**Task One: Building an ETL pipeline to support our Applied Data Scientists**

**Transforming the raw data into the features.**

Data files:

* **accounts.csv**: Contains account information for our customers.
* **skus.csv**: Contains the details about our product lines.
* **invoices.csv**: Contains the invoices issued to each account.
* **invoice\_line\_items.csv**: Contains the details of the SKUs sold on each invoice.

Required features in the output file:

* **inv\_id**: the ID of the invoice to be passed into the model.
* **acct\_id**: the ID of the account for that invoice.
* **inv\_total**: Total value (in dollars) for the invoice.
* **inv\_items**: Total number of items in the invoice
* **acct\_age**: The age of the account (in days) at the time the invoice was issued (i.e. the number of days between when the account was set up, and the invoice date).
* **num\_inv\_120d**: the number of invoices for the account in the 120 days prior to the invoice's issuing date.
* **cum\_tot\_inv\_acct**: The cumulative number of invoices for the account up to date that the invoice was issued.
* **is\_late**: A flag to indicate if the invoice was paid late (i.e. more than 30 days after issuing). Contains 1 if invoice was paid late, 0 otherwise.

I used python to work on the files and find the features. Please refer to the code in ETL.ipynb file.

Import the files into python using the path of the file or load them into the directory. The data is stored in dataframe named df. We can create a separate csv file for this data.

The features were used to create classification models that would help us identify customers and invoices that are at the risk of late payments. I divided the dataset in train data (70%) and test data (30%).

I created classification models using:

* K-Nearest Neighbour
* Decision Tree
* Random Forest

We achieved 84% accuracy with Random Forest algorithm.

After this, I connected to PostgreSQL database to load the data.

**Tests your ETL job**

To perform the ETL job, I used PostgreSQL. The database is setup on my local machine. I created a database named ‘reece’ where I saved the data from the files. The details to connect to the database are below:

Graphical user interface, application

Description automatically generated

We need to create the database before adding tables to it. To implement this program, create a database with name ‘reece’.

**Connecting to the database:**

To connect to the database, first import the libraries.

import psycopg2

import psycopg2.extras

Then, enter the following details. The localhost connects through port 5432 which is default for PostgreSQL. We can change it if required.

hostname = 'localhost'

database = 'reece'

username = 'postgres'

pwd = 'admin'

port\_id = 5432

conn = None

**Establishing the connection:**

connection = psycopg2.connect(

host = hostname,

dbname = database,

user = username,

password = pwd,

port = port\_id)

cursor = connection.cursor()

After establishing the connection, we can use SQL to create tables and input data in the tables. Refer file ‘ETL.ipynb’.

**Close the connection:**

To close the connection, use the below command.

connection.close()

The data is successfully loaded in the database.

Graphical user interface, application

Description automatically generated

I used dataframe(df) to calculate the features using the files provided. The data in invoices.csv file had to be iterated repeatedly for calculating some of these features. The data must be stored in a temporary location to perform calculations before loading it to the database. After the calculations were done performing, I then loaded the data in the database.

When working with large data, sometimes data transformation may need a lot of calculations. These calculations can consume a lot of time and space. It is important that the machine is capable to perform these calculations quickly and efficiently. This is where cloud computing comes in.

Cloud computing can help you reduce the IT costs, is scalable, introduces flexibility in work practices and much more. You do not have to worry about your on-prem machines and can access your data from anywhere. Cloud computing provides data security features, so your data is secured from threats.

**Task Two: System Design**

I used Power BI and PostgreSQL in this example. Power BI is very powerful tool, and it has multiple data input methods that we can use to connect it with the database.

Connecting power BI to PostgreSQL database:

Graphical user interface

Description automatically generated

We have 2 options in data connectivity mode:

1. Import
2. DirectQuery

The import method imports the data into Power BI where as the DirectQuery method connects to the database. The data resides in the database itself; it is only modelled in Power BI. When collaborating with other people and creating dashboards, we use Power BI Service. In power BI service, you can choose to set up scheduled refresh. These makes sure that your data is up to date always.

When data is available in power BI, we can start creating data models. First, we create a date table that references all our data. I used DAX to create the date table. It consists of date, calendar year, quarter, month, and weekday. I created a sample dashboard to visualize our data:

Chart, line chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Filters to filter the data:



You can select the Calendar year, Quarter and Month you want to view and choose the account id you wish to view the data of, and it will display the Name of the account holder and details based on his invoices. It shows how many invoices were generated in the select month and how many were generated throughout the year.

It will also tell you how many total items were bought and what was the total amount for the selected month and compares and displays the value of the previous month as well.

You can select the invoice id and view all the details of that invoice including issue and payment dates, amount, items, quantities, contact details and other features.

The data in the database can also be accessed using SQL. There are two methods to do so:

1. It can be done using python and connecting to the source database. We have already established the connection and only need to execute the SQL query.

Graphical user interface, text, application

Description automatically generated

Edit out the command in the #Execute Query section to whatever SQL command you wish to run, and press run. No changes to python code need to be made.

2. Executing SQL commands from the database itself. Every database gives you access to view its data using SQL. Simply write the SQL query and execute.

