Problem Statement

Challenge Overview:

This challenge comprises two distinct parts: Analytics and Machine Learning. You will work with CSV datasets stored in an S3 bucket named "**employee-data**" with a unique prefix followed by a random number. Within this bucket, there are two essential folders:

1. Inputfile: This folder contains the **employee_data.csv** dataset, which you will utilize for the Analytics part. Your objective is to leverage AWS Glue for data cleaning and transformations, subsequently storing the results in Redshift tables.
2. redshift_cleaned_data: This folder contains the **employee_cleaned_data.csv** dataset, designated for the Machine Learning tasks.

Analytics and ML pipeline:

Analytics: S3 – Glue – Redshift
Machine Learning: SageMaker -> Grab data from S3 -> Build the model -> Push the model data to S3

Note: Don't worry about the CSV files present in the local folder, only use the CSV files from the s3 bucket itself for the tasks outlined in this challenge.
Note: You can do any of the task first, both tasks are independent.

Your mission is to tackle each part systematically, ensuring efficient data processing and preparation for both Analytics and Machine Learning endeavors. Let's dive into the specifics of each task!

IMPORTANT NOTE: Use **US East (N. Virginia) us-east-1** Region only**.**

Setting Up an Amazon Redshift Cluster and Connection in AWS Glue:

Step 1: Create an Amazon Redshift Cluster

- **Navigate to Amazon Redshift**:
  - In the AWS Management Console, go to the Amazon Redshift service.
- **Create a New Cluster**:
  - Click on the "Create cluster" button.
  - Configure the cluster with the following details:
    - **Cluster Identifier**: Provide a unique identifier for the cluster (**redshift-cluster-1**).
    - **Node Type**: Select a node type based on your CPU, RAM, storage capacity, and drive type requirements (**dc2.large**).
    - **Number of Nodes**: Specify the number of nodes needed (**1**).
    - **Database Configurations**:
      - **Manually add the admin password**
      - **Admin Username**: Provide a login ID for the admin user (**awsuser**).
      - **Admin Password**: Enter password manually (**Awsuser1**).
    - **Cluster Permissions**: Create a new IAM role that grants the cluster necessary permissions (**Any S3 bucket** policy).
  - Keep default additional configurations (e.g., network, backup, security, maintenance, etc.).
- **Launch the Cluster**:
  - Once you have configured the cluster, click "Create cluster" to launch the Amazon Redshift cluster.
- **Monitor the Cluster**:
  - Monitor the progress of your cluster creation in the Amazon Redshift console.
  - Wait until the cluster status is "available" before proceeding.
- Kindly Don’t Create multiple redhsift clusters, keep only one cluster available with the name given in the problem statement.

Note 1: After cluster status is available, click on the actions and turn on the public access
Note 2: In cluster security group add inbound rule allow custom TCP 5439 port ipv4 address

Creating a VPC Endpoint for Amazon S3, AWS Glue, and Amazon Redshift Integration:

To facilitate seamless data transfer and integration between Amazon S3, AWS Glue, and Amazon Redshift, you need to create a VPC endpoint. Follow the instructions below to set up the VPC endpoint correctly.

Step 1: Create VPC Endpoint

- **Navigate to VPC Endpoint Creation**:
  - Access the AWS Management Console and go to the VPC service.
- **Configure VPC Endpoint Settings**:
  - Provide the following details:
    - **Name Tag**: Create a tag with a key of **s3-glue-redshift-endpoint**.
    - **Service Category**: Select **AWS services**.
    - **Service Name**: Choose **com.amazonaws.us-east-1.s3** with the type **Gateway** from the list.
    - **VPC**: Select the default VPC from the dropdown menu.
    - **Route Table**: Select the default route table.
  - **Note:** Leave other settings as default.
  - Click on create endpoint.

Navigate to the AWS Glue Data Catalog:

- **In the AWS Glue console**:
  - Click on the "**Databases**" link in the left-hand menu.
- **Create a New Database**:

- In the "**Databases**" section, click on the "**Add database**" button to create a new database.
- In the "**Create database**" form, provide a name for the database (**employees_db**).

