=int(DWH_PORT),
ToPort=int(DWH_PORT)
)
except Exception as e:
print("Port Already Configured...")
def main():
config = configparser.ConfigParser()
config.read('dwh.cfg')
KEY = config.get("AWS","KEY")
SECRET = config.get("AWS","SECRET")
create_iam_role(config,KEY,SECRET)
create_redshift_cluster(config,KEY,SECRET)
if __name__ == "__main__":
main()





Python script that connects to a Redshift cluster using psycopg2 library, reads the database credentials and queries from a configuration file (dwh.cfg), and then loads and inserts data into the Redshift database using SQL queries defined in the separate file sql_queries.py.

The load_telecom() function iterates over the list of COPY queries defined in sql_queries.py to load data from S3 into the Redshift cluster. The insert_tables() function iterates over the list of INSERT queries defined in sql_queries.py to transform and insert data into the fact and dimension tables.

The main() function is the entry point to the script, which first reads the configuration file to get the Redshift cluster endpoint, database name, user name, password, and port. It then connects to the cluster using the psycopg2 library, calls the insert_tables() function to load data into the tables, and closes the connection after all data is inserted. Creating an AWS Redshift cluster and an IAM role using the Boto3 Python library for AWS. It reads configuration details from a configuration file dwh.cfg and uses them to create the cluster and role.The create_iam_role function creates an IAM role with a trust relationship policy allowing Redshift to assume the role. It then attaches the AmazonS3ReadOnlyAccess policy to the role, allowing Redshift to access data in S3. The function also updates the configuration file with the role's ARN. The create_redshift_cluster function creates a Redshift cluster with the specified configuration, including the IAM role created earlier. It then waits for the cluster to be available and retrieves its properties. The function updates the configuration file with the cluster's endpoint and role ARN, and then authorizes inbound traffic on the cluster's security group on the specified port. Finally, the main function reads the configuration file, retrieves the AWS access key and secret access key, and calls the create_iam_role and create_redshift_cluster functions.

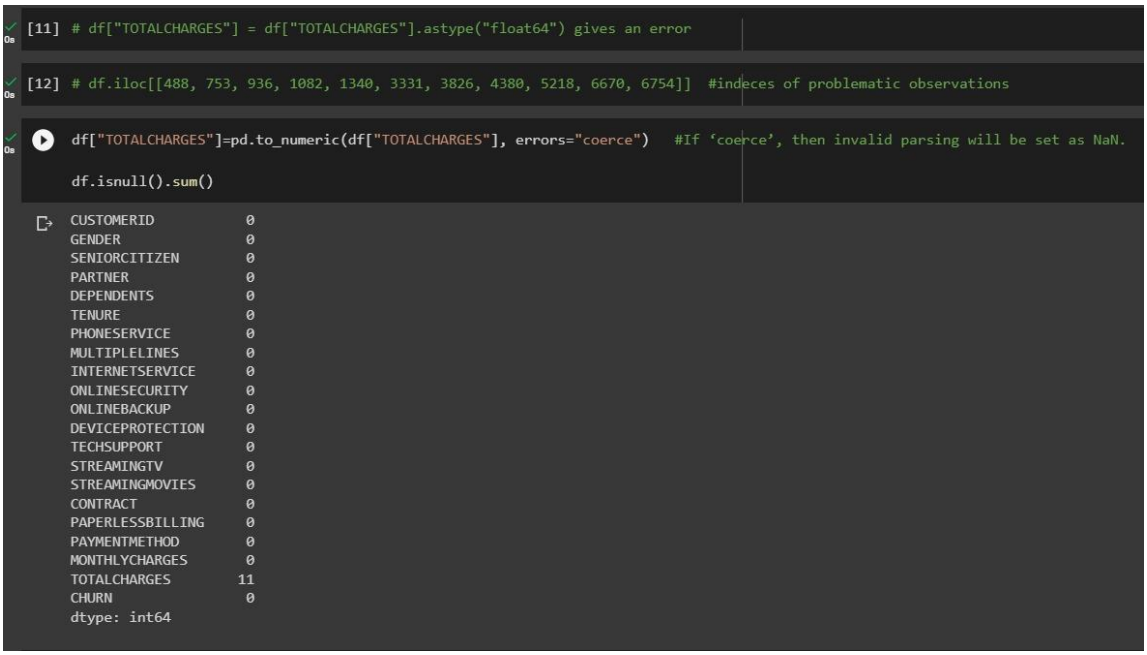
delete_infrastructure.py is used to The delete_cluster function takes in a configuration file (config), AWS access key ID (KEY), and AWS secret access key (SECRET). It uses the boto3.client method to create a Redshift client and the delete_cluster method to delete the cluster with the identifier specified in the configuration file. The SkipFinalClusterSnapshot parameter is set to True to skip taking a final snapshot of the cluster before deletion. The function then waits for the cluster to be deleted using a waiter object. The delete_iam_role function takes in the same inputs and deletes the IAM role with the name specified in the configuration file. Before deleting the role, it uses the detach_role_policy method to detach the "AmazonS3ReadOnlyAccess" policy from the role.

