December 30 days challenge

```
Project title : AWS Data Integration and Visualization
Your name: Rohit Sahu
Date of creation: 1st Jan 2024
```

```
**Executive Summary:**

The "AWS Data Integration and Visualization" project aims to streamline and
enhance our data processing and visualization capabilities by leveraging key AWS
services. The primary purpose is to centralize, transform, and analyze data from
various sources using AWS, Python, S3 and Amazon Redshift, ultimately providing
actionable insights through Power BI visualization.

**Purpose and Goals:**

The purpose of this project is to improve data management efficiency and derive
valuable insights to support informed decision-making. Key goals include:

1. **Data Integration:** Utilize AWS Glue to seamlessly integrate data from
diverse sources into a unified format for analysis.

2. **Data Transformation:** Leverage Python scripts to perform necessary
transformations on raw data, ensuring consistency and accuracy.

3. **Data Warehousing:** Utilize Amazon Redshift as a powerful data warehouse to
store and organize processed data efficiently.

4. **Visualization:** Implement Power BI for creating interactive and visually
appealing dashboards to present insights to stakeholders.

**Key Features and Benefits:**

*Key Features:*

- Automated Data Integration: AWS Glue facilitates automated and scalable data
integration, reducing manual effort.

- Flexible Data Transformations: Python scripts allow for flexible and customized
data transformations to meet specific business requirements.

- Robust Data Storage: Amazon Redshift provides a high-performance and scalable
data warehousing solution for efficient data storage and retrieval.

- Interactive Dashboards: Power BI enables the creation of dynamic and interactive
dashboards for intuitive data exploration and analysis.

*Benefits:*

- Improved Decision-Making: Centralized and well-organized data enhances the
quality and speed of decision-making processes.

- Time and Cost Efficiency: Automation through AWS Glue reduces manual
intervention, saving time and operational costs.
```

- Scalability: AWS services provide scalability, allowing the system to handle increased data volumes and user demands.

- Enhanced Visualization: Power BI's visualization capabilities offer stakeholders a clear and intuitive understanding of complex data sets.

This project lays the foundation for a robust and efficient data processing and visualization system, empowering our organization with timely and accurate insights for strategic decision-making.

AIM: Understanding the concept and flow of cloud ETL & EDA using tools.
“Non-Programming & Programming”

Technical Tool will include in the this project
AWS S3,
GLUE ETL Process,
Redshift Warehouse,
Python Programming,
Descriptive & ML Programming Tech.

Dashboard Integration

files are stored in the S3 Bucket with the names of Housepricing.csv.
Connect the S3 bucket with Redshift directly and load the data from S3 Repo to the Redshift.
Conn++
Your task is to create a data pipeline that ingests the CSV files from the S3 bucket, transforms them into a suitable format for analysis and machine learning, and loads them into a Redshift cluster that you have created. You should use AWS/other Glue to perform the data ingestion and transformation, and AWS/other S3 to store the intermediate and final data. You should also use AWS/other CloudFormation to automate the creation and configuration of the AWS/other resources that you need for the data pipeline.
    Source Repo —-- Destination Repo
Programming Integration
    Install AWS integration libraries.
    Create a function for Call and download the file from S3 and make it as dataframe (df).
    You are given files and load them to a public S3 bucket that should then contain a CSV files: housepricing.csv.
    Connect Redshift warehouse using python.
    New Dataframe load into the table like bulk loading process into the Redshift using python.
    Connect the redshift cluster again and create a filter dataframe that will do some Descriptive and ML operation.
    You should also apply some machine learning techniques to the Redshift cluster using AWS/other Sagemaker.

“10 Step Only”
Follow 10 Step to do complete project

you have to create a Project folder through python code in S3 location.
Upload the file with the Extension of .csv
Now, again load the files with data frame and merge together and apply data preprocessing steps and find the best KPIs for good fit model data values.
Upload whole data rows in the Redshift cluster in the table now time to share tables details and schema.

```
Now connect Redshift and collect sample data set from the redshift table using P-
SQL query to create a data frame and use it for the ML model.
Apply some checks like Correlation, VIF, KPIs selection methods.
Do some EDA parts over here and write down your understanding of data.
Find the Dependent and Independent KPIs for machine learning algo
Do apply some different-2 machine learning algorithms for best prediction.
```

```
**Project Description:**

The "AWS Data Integration and Visualization" project is a comprehensive initiative
aimed at optimizing our organization's data processing and visualization
capabilities through the effective use of Amazon Web Services (AWS) technologies.
The project addresses the increasing need for centralized, streamlined, and
insightful data management to support data-driven decision-making.

**Background and Context:**

In the era of big data, our organization faces the challenge of managing and
deriving actionable insights from diverse data sources. Siloed data, manual
integration processes, and limited visualization tools hinder our ability to
harness the full potential of our data assets. To overcome these challenges, we
propose the implementation of a robust data integration and visualization solution
leveraging AWS services.

The chosen AWS services include AWS Glue for automated data integration, Python
for flexible data transformations, Amazon Redshift for scalable data warehousing,
and Power BI for creating interactive and visually compelling dashboards. This
combination is selected for its compatibility, scalability, and ability to
seamlessly integrate into our existing infrastructure.

**Objectives:**

1. **Automated Data Integration:** Implement AWS Glue to automate the extraction,
transformation, and loading (ETL) of data from various sources into a centralized
data store.

2. **Flexible Data Transformations:** Utilize Python scripts for data
transformations, ensuring the data is cleaned, standardized, and ready for
analysis.

3. **Scalable Data Warehousing:** Deploy Amazon Redshift as a powerful and
scalable data warehouse solution to store and organize processed data efficiently.

4. **Interactive Data Visualization:** Implement Power BI to create intuitive and
interactive dashboards, providing stakeholders with a user-friendly platform for
exploring and understanding complex datasets.

**Scope:**

The project scope encompasses the following key areas:

- Integration of data from diverse sources, including internal databases, external
APIs, and flat files.
- Implementation of automated ETL processes using AWS Glue to ensure real-time
data updates or python coding.
- Utilization of Python scripts for data transformation and cleaning.
- Deployment of Amazon Redshift for efficient storage and retrieval of processed
data.
```

- Creation of Power BI dashboards for visualization and exploration of data insights.

The project will focus on the integration and visualization of data related to [specific domain or business area], providing immediate benefits to [relevant departments or stakeholders].

This project is expected to deliver a scalable, efficient, and user-friendly data processing and visualization system, setting the foundation for informed decision-making across our organization.

Project Scope:

Defining the project scope is crucial to set clear boundaries and expectations for the "AWS Data Integration and Visualization" project.

Inclusions:

Data Integration: The project will include the integration of data from diverse sources, such as internal databases, external APIs, and flat files.
Automated ETL Processes: Implementation of automated ETL processes using AWS Glue to ensure real-time data updates and maintain data accuracy but at the same time we are doing with python code as well becuase in the real time processing Glue is costly.
Data Transformation: Utilization of Python scripts for data transformation and cleaning, ensuring standardized and reliable data for analysis.
Data Warehousing: Deployment of Amazon Redshift for efficient storage and retrieval of processed data, providing a scalable and high-performance data warehousing solution.
Visualization Dashboards: Creation of Power BI dashboards for interactive and intuitive data visualization, allowing stakeholders to explore and understand complex datasets.

Deliverables:
Python Transformation Scripts: Well-documented Python scripts for data transformation and cleaning, ensuring transparency and ease of future modifications.
Amazon Redshift Data Warehouse: Deployment of Amazon Redshift, including the schema design and optimization for efficient storage and retrieval.
Power BI Dashboards: Creation of interactive Power BI dashboards, allowing stakeholders to visualize and explore key insights from integrated data.
Technical Documentation: Comprehensive technical documentation covering the setup, configuration, and maintenance aspects of the integrated system.

Milestones:

Project Kickoff (Week 1): Project initiation, Aim and channel flow, Step to do all the activity without any cost pay.