Navigate to the Connections Section:

- **In the AWS Glue console**:
  - Go to the "**Connections**" section in the left-hand menu.
- **Create a New Connection**:
  - Click on create **new connection**.
  - Search for **Redshift** and select it, then proceed to the next step.
  - Select the Redshift connection you have created and enter the username and password of your Redshift cluster.
- **Connection Name**: Name the connection as **redshift_connection**.
- **Save the Connection**:
  - Once you have entered all the required information, click "**Save**" to create the connection.
- **Test the Connection**:
  - After creating the connection, test the connection to ensure it is successful.
- **Use the Connection in Your Glue Jobs**:
  - Now that the connection is set up, you can use it in your AWS Glue jobs to read from or write to your Redshift cluster.

Creating Crawlers in AWS Glue:

You are tasked with creating crawlers in AWS Glue to connect to data stores, determine the schema for your data, and create metadata tables in your data catalog. Follow the instructions below to create the necessary crawlers for your datasets.

Create Crawler for Source Data (S3 Bucket):

Step 1: Create Crawler

- **Navigate to AWS Glue:**
    - Go to the AWS Glue service.
- **Create Crawler:**
    - Click on "**Crawlers**" in the left-hand menu.
    - Click on the "**Create crawler**" button.
- **Configure Crawler:**
    - Provide the following details:
    - **Name:** Enter a name for the crawler (**Employee_Data**).
    - **Description:** Optionally, provide a description for the crawler.
    - **Data Store:** Select "**S3**" as the data store type.
    - **Include Path:** Specify the path to the S3 bucket where the source data (**employee_data.csv**) is located.
    - Add Existing IAM Role **AWS-S3-Glue-Redshift-Role**
    - Select database(**employees_db**) which you have created before.
- **Run Crawler:**
    - Click "**Next**" to proceed with the crawler configuration.
    - Review the settings and click "**Finish**".
    - Run the crawler to connect to the S3 bucket, determine the schema, and create metadata tables in the data catalog.

Create Crawlers for target Redshift Tables:

Step 2: Create Crawlers for Redshift Tables

For each Redshift output tables:

NOTE: Create redhsift output tables before creating the **Crawlers for Redshift Tables**, kindly check in the tasks for the output schema

- **Navigate to AWS Glue:**

    - Access the AWS Management Console and go to the AWS Glue service.
- **Create Crawler:**
    - Click on "Crawlers" in the left-hand menu.
    - Click on the "Add crawler" button.
- **Configure Crawler:**
    - Provide the following details:
    - **Name:** Enter a name for the crawler (**employees_cleaned_data**).
    - **Data Store:** Select "Amazon Redshift" as the data store type.
    - **JDBC Connection:** Choose the Redshift JDBC connection you previously created.
    - **Include Path:** Specify the path to the Redshift table (**dev/public/employees_cleaned_data**) (redshift database/schema/table_name).
    - Add Existing IAM Role **AWS-S3-Glue-Redshift-Role**
    - Select database(**employees_db**) which you have created before.
- **Run Crawler:**
    - Click "Next" to proceed with the crawler configuration.
    - Review the settings and click "Finish".
    - Run the crawler to connect to the Redshift table, determine the schema, and create metadata tables in the data catalog.
    - Repeat creating crawler steps for another each Redshift table crawler (**employees_dept_count**).
    - **(dev/public employees_dept_count)** path for the redhsift table for **employees_dept_count** crawler.

Create job with name s3_glue_redshift_job

Note: Do the below tasks in this job(**s3_glue_redshift_job**) itself using glue **visual ETL** tool only. Use only **AWS-S3-Glue-Redshift-Role** role for the job.

- In job details give name to the job and assign **AWS-S3-Glue-Redshift-Role** role to it and leave other configurations default.

1. **Task 1: Data Cleaning**
    a. **Input:** Retrieve data from the Employee Data crawler in the AWS Glue Data Catalog.
    b. **Filtering:** Identify employees whose age is greater than 25.
    c. **Schema Verification:** Ensure the schema of the filtered data matches the expected structure and datatypes.
    d. **Load data to Glue Catalog:** Store the filtered data in the **employees_cleaned_data** crawler in the AWS Data Catalog.
    e. Cleaned data is used for the further Machine learning Tasks.