The main function reads the configuration file, gets the AWS access key ID and secret access key, and calls the delete_iam_role and delete_cluster functions. It then prints a message indicating that all infrastructure has been deleted. Overall, this code seems to be a useful script for cleaning up an AWS environment by deleting a Redshift cluster and an IAM role.

7.1 Prediction and analytics

Here, in the below snippets show the predictive and analytics of the telecom churn data. First, preprocessing work is done where the data is explored and then checked for avg, mean and std of each column and then null values are checked. Once that is done the EDA process occurs where various plots are plotted to analyze the data. Then finally ml predictions are done where XGboost algorithm gave the highest accuracy of 84%.

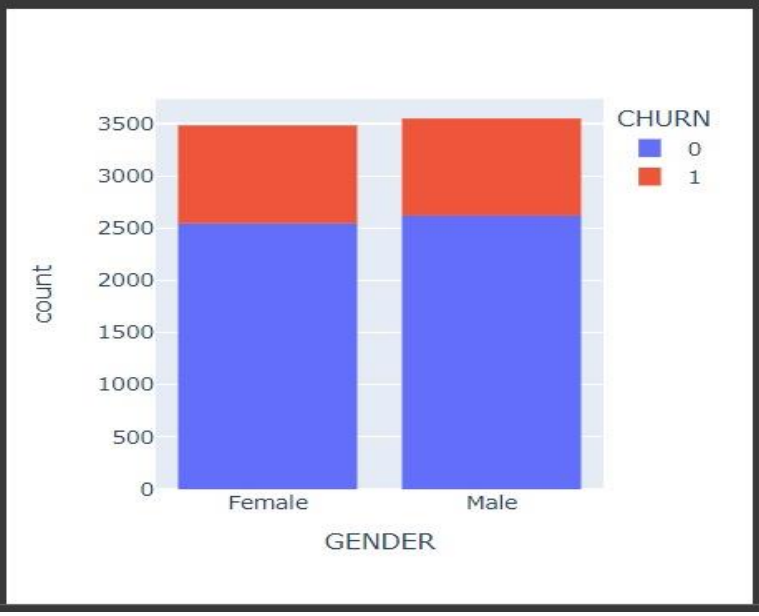
```
[7]  df["CUSTOMERID"].nunique()

     7043

[8]  df.CHURN.value_counts()/len(df)

     No     0.73463
     Yes    0.26537
     Name: CHURN, dtype: float64

[9]  df["CHURN"] = np.where(df["CHURN"] == "Yes", "1", "0")
     df["CHURN"] = df["CHURN"].astype("int64")

     df.CHURN.value_counts()/len(df)

     0    0.73463
     1    0.26537
     Name: CHURN, dtype: float64
```



```
[11] # df["TOTALCHARGES"] = df["TOTALCHARGES"].astype("float64") gives an error

[12] # df.iloc[[488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754]]  #indeces of problematic observations

     df["TOTALCHARGES"]=pd.to_numeric(df["TOTALCHARGES"], errors="coerce")   #If 'coerce', then invalid parsing will be set as NaN.

     df.isnull().sum()

     CUSTOMERID          0
     GENDER              0
     SENIORCITIZEN       0
     PARTNER             0
     DEPENDENTS          0
     TENURE              0
     PHONESERVICE        0
     MULTIPLELINES       0
     INTERNETSERVICE     0
     ONLINESECURITY      0
     ONLINEBACKUP        0
     DEVICEPROTECTION    0
     TECHSUPPORT         0
     STREAMINGTV         0
     STREAMINGMOVIES     0
     CONTRACT            0
     PAPERLESSBILLING    0
     PAYMENTMETHOD       0
     MONTHLYCHARGES      0
     TOTALCHARGES       11
     CHURN               0
     dtype: int64
```