Data Integration (Weeks 2): Implementation of AWS Redshift Cluster, S3 container, PowerBI tool and install all the libraies
which will help to connect all and other tool as well for complete the data flow.
: Development and testing of Python scripts for data transformation and cleaning.
: Deployment of Amazon Redshift for scalable data warehousing and S3 container.

```
                                : Deploy the Python intregration code for all
import/Export from S3,Redshift.

Machine Learning implementation (Weeks 3): Deploy the code for house sale price
prediction using N number of KPIs (Sample data downloaded from the Kaggle).

Power BI Implementation (Weeks 4): Creation and testing of Power BI dashboards for
data visualization.
                                         : Compilation of technical documentation and Flow
of Tools
```

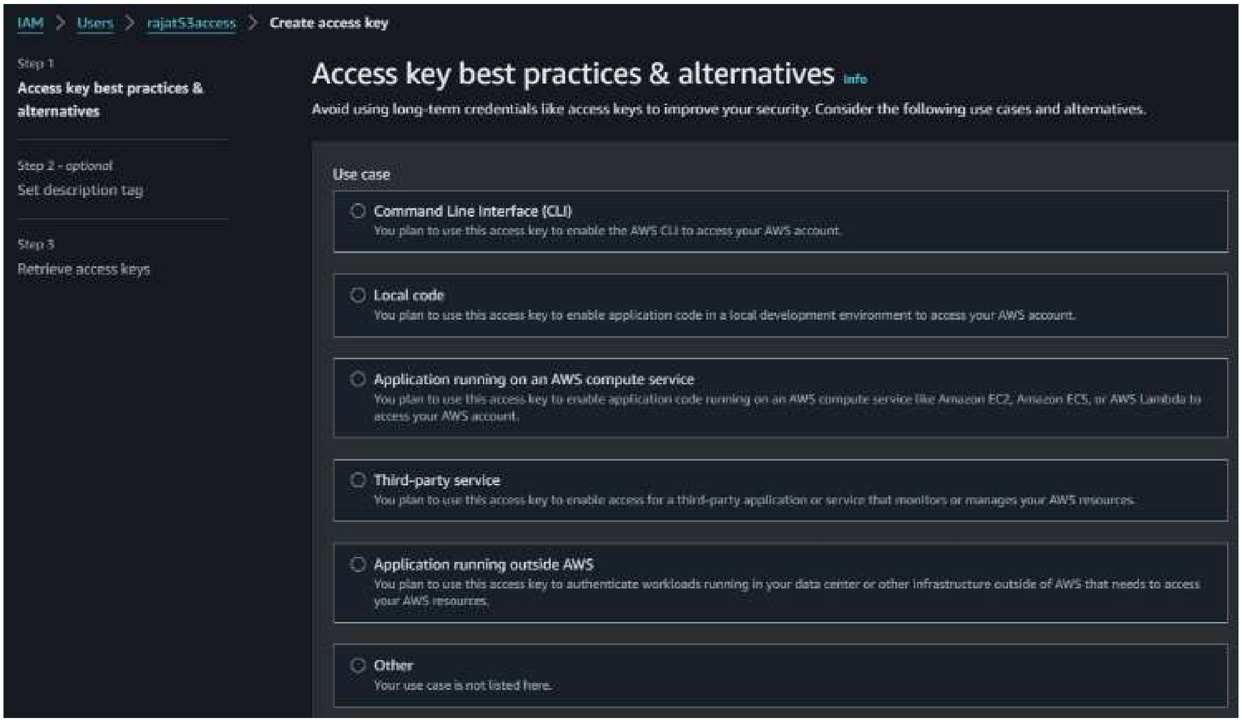
AIM: Understanding the concept and flow of cloud ETL & EDA using tools.

<u>"Non-Programming & Programming"</u>

Technical Tool will include in the this project

1. AWS S3,
2. GLUE ETL Process,
3. Redshift Warehouse,
4. Python Programming,
5. Descriptive & ML Programming Tech.

How to know about the AccessKey and Secretkey of the Aws credentials.



In [1]:
```
## Install the Libraries for intregrate the usefull libraries
import boto3
from boto3.session import Session
import pandas as pd
from io import StringIO, BytesIO
import psycopg2
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as plt
import seaborn as sns
```

In [2]:
```
# Way to ignore all the warnning, when we will process code and installed the librar
import warnings
warnings.filterwarnings('ignore')
```

S3 Connectivity and Integration

In [10]:
```
#Create S3 Session using access key and secret key for connect the S3 container using
session = Session(aws_access_key_id='Your_access_key_id',aws_secret_access_key='Your_
s3= session.client('s3')
```

In [11]:
```
# Total numbers of buckets present in S3 container.
s3.list_buckets()['Buckets']
```

Out[11]:
```
[{'Name': 'aws-rajat-project',
  'CreationDate': datetime.datetime(2023, 11, 26, 15, 23, 36, tzinfo=tzutc())}]
```

In [13]:
```
# Connect S3 and import raw data and dataframe visual.
response = s3.get_object(Bucket='aws-rajat-project', Key='source/HousePricetrainning
content = response['Body'].read()
df = pd.read_csv(io.BytesIO(content))
df.head()
```

Out[13]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...

5 rows × 81 columns

In [14]:
```
#Total number of columns in raw data.
df.columns
```

Out[14]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [15]:
```
# Drop Id column
df = df.drop(columns=['Id'], axis=1)
```

In [16]:
```
#Find the total number of unique values in each column. It is a way of finding error
for i in list(df.columns):
    print("###")
    print("Value Count of the Columns : ",i)
    print(df[i].value_counts(normalize=True)*100)
```

```
NWAmes     5.000000
SawyerW    4.041096
BrkSide    3.972603
Crawfor    3.493151
Mitchel    3.356164
NoRidge    2.808219
Timber     2.602740
IDOTRR     2.534247
ClearCr    1.917808
StoneBr    1.712329
SWISU      1.712329
MeadowV    1.164384
Blmngtn    1.164384
BrDale     1.095890
Veenker    0.753425
NPkVill    0.616438
Blueste    0.136986
Name: proportion, dtype: float64
###
```

DataFrame Column selection

In [17]:
```
# Selection data columns for the data model.
df = df[['LotArea','Street','LotShape',
         'HouseStyle','YearBuilt','YearRemodAdd','RoofStyle','RoofMatl','Foundation'
        'CentralAir','Electrical','Functional','Fireplaces','YrSold','SaleType','Sal
```

In [18]:
```
# Final Testing dataframe, which will help to identify the sale value !
df.head(2)
```

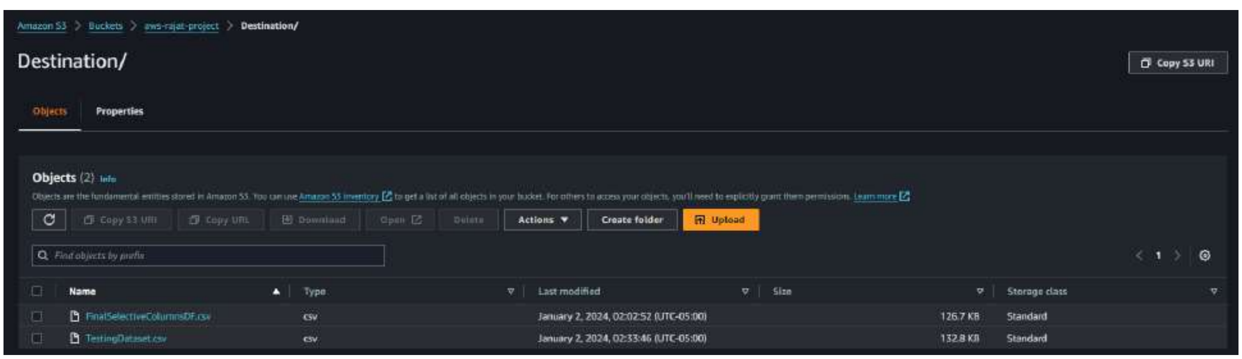
Out[18]:

	LotArea	Street	LotShape	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Foundation
0	8450	Pave	Reg	2Story	2003	2003	Gable	CompShg	PConc
1	9600	Pave	Reg	1Story	1976	1976	Gable	CompShg	CBlock

In [19]:
```
# To save for replications only
df.to_csv(r'C:/Users/srajat/Desktop/TestingDataset.csv')
```

In [20]:
```python
# Check all the datatypes of the columns.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   LotArea        1460 non-null   int64
 1   Street         1460 non-null   object
 2   LotShape       1460 non-null   object
 3   HouseStyle     1460 non-null   object
 4   YearBuilt      1460 non-null   int64
 5   YearRemodAdd   1460 non-null   int64
 6   RoofStyle      1460 non-null   object
 7   RoofMatl       1460 non-null   object
 8   Foundation     1460 non-null   object
 9   CentralAir     1460 non-null   object
 10  Electrical     1459 non-null   object
 11  Functional     1460 non-null   object
 12  Fireplaces     1460 non-null   int64
 13  YrSold         1460 non-null   int64
 14  SaleType       1460 non-null   object
 15  SaleCondition  1460 non-null   object
 16  SalePrice      1460 non-null   int64
dtypes: int64(6), object(11)
memory usage: 194.0+ KB
```



In [21]:
```python
#Upload Final File Structure in the S3 for future perpose uses !!
csv_buffer = StringIO()
df.to_csv(csv_buffer, index=False)

bucket_name = 'aws-rajat-project'
file_key = 'Destination/FinalSelectiveColumnsDF.csv'

s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=file_key)
```