Output Table: employees_cleaned_data

Column Name	Data Type
employee_id	int
department	Varchar(100)
region	Varchar(100)
education	Varchar(100)
gender	Varchar(100)
recruitment_channel	Varchar(100)
no_of_trainings	int
age	int
previous_year_rating	int
length_of_service	int
kpis_met_more_than_80	int

awards_won	int
avg_training_score	int

2. **Task 2: Data Transformation**
    a. Use the cleaned data node for this task
    b. **Input:** Access data from the Cleaned_data node crawler in the AWS Glue Data Catalog.
    c. **Departmental Analysis:** Calculate the count of employees for each department and store count in the new column Employees_Count.
    d. **Schema Verification:** Confirm that the schema and datatypes of the departmental analysis output are correct as given below.
    e. **Load data to Glue Catalog:** Store the output in the **employees_dept_count** crawler in the AWS Glue Catalog.

Output Table: employees_dept_count

Column Name	Data Type
department	Varchar(100)
Employees_count	int

Note1: Check the output in the redshift tables to ensure that data loaded successfully after completing of all the tasks.

Note2: Upon completion of the Analytics part, the cleaned data will be stored in a Redshift table. Assume the cleaned data in the redshift table is transferred to the s3 bucket. You can access this cleaned data in the S3 bucket "employee-data" within the "redshift_cleaned_data" folder. This preprocessed data is ready for utilization in the subsequent Machine Learning tasks.

Cloud-Driven HR Insights with Machine Learning

Cloud Driven Tasks:

Task 1: Launching a SageMaker Instance

Objective: Launch an Amazon SageMaker notebook instance to develop and deploy machine learning models.

Instructions:

1. **Access Amazon SageMaker**:
    - In the AWS Management Console, find and select "Services".
    - Under the "Machine Learning" category, click on "SageMaker" to open the SageMaker dashboard.
2. **Create a New Notebook Instance**:
    - In the SageMaker dashboard, click on “Notebook instances” in the left sidebar.
    - Click the “Create notebook instance” button at the top right corner.
    - Enter a name for your notebook instance, e.g., **MyFirstMLInstance**.
    - Choose an instance type. Use **ml.t3.medium** instance option.
    - Under "Permissions and encryption", choose or create a new role that has necessary permissions to access S3 buckets.
3. **Create the Notebook Instance**:
    - Review all the settings to ensure they are correct.
    - Click “Create notebook instance” at the bottom of the form.
4. **Access the Notebook**:
    - Wait for the status of the instance to change from “Pending” to “InService”.
    - Click “Open Jupyter” to access the Jupyter dashboard and start working on your machine learning projects.

ML Driven Tasks:

Task 1: Setting Up AWS Resources

Instructions:

- Begin by importing the AWS SDK for Python (**boto3**) and other necessary libraries.
- Use the AWS SDK to establish a session and obtain the execution role for your AWS account.
- Use the bucket name: **employee-data**
- Use SageMaker library to set the execution roles to access the data from S3 bucket.

Hints:

- You might need to look up how to use **boto3.client('s3')** to interact with the S3 service.

Task 2: Loading and Preprocessing Data

Instructions:

- Build the S3 path for the dataset '**employee_cleaned_data.csv'** in folder named '**inputfiles'** using string formatting to concatenate the **bucket** name and file key.
- Load the dataset into a pandas DataFrame. Remove the column that might be used to uniquely identify entries, such as **employee_id**, using DataFrame operations.
- Extract numeric values from the 'region' column using a string extraction method and convert this column to a numeric data type.

Task 3: Data Analysis and Visualization

Instructions:

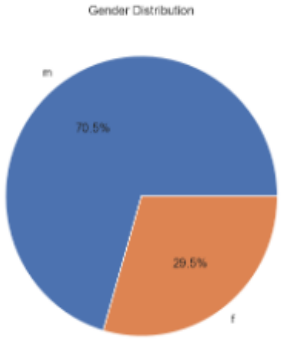
- Verify and print the number of duplicate records if exists and the shape of the DataFrame after cleaning to ensure the dataset is ready for analysis.