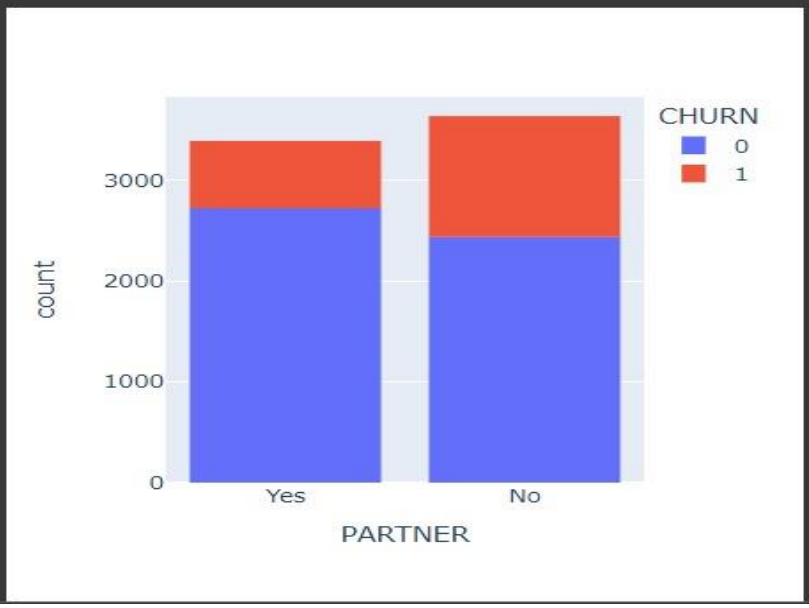
```
for col in cat_cols:
    cat_summary(df, col, "CHURN", plot=True)
```

```
        GENDER      Ratio
Male      3549  50.469283
Female    3483  49.530717
##########################################
        TARGET_MEAN
GENDER
Female     0.269595
Male       0.262046
```



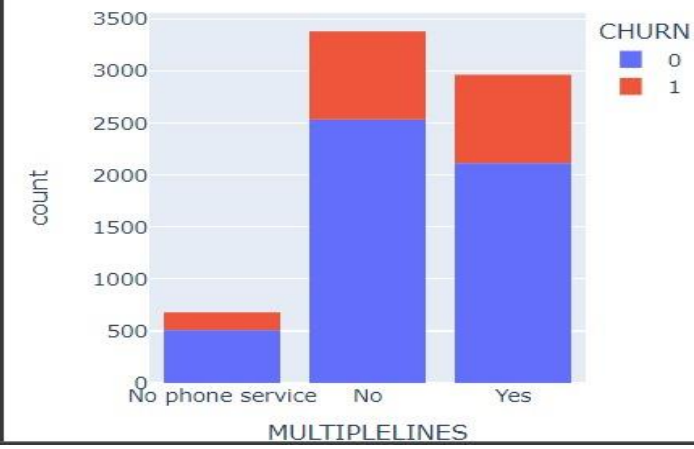
```
for col in cat_cols:
    cat_summary(df, col, "CHURN", plot=True)
```

```
     PARTNER      Ratio
No      3639  51.749147
Yes     3393  48.250853
##########################################
         TARGET_MEAN
PARTNER
No          0.329761
Yes         0.197171
```