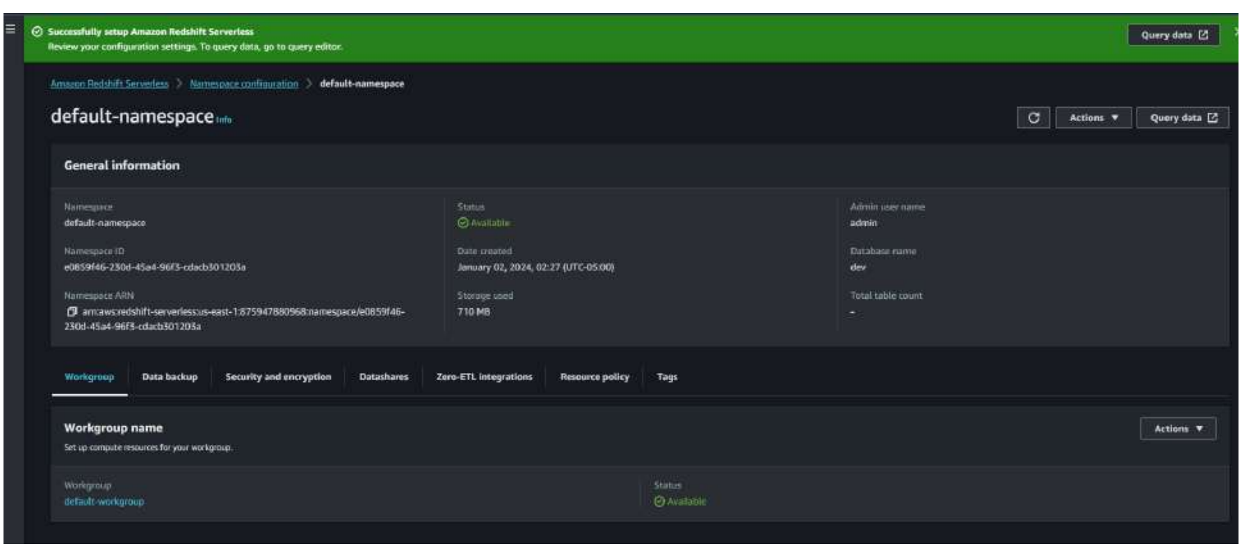
Out[21]:

```
{'ResponseMetadata': {'RequestId': '8P8ZAHGHZ4S94C7E',
  'HostId': 'hkq+Bk9KzHX0wxOHyWU+wpHFwW7MBf4qoG6taO/OfhjqCn/S3ZAW8wziX3CfqliL8NNenx
KFmPo=',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amz-id-2': 'hkq+Bk9KzHX0wxOHyWU+wpHFwW7MBf4qoG6taO/OfhjqCn/S3Z
AW8wziX3CfqliL8NNenxKFmPo=',
   'x-amz-request-id': '8P8ZAHGHZ4S94C7E',
   'date': 'Tue, 02 Jan 2024 08:25:35 GMT',
   'x-amz-server-side-encryption': 'AES256',
   'etag': '"4aca0d2d841b6683e55b40cb28c58e0e"',
   'server': 'AmazonS3',
   'content-length': '0'},
  'RetryAttempts': 0},
 'ETag': '"4aca0d2d841b6683e55b40cb28c58e0e"',
 'ServerSideEncryption': 'AES256'}
```


Data Upload Characterstics

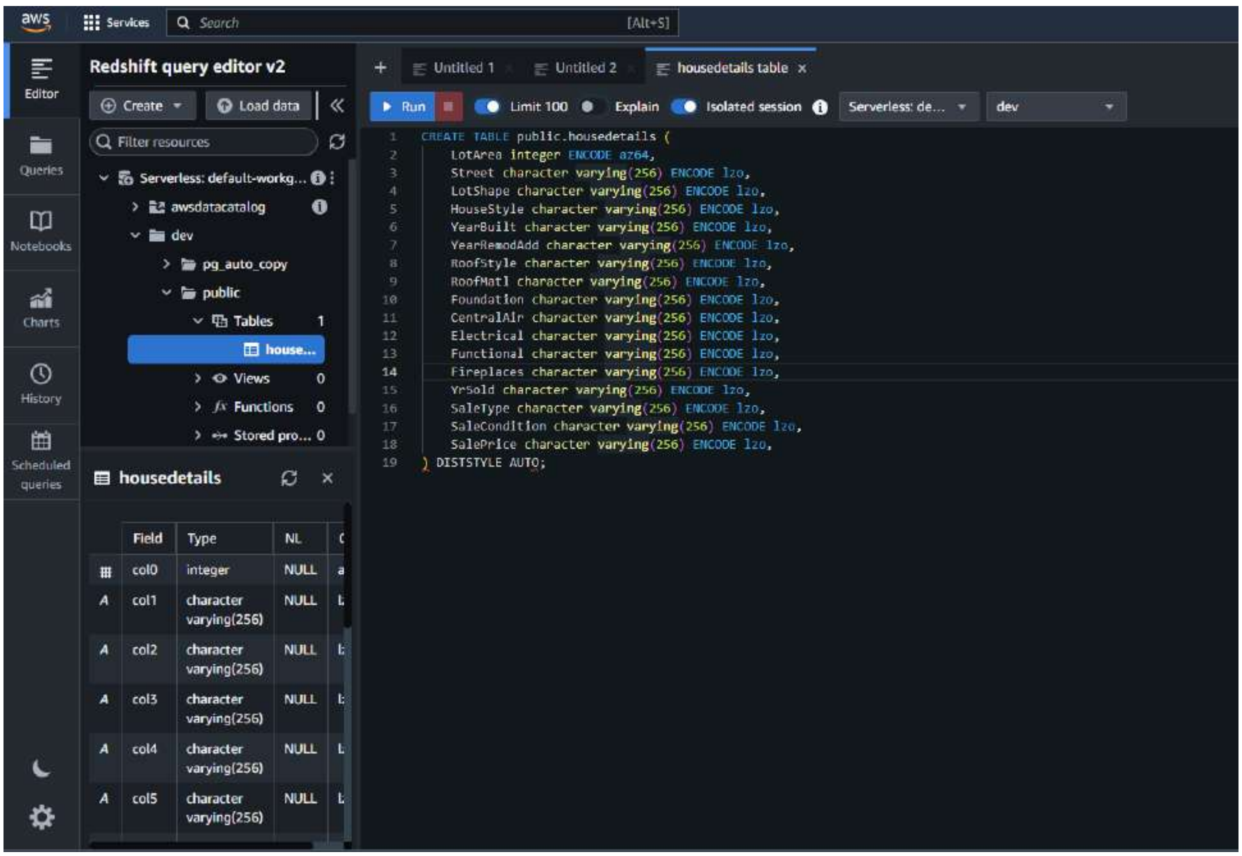
Serverless: awsprojectworkbook Database: dev Schema: public Table: HouseDetails IAM role: arn:aws:iam::875947880968:role/service-role/AmazonRedshift-CommandsAccessRole-20231201T133454

#Command for upload file from the s3 to redshift table direct """COPY dev.public.HouseDetails FROM 's3://aws-rajat-project/Destination/FinalSelectiveColumnsDF.csv' IAM_ROLE 'arn:aws:iam::875947880968:role/service-role/AmazonRedshift-CommandsAccessRole-20231201T133454' FORMAT AS CSV DELIMITER ',' QUOTE '"' IGNOREHEADER 1 REGION AS 'ap-south-1'"""



Establish a connection in redshift cluster

```
conn = psycopg2.connect(
    dbname='dev',
    user='********',
    password='**************',

host='***********************************************************.amazonaws.com',
    port='5439'
)
cursor = conn.cursor()
```




```
create="""CREATE TABLE public.housedetails (
    LotArea integer ENCODE az64,
    Street character varying(256) ENCODE lzo,
    LotShape character varying(256) ENCODE lzo,
    HouseStyle character varying(256) ENCODE lzo,
    YearBuilt character varying(256) ENCODE lzo,
    YearRemodAdd character varying(256) ENCODE lzo,
    RoofStyle character varying(256) ENCODE lzo,
    RoofMatl character varying(256) ENCODE lzo,
    Foundation character varying(256) ENCODE lzo,
    CentralAir character varying(256) ENCODE lzo,
    Electrical character varying(256) ENCODE lzo,
    Functional character varying(256) ENCODE lzo,
```

```
    Fireplaces character varying(256) ENCODE lzo,
    YrSold character varying(256) ENCODE lzo,
    SaleType character varying(256) ENCODE lzo,
    SaleCondition character varying(256) ENCODE lzo,
    SalePrice character varying(256) ENCODE lzo,
) DISTSTYLE AUTO;"""
cursor.execute(create)
```

In [22]:

```
# DataFrame columns.
df.columns
```

Out[22]:

```
Index(['LotArea', 'Street', 'LotShape', 'HouseStyle', 'YearBuilt',
       'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Foundation', 'CentralAir',
       'Electrical', 'Functional', 'Fireplaces', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

```
# Way to find the, how many sample data base present in the cluster
cursor.execute('SELECT current_database();')
database_name = cursor.fetchone()[0]
database_name
```

In [23]:

```
# Connect Final Dataframe from the Destination container for uploading process in Red
response = s3.get_object(Bucket='aws-rajat-project', Key='Destination/FinalSelective
response
content = response['Body'].read()
ImportFromS3DF = pd.read_csv(io.BytesIO(content))
ImportFromS3DF.head()
```

Out[23]:

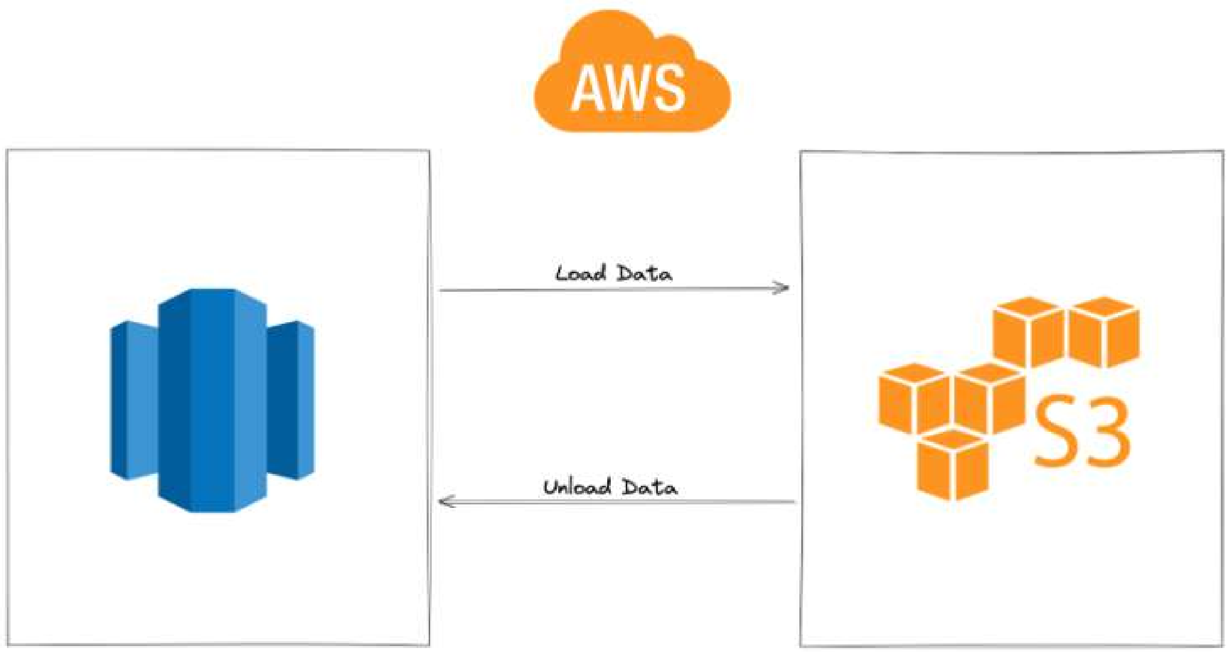
	LotArea	Street	LotShape	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Foundation
0	8450	Pave	Reg	2Story	2003	2003	Gable	CompShg	PConc
1	9600	Pave	Reg	1Story	1976	1976	Gable	CompShg	CBlock
2	11250	Pave	IR1	2Story	2001	2002	Gable	CompShg	PConc
3	9550	Pave	IR1	2Story	1915	1970	Gable	CompShg	BrkTil
4	14260	Pave	IR1	2Story	2000	2000	Gable	CompShg	PConc

In [24]:

```
# Show for the samplling.
ImportFromS3DF.head(2)
```

Out[24]:

	LotArea	Street	LotShape	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Foundation
0	8450	Pave	Reg	2Story	2003	2003	Gable	CompShg	PConc
1	9600	Pave	Reg	1Story	1976	1976	Gable	CompShg	CBlock

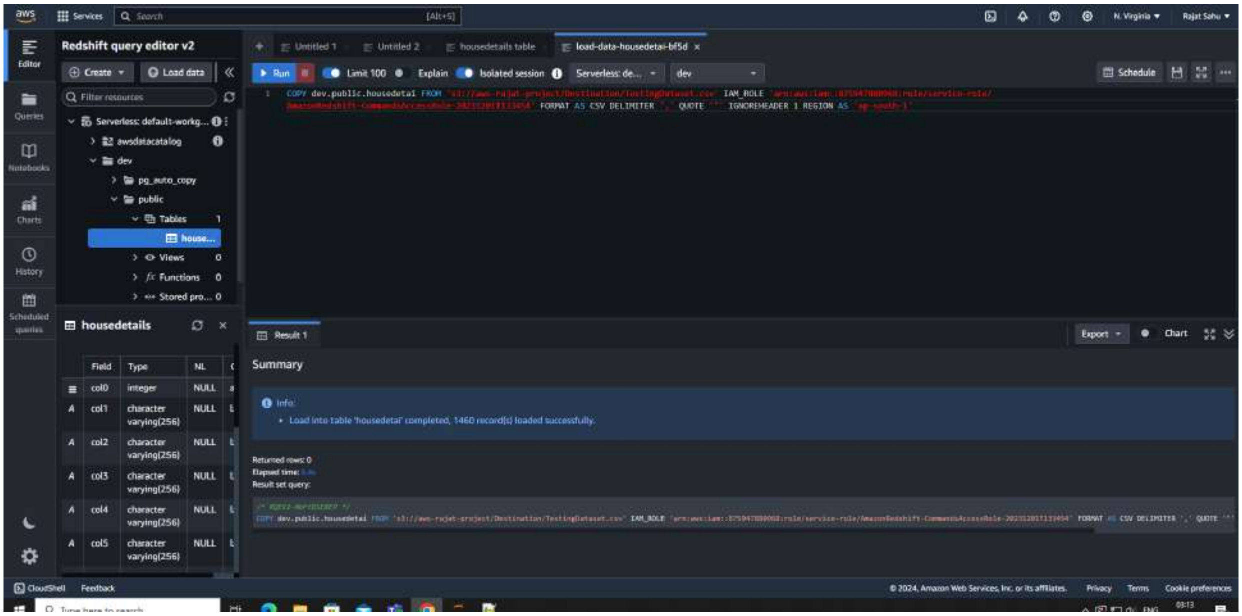



```
# Establish a connection and upload all the in Redshift cluster.
conn = psycopg2.connect(
    dbname='dev',
    user='******',
    password='***********',
    host='******************************************************.amazonaws.com',
    port='5439'
)
cursor = conn.cursor()
for i in range(0,len(ImportFromS3DF)):
    LotArea,Street,LotShape,HouseStyle,YearBuilt=df.loc[i][0],df.loc[i]
[1],df.loc[i][2],df.loc[i][3],df.loc[i][4]
    YearRemodAdd,RoofStyle,RoofMatl,Foundation,CentralAir=df.loc[i][5],df.loc[i]
[6],df.loc[i][7],df.loc[i][8],df.loc[i][9]
    Electrical,Functional,Fireplaces,YrSold,SaleType=df.loc[i][10],df.loc[i]
[11],df.loc[i][12],df.loc[i][13],df.loc[i][14]
    SaleCondition, SalePrice=df.loc[i][15],df.loc[i][16]
    print("INSERT INTO dev.public.housedetails VALUES
("+str(LotArea)+",'"+str(Street)+"','"+str(LotShape)+"','"+str(HouseStyle)+"',"+st
r(YearBuilt)+","+str(YearRemodAdd)+",'"+str(RoofStyle)+"','"+str(RoofMatl)+"','"+s
tr(Foundation)+"','"+str(CentralAir)+"','"+str(Electrical)+"','"+str(Functional)+"
',"+str(Fireplaces)+","+str(YrSold)+",'"+str(SaleType)+"','"+str(SaleCondition)+"'
,"+str(SalePrice)+")")
    cursor.execute("INSERT INTO dev.public.housedetails VALUES
("+str(LotArea)+",'"+str(Street)+"','"+str(LotShape)+"','"+str(HouseStyle)+"',"+st
r(YearBuilt)+","+str(YearRemodAdd)+",'"+str(RoofStyle)+"','"+str(RoofMatl)+"','"+s
tr(Foundation)+"','"+str(CentralAir)+"','"+str(Electrical)+"','"+str(Functional)+"
',"+str(Fireplaces)+","+str(YrSold)+",'"+str(SaleType)+"','"+str(SaleCondition)+"'
,"+str(SalePrice)+")")
    conn.commit()
```

```
Amazon Redshift provides multiple ways to load data into a Redshift cluster to
accommodate different use cases and preferences. Here are some common methods:
```

```
Amazon Redshift COPY Command:
The COPY command is one of the most efficient ways to load large amounts of data
into Redshift from various data sources, including Amazon S3, Amazon DynamoDB, or
other Redshift clusters.

**COPY table_name FROM 's3://your-s3-bucket/your-data-prefix'
CREDENTIALS 'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-
access-key>'
CSV;**

Amazon Redshift Spectrum:
Redshift Spectrum allows you to query data stored in Amazon S3 directly without
loading it into Redshift tables. This is useful for analyzing large datasets
without the need for data movement.

**CREATE EXTERNAL SCHEMA spectrum_schema
FROM DATA CATALOG DATABASE 'your-database-name'
IAM_ROLE 'arn:aws:iam::your-account-id:role/your-Redshift-role'
CREATE EXTERNAL DATABASE IF NOT EXISTS;
**

SQL INSERT Statements ( in this project we will flow this one.)
You can use SQL INSERT statements to insert data into Redshift tables. This is
suitable for smaller datasets or when you need more control over the insertion
process.