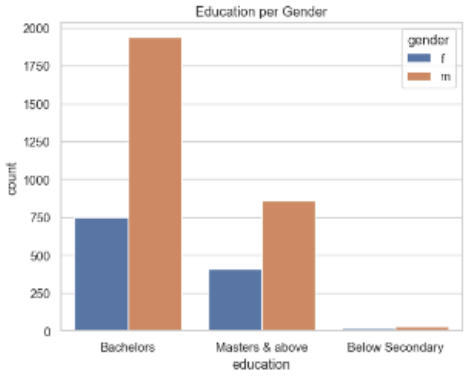
- Create a pie chart to visualize the distribution of values in the 'gender' column.
- Use seaborn to generate a count plot for the 'education' column, categorizing data by gender to analyze demographic distributions.

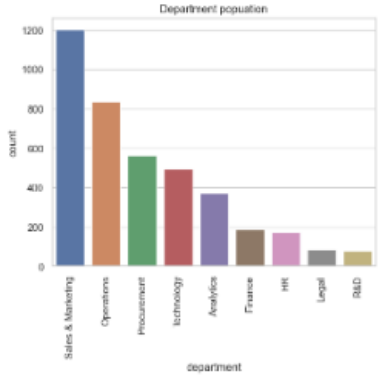
Hints:

- When creating visualizations, setting parameters like **figsize** can help improve the readability of the plot.

Sample Graphs:







Task 4: Feature Engineering

Instructions:

- Split the data based on the independent and dependent variables and store it in for example X and y.
- Transform the columns in the data X using sklearn’s column transfer technique and preprocess them using one hot encoder & passthrough methods.
- Create new features that could be predictive of outcomes like turnover (e.g., employee engagement scores based on multiple factors). Use sklearns feature selection functions to grab the best feature based on techniques like regression with a `k` value of 5.

- Fit and transform the data in `X` and using the feature selector, store the new data in variable like `X_selected`.
- Verify the selected columns by checking the shape of the data.

Sample output:

- List of column names
- Ex: [‘column1’, ‘column2’ .....]

Task 5: Creating the model

Instructions:

- Split your data into training and testing sets using train_test_split from scikit-learn with a test size of 20% and random state of `0`. Use the feature engineered data i.e `X_selected`.
- Feature scale the X_train and X_test data using Sklearn’s standard scaler.
- Build a LogisticRegression model using scikit-learn with a random state of `0`.
- Get the predicted value and store it in a variable such as `y_pred`.
- Evaluate models using metrics such as accuracy, precision, recall score and F1-score.
- Plot a heatmap of the confusion matrix against the actual and the predicted values for more insights.

Sample output:

- Accuracy – floating point value greater than 65.0
- Precision – floating point value greater than 55.0; not less than 50.0
- F1-Score – floating point value greater than 50.0
- Recall Score – floating point value greater than 45.0

Task 6: Deploying a Machine Learning Model

Instructions:

- Serialize your trained logistic regression model using joblib or a similar library and store it in the specified S3 bucket, i.e **employee-data**.
- Using the tempfile library, along with joblib generate a pickle file of the final model we have created. With the help of s3_client, push the pickle file to the S3 folder named **ml-output** inside the bucket **employee-data** and print a confirmation message as an output to verify.

Hints:

- Temporary files in Python can be managed using **tempfile.TemporaryFile()**.
- Look into **joblib.dump()** for saving the model.

Sample message:

- Successfully pushed data to S3: model.pkl

Task 7: Prediction using the deployed model

Instructions:

- With the similar approach of pushing the pickle file to the S3 bucket, using tempfile and joblib, download the pickle file and load it in a variable for example `model`.
- Fit the newly loaded `model` and run predictions on the testing data and store it in some variable to calculate the metrics.
- Execute a prediction using the test dataset and compute the accuracy of the model using standard metrics.

Hints:

- Temporary files in Python can be managed using **tempfile.TemporaryFile()**.
- Look into **joblib.load()** for loading the model