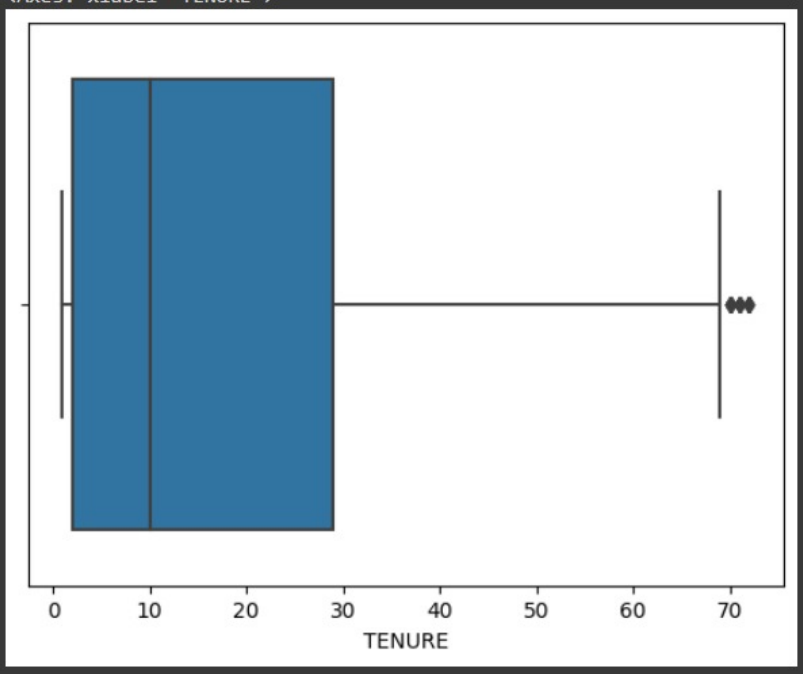
```
for col in cat_cols:
    cat_summary(df, col, "CHURN", plot=True)
```

```
                    MULTIPLELINES      Ratio
No                           3385  48.137088
Yes                          2967  42.192833
No phone service              680   9.670080
##########################################
                    TARGET_MEAN
MULTIPLELINES
No                     0.250812
No phone service       0.250000
Yes                    0.286485
```



```
sns.boxplot(x=df.loc[df["CHURN"]==1,"TENURE"])
```

```
<Axes: xlabel='TENURE'>
```




```
y = df["CHURN"]
X = df.drop(["CHURN"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=45)
```