**INSERT INTO target_table (column1, column2, ...)
SELECT column1, column2, ...
FROM source_table;
**

ETL Tools (e.g., AWS Glue, Apache Spark):
ETL (Extract, Transform, Load) tools like AWS Glue or Apache Spark can be used to
prepare and load data into Redshift. These tools provide a graphical interface for
designing ETL workflows.
Example: Create an ETL job in AWS Glue to transform and load data into Redshift.
```

2nd Way to load bulk amount of data from S3




```
import psycopg2

# Establish a connection
conn = psycopg2.connect(
    dbname='dev',
    user='*****',
    password='***********',
    host='************************************************.amazonaws.com',
    port='5439'
)

cursor = conn.cursor()

# Specify the values in the VALUES statement
values = "
(8452123,'Pave','Reg','2Story',2003,2003,'Gable','CompShg','PConc','Y','SBrkr','Typ',0,2008,'WD','Normal',208500)"
# Replace the values with your actual values

# Construct the full INSERT statement
insert_statement = f"INSERT INTO dev.public.housedetails VALUES {values}"

# Execute the INSERT statement
cursor.execute(insert_statement)

# Commit the transaction
conn.commit()

# Close the cursor and connection
cursor.close()
conn.close()
```

In [25]:
```
#Object Columns selection for the lable the data.
object_columns = df.select_dtypes(include=['object'])
columns_to_encode=list(object_columns.columns)
label_encoder = LabelEncoder()
for col in columns_to_encode:
    if col in df.columns:
        df[col + '_encoded'] = label_encoder.fit_transform(df[col])
```

In [26]:
```
df.columns
```

Out[26]:
```
Index(['LotArea', 'Street', 'LotShape', 'HouseStyle', 'YearBuilt',
       'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Foundation', 'CentralAir',
       'Electrical', 'Functional', 'Fireplaces', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice', 'Street_encoded', 'LotShape_encoded',
       'HouseStyle_encoded', 'RoofStyle_encoded', 'RoofMatl_encoded',
       'Foundation_encoded', 'CentralAir_encoded', 'Electrical_encoded',
       'Functional_encoded', 'SaleType_encoded', 'SaleCondition_encoded'],
      dtype='object')
```

#Upload Final File Structure in the S3 for future perpose uses !!

from io import StringIO csv_buffer = StringIO() df.to_csv(csv_buffer, index=False)

bucket_name = 'aws-rajat-project' file_key = 'Destination/FinalSelectiveColumnsDF.csv'

s3.put_object(Body=csv_buffer.getvalue(), Bucket=bucket_name, Key=file_key)

In [27]:
```
df.head()
```

Out[27]:

	LotArea	Street	LotShape	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Foundation
0	8450	Pave	Reg	2Story	2003	2003	Gable	CompShg	PConc
1	9600	Pave	Reg	1Story	1976	1976	Gable	CompShg	CBlock
2	11250	Pave	IR1	2Story	2001	2002	Gable	CompShg	PConc
3	9550	Pave	IR1	2Story	1915	1970	Gable	CompShg	BrkTil
4	14260	Pave	IR1	2Story	2000	2000	Gable	CompShg	PConc

5 rows × 28 columns

In [28]:
```
#Only replacement column position.
FinalSelectionColumns = df.select_dtypes(include=['int64','int32'])
FinalSelectionColumns.head(2)
```

Out[28]:

	LotArea	YearBuilt	YearRemodAdd	Fireplaces	YrSold	SalePrice	Street_encoded	LotShape_encoded
0	8450	2003	2003	0	2008	208500	1	3
1	9600	1976	1976	1	2007	181500	1	3

In [29]:
```
FinalSelectionColumns.columns
```

Out[29]:
```
Index(['LotArea', 'YearBuilt', 'YearRemodAdd', 'Fireplaces', 'YrSold',
       'SalePrice', 'Street_encoded', 'LotShape_encoded', 'HouseStyle_encoded',
       'RoofStyle_encoded', 'RoofMatl_encoded', 'Foundation_encoded',
       'CentralAir_encoded', 'Electrical_encoded', 'Functional_encoded',
       'SaleType_encoded', 'SaleCondition_encoded'],
      dtype='object')
```

In [30]:
```
FinalSelectionColumns = FinalSelectionColumns[['LotArea', 'YearBuilt', 'YearRemodAdd
       'Street_encoded', 'LotShape_encoded', 'HouseStyle_encoded',
       'RoofStyle_encoded', 'RoofMatl_encoded', 'Foundation_encoded',
       'CentralAir_encoded', 'Electrical_encoded', 'Functional_encoded',
       'SaleType_encoded', 'SaleCondition_encoded','SalePrice']]
FinalSelectionColumns.head(2)
```

Out[30]:

	LotArea	YearBuilt	YearRemodAdd	Fireplaces	YrSold	Street_encoded	LotShape_encoded	HouseSty
0	8450	2003	2003	0	2008	1	3	
1	9600	1976	1976	1	2007	1	3	

In [31]:
```
FinalSelectionColumns.columns
```

Out[31]:
```
Index(['LotArea', 'YearBuilt', 'YearRemodAdd', 'Fireplaces', 'YrSold',
       'Street_encoded', 'LotShape_encoded', 'HouseStyle_encoded',
       'RoofStyle_encoded', 'RoofMatl_encoded', 'Foundation_encoded',
       'CentralAir_encoded', 'Electrical_encoded', 'Functional_encoded',
       'SaleType_encoded', 'SaleCondition_encoded', 'SalePrice'],
      dtype='object')
```

In [32]:
```
# Check final all dataTypes
FinalSelectionColumns.info()
```

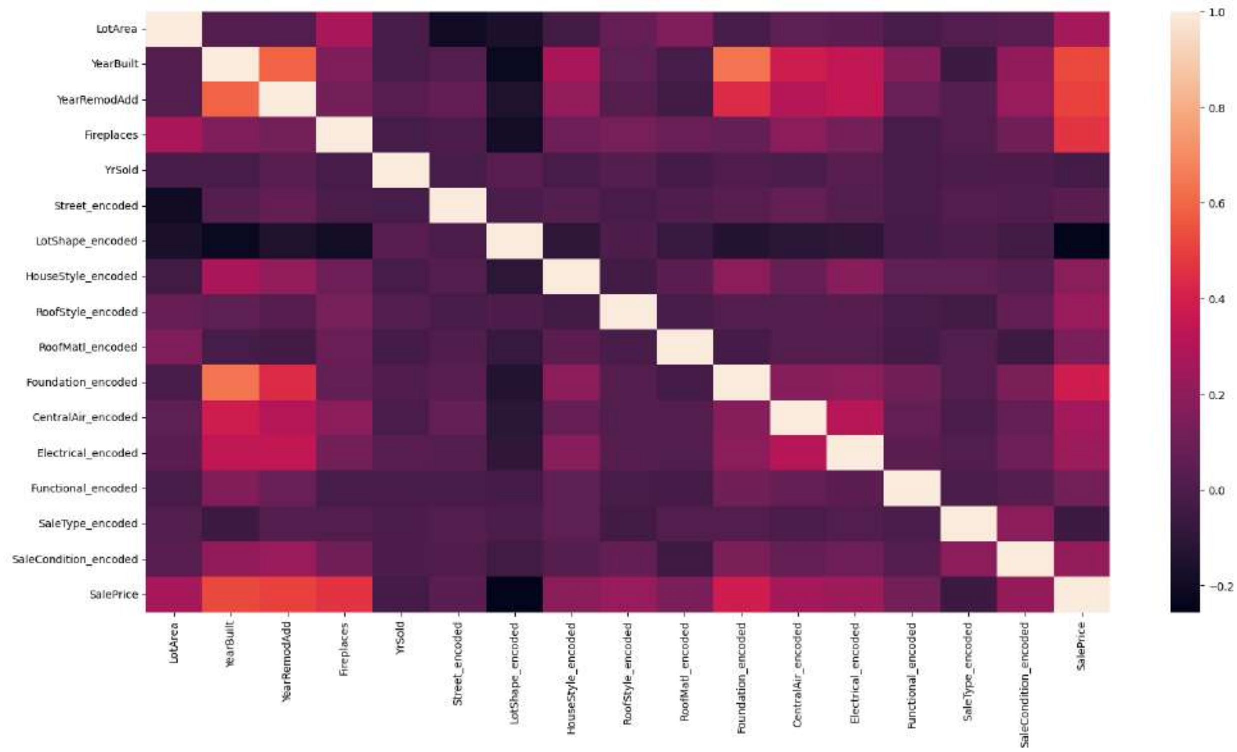
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   LotArea               1460 non-null   int64
 1   YearBuilt             1460 non-null   int64
 2   YearRemodAdd          1460 non-null   int64
 3   Fireplaces            1460 non-null   int64
 4   YrSold                1460 non-null   int64
 5   Street_encoded        1460 non-null   int32
 6   LotShape_encoded      1460 non-null   int32
 7   HouseStyle_encoded    1460 non-null   int32
 8   RoofStyle_encoded     1460 non-null   int32
 9   RoofMatl_encoded      1460 non-null   int32
 10  Foundation_encoded    1460 non-null   int32
 11  CentralAir_encoded    1460 non-null   int32
 12  Electrical_encoded    1460 non-null   int32
 13  Functional_encoded    1460 non-null   int32
```

In [33]:
```
correlation_matrix = FinalSelectionColumns.corr()
threshold = 0.75
high_correlation_pairs = (correlation_matrix.abs() > threshold) & (correlation_matri
high_correlation_features = []
for col in high_correlation_pairs.columns:
    correlated_cols = high_correlation_pairs.index[high_correlation_pairs[col]].toli
    for correlated_col in correlated_cols:
        high_correlation_features.append((col, correlated_col))
```