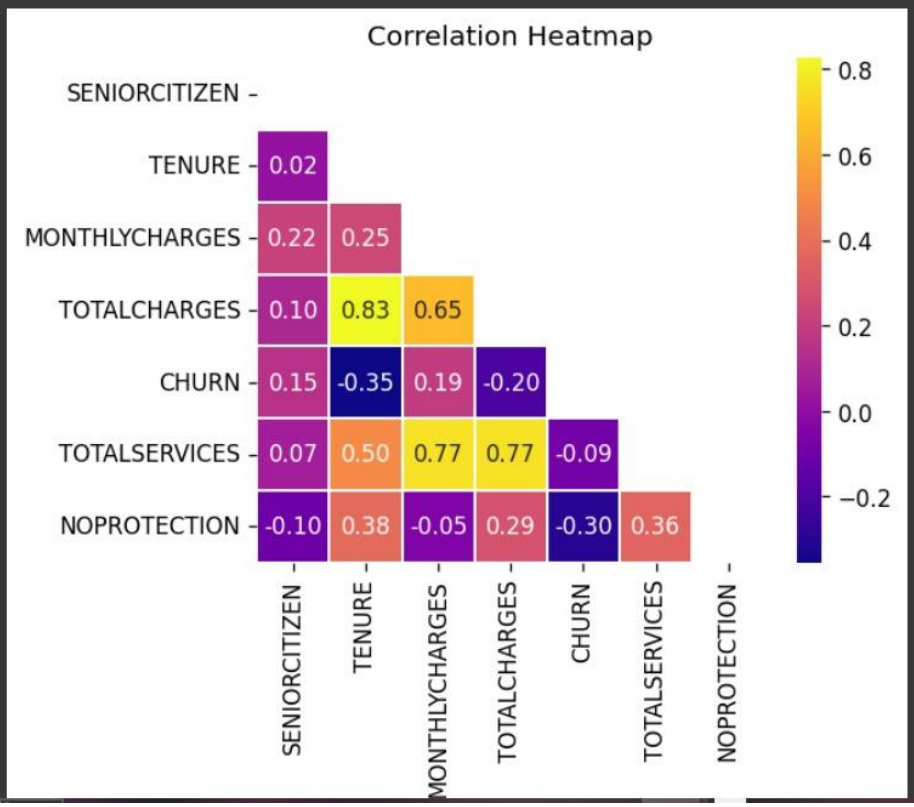
```
[86] lr_model = LogisticRegression(random_state=17).fit(X_train, y_train)
     y_pred = lr_model.predict(X_test)
     y_prob = lr_model.predict_proba(X_test)[:, 1]
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1552
           1       0.67      0.53      0.59       558

    accuracy                           0.81      2110
   macro avg       0.75      0.72      0.73      2110
weighted avg       0.80      0.81      0.80      2110
```

```
[87] roc_auc_score(y_test, y_prob)
```

```
0.8444526429072904
```

```
plt.figure(dpi = 120,figsize= (5,4))
mask = np.triu(np.ones_like(df.corr(),dtype = bool))
sns.heatmap(df.corr(),mask = mask, fmt = ".2f",annot=True,lw=1,cmap = 'plasma')
plt.yticks(rotation = 0)
plt.xticks(rotation = 90)
plt.title('Correlation Heatmap')
plt.show(block=True)
```



8. Conclusion

The report shows the process of data warehousing of Telecom Churn data using AWS Redshift and there are two implementations. First method is using the AWS Redshift cluster and the data is loaded using s3 bucket. And the schema shown above is used and then connected with AWS redshift. The other process uses ETL python operations as shown in the above code snippets which does the same process. Once the data is loaded it is analyzed and checked for query performance. And Query performance in AWS redshift is really fast. Then predictive analysis is done to predict Telecom churn rate with 84% accuracy using the XGboost algorithm.

9. References

[1] Yash Pandey, Roshan Jha, Dr. Umamaheshwari et al. Customer Churn analysis in telecom organization, Journal of Positive School Psychology.
https://journalppw.com/index.php/jpsp/article/download/4316/2855/4909

[2]https://www.snaplogic.com/blog/snowflake-data-platform

[3]https://www.bmc.com/blogs/aws-redshift-vs-snowflake/

[5]Chen, H., Chiang, R.H., Storey, V.C.Business intelligence and analytics: From big data to big impact. MIS quarterly pp. 1165–1188 (2012).

[6] Ullah, Irfan & Raza, Basit & Malik, Ahmad & Imran, Muhammad & Islam, Saif & Kim, Sung Won. (2019). A Churn Prediction Model
Using Random Forest: Analysis of Machine Learning Techniques for Churn Prediction and Factor Identification in Telecom Sector. IEEEAccess. PP. 1-1.10.1109/ACCESS.2019.2914999.

[7] Shaaban, Essam & Helmy, Yehia & Khedr, Ayman & Nasr, Mona. (2012). A Proposed Churn Prediction Model. International Journal of
Engineering Research and Applications (IJERA. 2. 693-697.5488 Journal of Positive School Psychology©2021 JPPW. All rights reserved

[8] M. Hassouna, Agent-Based Modelling, and simulation: An Examination of Customer Retention in the UK Mobile Market. Ph.D.
thesis, Brunel University, UK, 2012.

[9] https://docs.aws.amazon.com/redshift/latest/dg/tutorial-loading-data.html

[10]https://towardsdatascience.com/end-to-end-machine-learning-project-telco-customer-churn-90744a8df97d