In [34]:
```
### High Correlation metrics found
high_correlation_features
```

Out[34]:
```
[]
```

In [36]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming FinalSelectionColumns is your DataFrame
correlation_matrix = FinalSelectionColumns.corr()

plt.figure(figsize=(20, 10))
sns.heatmap(correlation_matrix)
plt.show()
```



In [37]:
```python
FinalSelectionColumns.isnull().sum()
```

Out[37]:
```
LotArea                0
YearBuilt              0
YearRemodAdd           0
Fireplaces             0
YrSold                 0
Street_encoded         0
LotShape_encoded       0
HouseStyle_encoded     0
RoofStyle_encoded      0
RoofMatl_encoded       0
Foundation_encoded     0
CentralAir_encoded     0
Electrical_encoded     0
Functional_encoded     0
SaleType_encoded       0
SaleCondition_encoded  0
SalePrice              0
dtype: int64
```

In [38]:
```
# Training Data Fetch from the Redshift Cluster, Now we will go with this data frame
cursor.execute('select LotArea, YearBuilt, YearRemodAdd, Fireplaces, YrSold, Street_
filterdata = cursor.fetchall()
column = [desc[0] for desc in cursor.description]
mlDataFrame=pd.DataFrame(filterdata,columns=column)

# Close the cursor and connection
cursor.close()
conn.close()
```

...

In [ ]:

In [40]:
```
mlDataFrame=pd.read_csv(r"C:/Users/srajat/Desktop/TestingDataset.csv")
```

In [41]:
```
# AS WE ARE SEEING OUR DATA HAVE CATEGORIAL DATA SO WE HAVE TO CONVERT THE DATA INTO
from sklearn.preprocessing import LabelEncoder
list1=[item for item in mlDataFrame.columns if mlDataFrame[item].dtypes=='object']
le=LabelEncoder()
for i in list1:
    df[i]=le.fit_transform(df[i])
```

In [42]:
```
# now our new data after label encoding
mlDataFrame.head()
```

Out[42]:

	Unnamed: 0	LotArea	Street	LotShape	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	F
0	0	8450	Pave	Reg	2Story	2003	2003	Gable	CompShg	
1	1	9600	Pave	Reg	1Story	1976	1976	Gable	CompShg	
2	2	11250	Pave	IR1	2Story	2001	2002	Gable	CompShg	
3	3	9550	Pave	IR1	2Story	1915	1970	Gable	CompShg	
4	4	14260	Pave	IR1	2Story	2000	2000	Gable	CompShg	

In [43]:
```
mlDataFrame.info()
column_to_drop = 'Unnamed: 0'
mlDataFrame = mlDataFrame.drop(columns=column_to_drop)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Unnamed: 0     1460 non-null   int64 
 1   LotArea        1460 non-null   int64 
 2   Street         1460 non-null   object
 3   LotShape       1460 non-null   object
 4   HouseStyle     1460 non-null   object
 5   YearBuilt      1460 non-null   int64 
 6   YearRemodAdd   1460 non-null   int64 
 7   RoofStyle      1460 non-null   object
 8   RoofMatl       1460 non-null   object
 9   Foundation     1460 non-null   object
 10  CentralAir     1460 non-null   object
 11  Electrical     1459 non-null   object
 12  Functional     1460 non-null   object
 13  Fireplaces     1460 non-null   int64 
 14  YrSold         1460 non-null   int64 
 15  SaleType       1460 non-null   object
 16  SaleCondition  1460 non-null   object
 17  SalePrice      1460 non-null   int64 
dtypes: int64(7), object(11)
memory usage: 205.4+ KB
```

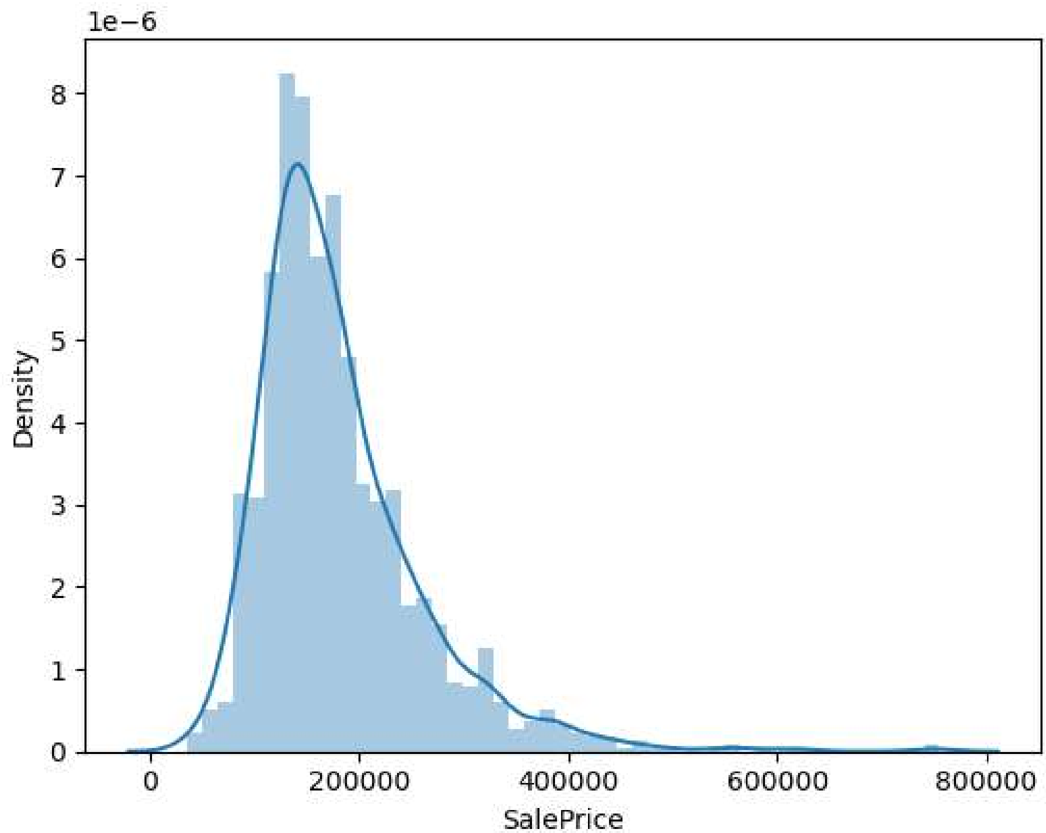
In [44]:
```
mlDataFrame.columns
```

Out[44]:
```
Index(['LotArea', 'Street', 'LotShape', 'HouseStyle', 'YearBuilt',
       'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Foundation', 'CentralAir',
       'Electrical', 'Functional', 'Fireplaces', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [45]:
```
print(len(mlDataFrame.columns))
```

```
17
```

```
<Axes: xlabel='SalePrice', ylabel='Density'>
```



In [55]:
```
from sklearn.preprocessing import LabelEncoder
list1=[item for item in mlDataFrame.columns if mlDataFrame[item].dtypes=='object']
le=LabelEncoder()
for i in list1:
    mlDataFrame[i]=le.fit_transform(df[i])
```

In [56]:
```
#IN THIS WE ARE SEPERATING THE DATA FRAME BY DROPPING THE TARGET FEATURE
X=mlDataFrame.drop('SalePrice',axis=1)
Y=mlDataFrame['SalePrice']
print(X.shape)
print(Y.shape)
```

```
(1460, 16)
(1460,)
```

In [57]:
```
# IH THIS SHELL WE ARE DOING FEATURE SCALING AND CONVERTING THE RANGE OF [-1,1] WITH
from sklearn.preprocessing import StandardScaler
SS=StandardScaler()
SS.fit_transform(X,Y)
```

Out[57]:
```
array([[-0.20714171,  0.06423821,  0.75073056, ...,  0.13877749,
         0.31386709,  0.2085023 ],
       [-0.09188637,  0.06423821,  0.75073056, ..., -0.61443862,
         0.31386709,  0.2085023 ],
       [ 0.07347998,  0.06423821, -1.37893255, ...,  0.13877749,
         0.31386709,  0.2085023 ],
       ...,
       [-0.14781027,  0.06423821,  0.75073056, ...,  1.64520971,
         0.31386709,  0.2085023 ],
       [-0.08016039,  0.06423821,  0.75073056, ...,  1.64520971,
         0.31386709,  0.2085023 ],
       [-0.05811155,  0.06423821,  0.75073056, ...,  0.13877749,
         0.31386709,  0.2085023 ]])
```

In [58]:
```
#IN THIS WE ARE PREPARING THE DATA INTO TWO FORM TRAIN AND TEST. TRAIN IS FOR TRAINI
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size=0.7,random_state=31)
print(X_train.shape)
print(X_test.shape)
print(Y_test.shape)
print(Y_train.shape)
```

```
(1021, 16)
(439, 16)
(439,)
(1021,)
```

In [59]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, Y_train)
y_pred = lr.predict(X_test)
print(f"Accuracy of training set:", lr.score(X_train, Y_train))
print(f"Accuracy of testing set: ", lr.score(X_test, Y_test))
A=lr.score(X_test, Y_test)
```

```
Accuracy of training set: 0.5502607314072425
Accuracy of testing set:  0.5633306106305673
```

In [60]:
```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, Y_train)
print(f"Accuracy of training set:", reg_rf.score(X_train, Y_train))
print(f"Accuracy of testing set: ", reg_rf.score(X_test, Y_test))
B=reg_rf.score(X_test, Y_test)
```

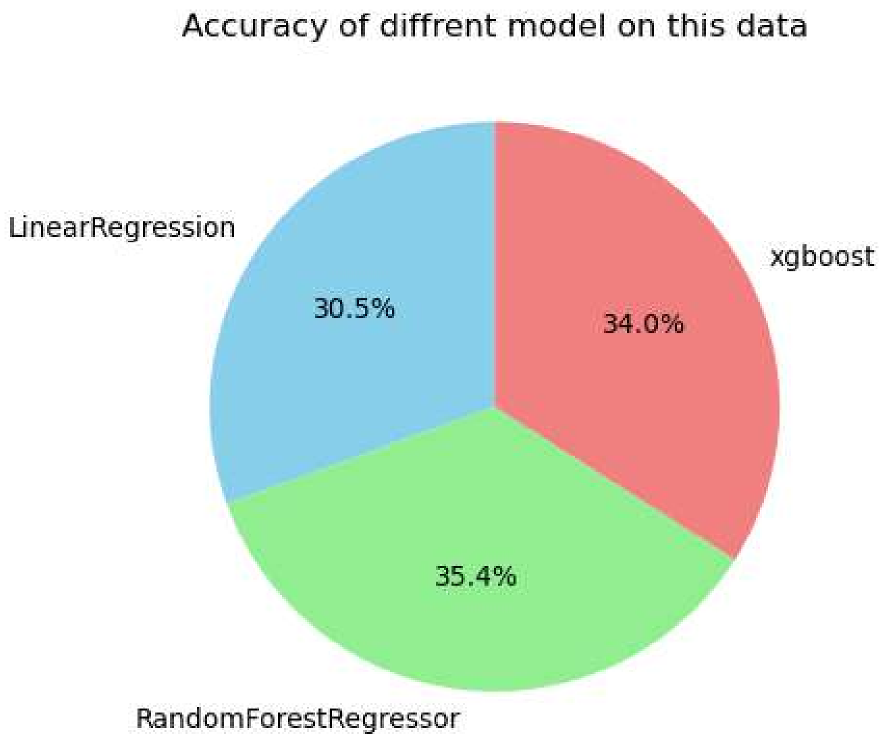
```
Accuracy of training set: 0.9542022449303872
Accuracy of testing set:  0.6533837634034039
```

In [61]:
```
import xgboost as xgb
# Create an XGBoost regressor
model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
model.fit(X_train, Y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

print(f"Accuracy of training set:", model.score(X_train, Y_train))
print(f"Accuracy of testing set: ", model.score(X_test, Y_test))
C=model.score(X_test, Y_test)
```

```
Accuracy of training set: 0.9929453027362252
Accuracy of testing set:  0.6273388705116633
```

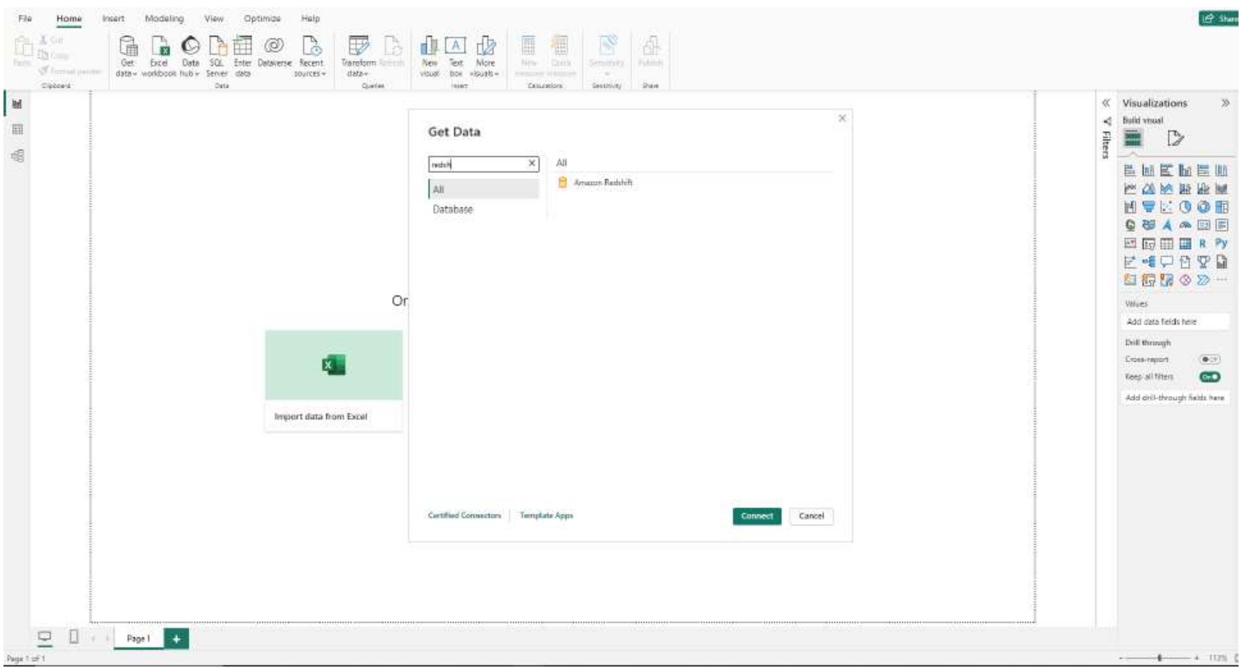
In [ ]:
```
!pip install xgboost
```

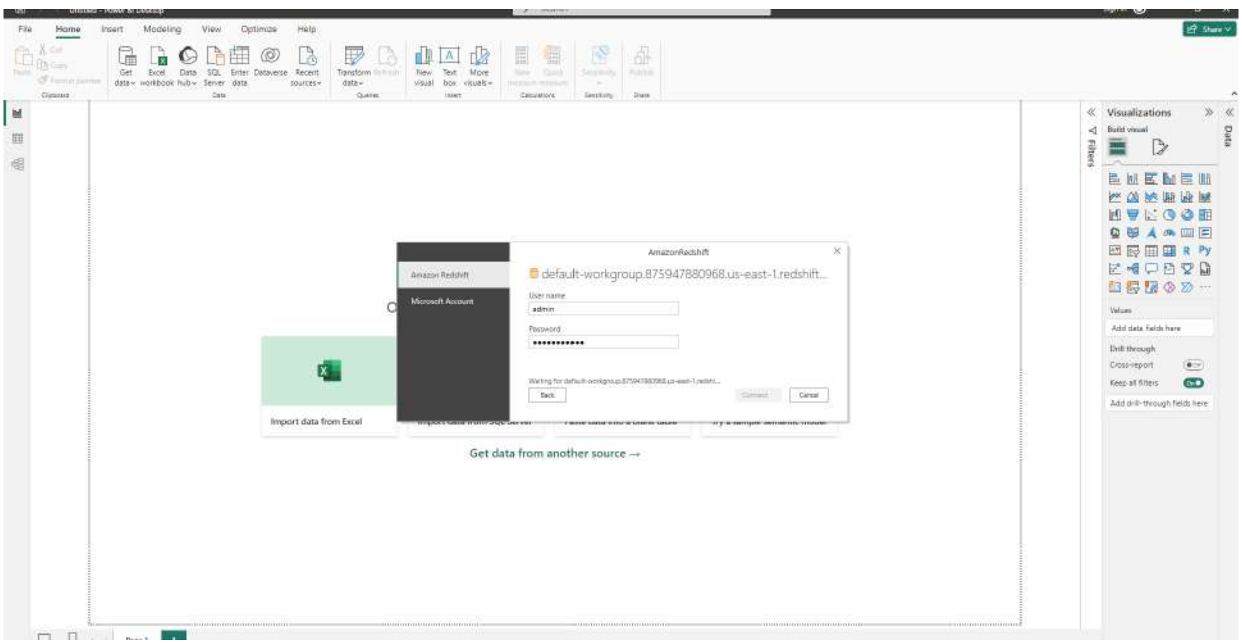
In [62]:
```
import matplotlib.pyplot as plt

labels = ['LinearRegression', 'RandomForestRegressor', 'xgboost']
values = [A,B,C]

# Create a pie chart
plt.pie(values, labels=labels, autopct='%1.1f%%', startangle=90, colors=['skyblue',

# Add a title
plt.title('Accuracy of diffrent model on this data')

# Display the pie chart
plt.show()
```



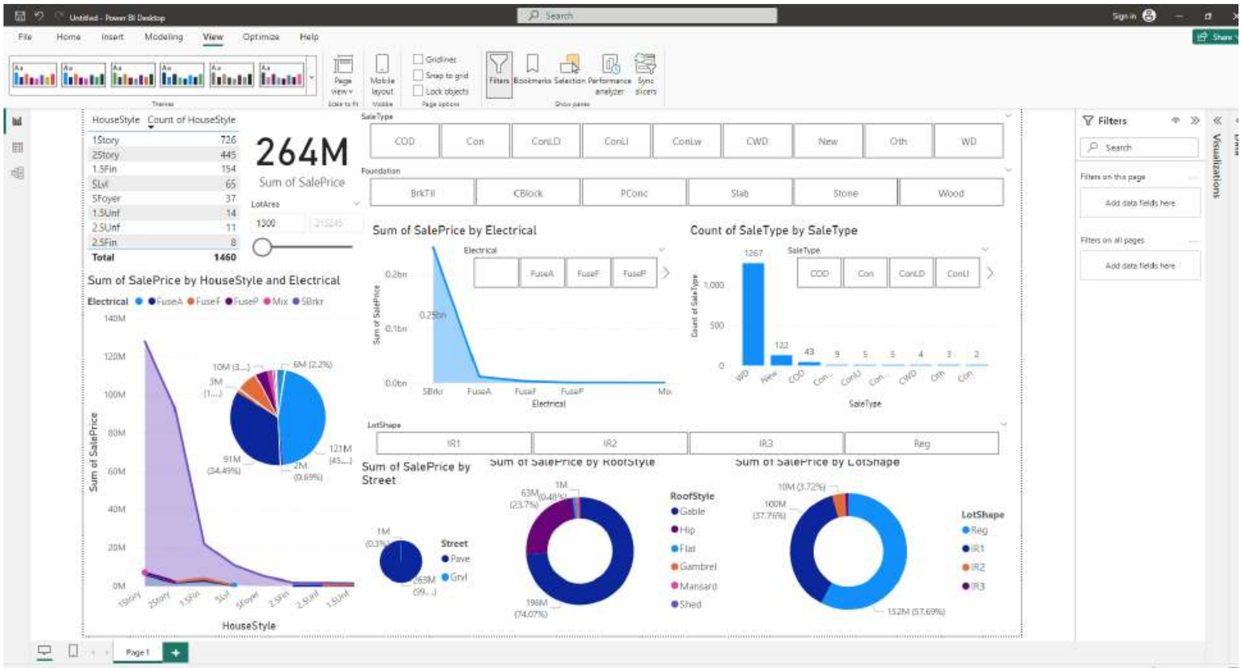
In summary, the successful implementation of the "AWS Data Integration and Visualization" project has laid the foundation for our understanding of data-driven decision making. Continuous improvement and adaptation to evolving business needs will be key to maximizing the benefits of the integrated system in the future.


Power BI
SERVICE

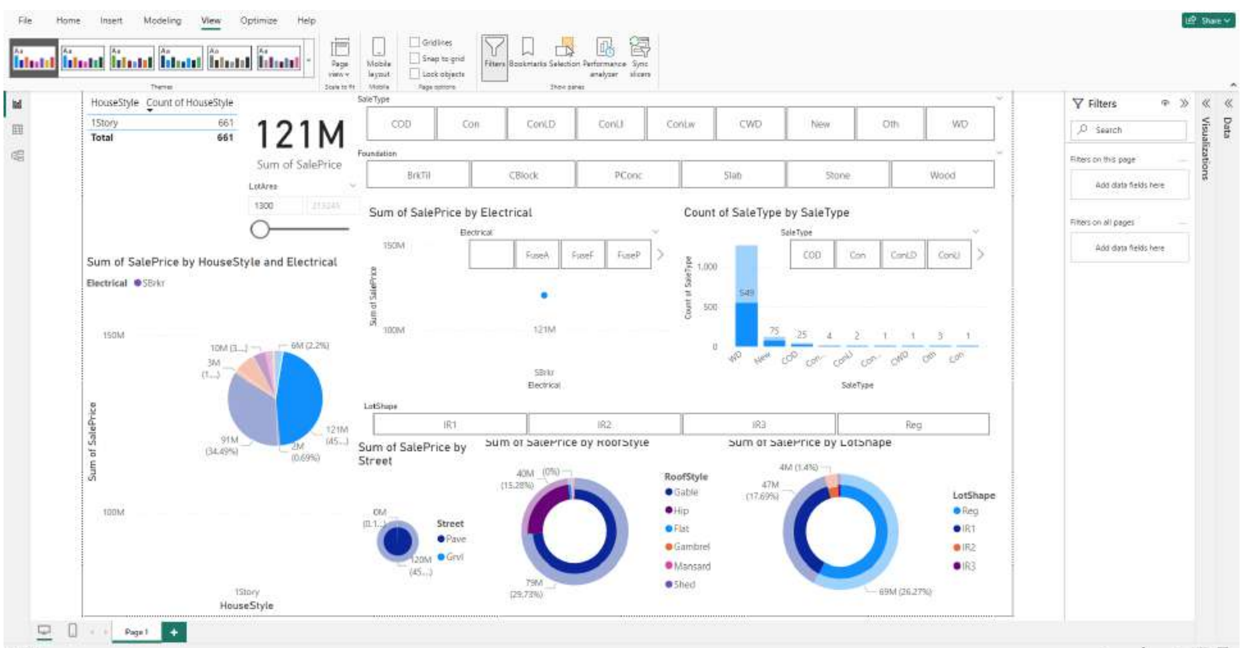

Get Data
All
Database
Amazon Redshift
Import data from Excel
Certified Connectors
Template Apps
Connect
Cancel
Visualizations
Page 1



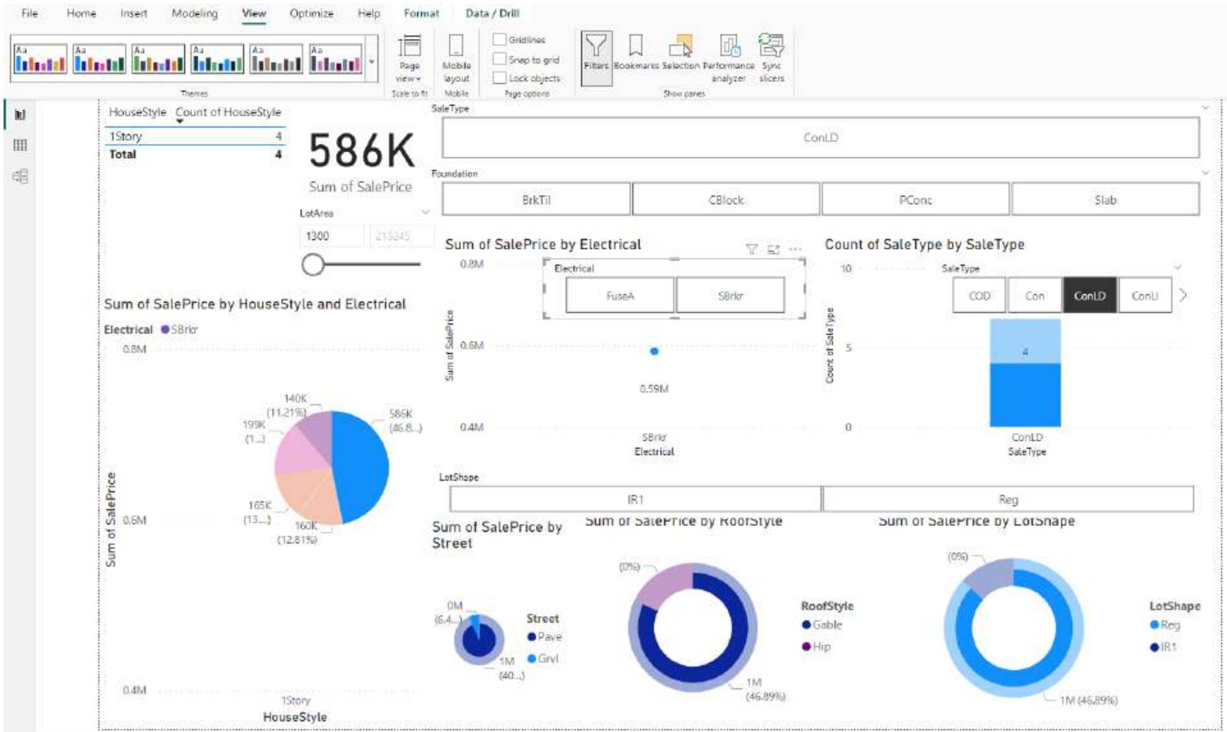
Complete Data Report Dashboard - Basics



Check 1st



Check 2nd


File
Home
Insert
Modeling
View
Optimize
Help
Format
Data / Drill
HouseStyle Count of HouseStyle
1Story 4
Total 4
586K
Sum of SalePrice
SaleType
ConLD
Foundation
BrkTil
CBlock
PConc
Slab
LotArea
Sum of SalePrice by Electrical
Count of SaleType by SaleType
Sum of SalePrice by HouseStyle and Electrical
Electrical
FuseA
SBrkr
LotShape
IR1
Reg
Sum of SalePrice by Street
Sum of SalePrice by RoofStyle
Sum of SalePrice by LotShape
Street
Pave
Grvl
RoofStyle
Gable
Hip
LotShape
Reg
IR1
HouseStyle

In [ ]: