

## Final Project ETL

CIS 9440 – Data Warehousing for Analytics

Due: April 24<sup>th</sup>, 2021

Group Number - <1>

Student - <Carmen Ruan>

### Final Project Milestone #3

Step by step summary of ETL process submitted as PDF. Include summary text and screenshots where necessary.

1. I decided to use Python 3.8 on Spyder to perform the ETL process and created the two files which are ETL\_Project\_Milestone\_3.py and dsn\_functions\_project.py
2. First I have loaded the necessary libraries that I will be using for the process and set up the Google BigQuery credentials. I created a new project on Google BigQuery which required that I download a new key.

```
# load libraries
import pandas as pd
import numpy as np

from google.cloud import bigquery
from google.oauth2 import service_account

# setup Google BigQuery credentials
key_path = 'C:\\Users\\cabul\\OneDrive\\Desktop\\Academics\\Spring Semester\\CIS 9440\\netflix-unemployment-analysis-935279ca1c4b.json'
credentials = service_account.Credentials.from_service_account_file(
    key_path, scopes=["https://www.googleapis.com/auth/cloud-platform"],)
client = bigquery.Client(credentials=credentials, project=credentials.project_id)
```

3. After that I start with the Extract functions. The extract function is used to extract relevant data for Netflix stock prices. It used SQL query to run in BigQuery to extract the Netflix stock price data and then have it stored inside a dataframe for me to use throughout the process. The same extraction process is applied to unemployment data. I considered setting LOCATION only to USA but decided that I don't really need to since locations will be having indexes and that can be used to find locations that are only in USA. I can add the WHERE statement back in whenever I feel like it is necessary as well.

```

## Extract relevant data for Netflix stock prices
def extract_Netflix_data():
    # SQL query to run in BigQuery to extract Netflix stock price data
    sql_query = """
        SELECT Date,
        Open,
        High,
        Low,
        Close,
        Volume
        FROM final_project.netflix_stock_prices
    """

    # store extracted data in new dataframe
    Netflix_df = client.query(sql_query).to_dataframe()

    # validate that >0 stories have been extracted and return dataframe
    if len(Netflix_df) > 0:
        print(len(Netflix_df), "netflix stock data extracted")
        return Netflix_df
    else:
        print("Netflix stock data extraction FAILED")

## Extract unemployment data
def extract_Unemployment_data():
    sql_query = """
        SELECT LOCATION,
        TIME,
        VALUE
        FROM final_project.unemployment_rate_data
        ORDER BY TIME DESC;
    """

    # WHERE LOCATION = "USA"
    # store extracted data in new dataframe
    Unemployment_df = client.query(sql_query).to_dataframe()

    # validate that >0 stories have been extracted and return dataframe
    if len(Unemployment_df) > 0:
        print(len(Unemployment_df), "unemployment data extracted")
        return Unemployment_df
    else:
        print("Unemployment rate extraction FAILED")

```

4. Then it is the data profiling process for the extracted data which is pretty much standard and didn't require changing for either dataframes.

```

def data_profiling_dataframe(df):
    # Get memory used by each column in the raw data dataset in MB
    mem_used_dtypes = pd.DataFrame(df.memory_usage(deep=True) / 1024**2)
    df_mem = df.memory_usage(deep=True).sum() / 1024**2
    file_size = float('nan')
    sz_increase = ((df_mem - file_size) / file_size)

    # Rename column
    mem_used_dtypes.rename(columns={ 0: 'memory'}, inplace=True)

    # Drop index memory usage since this is not required when merging with Data Quality Dataframe
    mem_used_dtypes.drop('Index', axis=0, inplace=True)

    # define number of columns in dataframe
    no_of_rows = len(df.columns)
    # Constructing the data_qlt_df dataframe and pre-assigning and columns
    data_qlt_df = pd.DataFrame(index=np.arange(0, no_of_rows),
                               columns=('column_name', 'col_data_type',
                                         'col_memory', 'non_null_values',
                                         'unique_values_count', 'column_dtype'))

    # Add rows to the data_qlt_df dataframe
    for ind, cols in enumerate(df.columns):
        # Count of unique values in the column
        col_unique_count = df[cols].nunique()

        data_qlt_df.loc[ind] = [cols,
                                df[cols].dtype,
                                mem_used_dtypes['memory'][ind],
                                df[cols].count(),
                                col_unique_count,
                                cols + '~' + str(df[cols].dtype)
                                ]

```

```

# Use describe() to get column stats of raw dataframe
raw_num_df = df.describe().T.round(2)

# Merging the df.describe() output with rest of the info to create a single Data Profile Dataframe
data_qlt_df = pd.merge(data_qlt_df, raw_num_df, how='left',
                       left_on='column_name', right_index=True)

# Calculate percentage of non-null values over total number of values
data_qlt_df['%_of_non_nulls'] = (data_qlt_df['non_null_values']/df.shape[0])*100

# Calculate null values for the column
data_qlt_df['null_values'] = df.shape[0] - data_qlt_df['non_null_values']

# Calculate percentage of null values over total number of values
data_qlt_df['%_of_nulls'] = 100 - data_qlt_df['%_of_non_nulls']

# Calculate percentage of each column memory usage compared to total memory used by raw data dataframe
data_qlt_df['%_of_total_memory'] = data_qlt_df['col_memory'] / data_qlt_df['col_memory'].sum() * 100

# Calculate the total memory used by a given group of data type
# See Notes section at the bottom of this notebook for advatages of using 'transform' function with group_by
data_qlt_df["dtype_total"] = data_qlt_df.groupby('col_data_type')['col_memory'].transform('sum')
data_qlt_df["dtype_total"] = "NA"

# Calculate the percentage memory used by each column data type compared to the total memory used by the group of data type
# the above can be merged to one calculation if we do not need the total as separate column
#data_qlt_df["%_of_dtype_mem2"] = data_qlt_df["Dtype Memory"] / (data_qlt_df.groupby('Data Type')['Dtype Memory'].transform('sum')) * 100
data_qlt_df["%_of_dtype_mem"] = "NA"

# Calculate the percentage memory used by each group of data type of the total memory used by dataset
data_qlt_df["dtype_%_total_mem"] = "NA"

# Calculate the count of each data type
data_qlt_df["dtype_count"] = "NA"

# Calculate the total count of column values
data_qlt_df["count"] = data_qlt_df['null_values'] + data_qlt_df['non_null_values']

# Reorder the Data Profile Dataframe columns
data_qlt_df = data_qlt_df[
    ['column_name', 'col_data_type', 'col_memory',
     '%_of_dtype_mem', '%_of_total_memory',
     'dtype_count', 'dtype_total', 'dtype_%_total_mem',
     'non_null_values', '%_of_non_nulls',
     'null_values', '%_of_nulls', 'unique_values_count',
     'count', 'mean', 'std', 'min', '25%',
     '50%', '75%', 'max']]

# drop unneeded columns
for c in ['%_of_dtype_mem', 'dtype_count', 'dtype_total', 'dtype_%_total_mem']:
    data_qlt_df.drop(c, axis = 1, inplace=True)

return data_qlt_df

```

5. With that, next comes the Transform functions. I cleaned the Netflix stock prices dataframe and returned the dataframe. This is also applied to the unemployment data.

```

## Clean Netflix stock prices data
def clean_Netflix_data(df):
    # check if table exists in your database
    try:
        table_id = 'final_projct.Adjusted_Price_Fact'
        table = client.get_table(table_id)
        print("Adjusted Price fact table exists, number of rows: ", table.num_rows)
        print("now filtering for only new data")

        # if the table is not already in the database, clean all data
    except:
        print("Adjusted Price fact table is not in database, cleaning all data")

    return df

## Unemployment rate data
def clean_Unemployment_data(df):
    # check if table exists in your database
    try:
        table_id = 'final_projct.Employment_Rate_Fact'
        table = client.get_table(table_id)
        print("Unemployment Rate Fact table exists, number of rows: ", table.num_rows)
        print("now filtering for only new data")

        # if the table is not already in the database, clean all data
    except:
        print("Unemployment Rate Fact table is not in database, cleaning all data")

    return df

```

6. In creating the location\_dimension, I had to create it based on the unemployment dataframe. I used created manual location\_id indices so that it will be easier to identify location later. It also had a column for country\_abbreviation. I realized that it wasn't necessary to find and include the country's full names as well so I omitted that.

```

def create_location_dimension(Unemployment_df):

    location_dim = pd.DataFrame(data=Unemployment_df,
                                columns = ['location_id', 'country_abbreviation'])

    location_dim['location_id']=range(1,len(location_dim)+1)

    location_dim['country_abbreviation']=Unemployment_df['LOCATION']

    print("location dimension created")
    return location_dim

```

7. The data dimension is just breaking down the dates into smaller parts such as finding the year, the year\_week, year\_day, month, month\_name and information pertaining all to the date.

```
def create_date_dimension():
    sql_query = """
        SELECT
            CONCAT (FORMAT_DATE("%Y",d),FORMAT_DATE("%m",d),FORMAT_DATE("%d",d)) as date_id,
            d AS full_date,
            EXTRACT(YEAR FROM d) AS year,
            EXTRACT(WEEK FROM d) AS year_week,
            EXTRACT(DAY FROM d) AS year_day,
            EXTRACT(YEAR FROM d) AS fiscal_year,
            FORMAT_DATE('%Q', d) as fiscal_qtr,
            EXTRACT(MONTH FROM d) AS month,
            FORMAT_DATE('%B', d) as month_name,
            FORMAT_DATE('%w', d) AS week_day,
            FORMAT_DATE('%A', d) AS day_name,
            (CASE WHEN FORMAT_DATE('%A', d) IN ('Sunday', 'Saturday') THEN 0 ELSE 1 END) AS day_is_weekday,
        FROM (
            SELECT
                *
            FROM
                UNNEST(GENERATE_DATE_ARRAY('2011-01-01', '2021-01-01', INTERVAL 1 DAY)) AS d )
    """

    # store extracted data in new dataframe
    date_df = client.query(sql_query).to_dataframe()

    # validate that >0 stories have been extracted and return dataframe
    if len(date_df) > 0:
        print("date dimension created")
        return date_df
    else:
        print("date dimension FAILED")
```

8. As I was creating the adjusted price fact table, I had to transform the dates so that the formats are consistent and had them all transformed to be YYYYMMDD format. The same is applied to the unemployment fact table and added an extra step for the location dimension where I had to merge the data based on the location of the data.

```
## create Adjusted Price Fact Table
def create_Adjusted_Price_Fact(df):
    # create date_id column
    df['date_id'] = df['Date'].apply(lambda x: x.strftime("%Y%m%d"))

    # drop unneeded columns
    for c in ['Date']: ##took out adj close
        df.drop(c, axis = 1, inplace=True)

    return df

## create Unemployment Rate Fact table
def create_Unemployment_Fact(df, location_dim):
    # create date_id column
    df['date_id'] = df['TIME'].apply(lambda x: x.strftime("%Y%m%d"))

    # create location_id dim
    df = df.merge(location_dim, left_on='LOCATION', right_on='country_abbreviation',
        how='inner')

    # drop unneeded columns
    for c in ['TIME', 'country_abbreviation']:
        df.drop(c, axis = 1, inplace=True)

    return df
```

9. After extracting and transforming, I just had to load the tables into BigQuery.

```
## Load table to BigQuery
def load_df_to_bigquery(df, table_name):

    dataset_id = 'netflix-unemployment-analysis.final_project'

    dataset_ref = client.dataset(dataset_id)
    job_config = bigquery.LoadJobConfig()
    job_config.autodetect = True
    job_config.write_disposition = "WRITE_TRUNCATE"

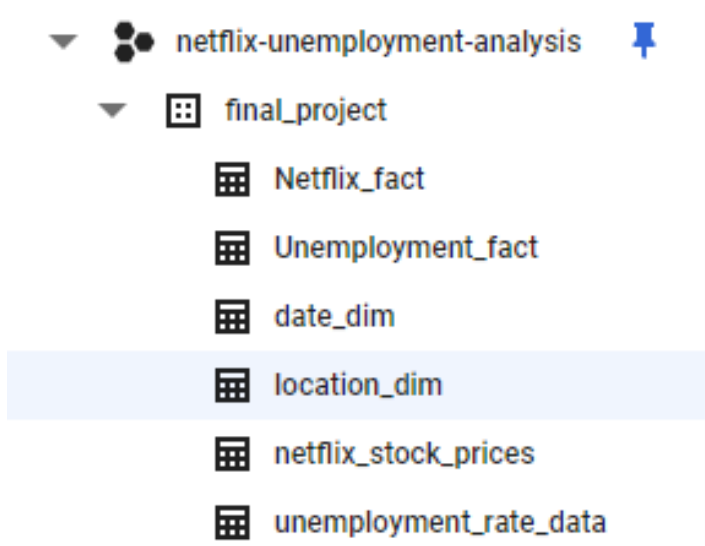
    upload_table_name = 'final_project.'+str(table_name)

    load_job = client.load_table_from_dataframe(df, upload_table_name,
                                                job_config=job_config)

    print("Starting job {}".format(load_job))
```

10. All of these functions are called in the ETL\_Project\_Milestone\_3.py file so that it looks cleaner. (all code will be attached)


11. Google BigQuery



Netflix\_fact

 QUERY TABLE

[Schema](#) [Details](#) [Preview](#)

Field name	Type	Mode	Policy tags 	Description
Open	FLOAT	NULLABLE		
High	FLOAT	NULLABLE		
Low	FLOAT	NULLABLE		
Close	FLOAT	NULLABLE		
Volume	INTEGER	NULLABLE		
date_id	STRING	NULLABLE		

Edit schema

Netflix\_fact

 QUERY TABLE

[Schema](#) [Details](#) [Preview](#)


Row	Open	High	Low	Close	Volume	date_id
1	30.661428	35.364285	29.0	29.524286	736982400	20110201
2	149.800003	191.5	144.25	181.660004	185144700	20170701
3	337.179993	371.48999	336.5	358.100006	148192100	20190201
4	95.0	101.269997	84.5	91.25	320138000	20160701
5	373.109985	393.519989	290.25	375.5	201979100	20200301
6	304.589996	332.049988	250.0	286.130005	257126400	20181101
7	418.829987	474.01001	404.25	455.040009	117279900	20200601
8	114.599998	129.289993	85.5	115.029999	469661800	20150801
9	454.0	575.369995	454.0	488.880005	232227800	20200701
10	385.450012	419.769989	328.0	337.450012	305393800	20180701
11	91.230003	98.849998	90.5	97.449997	182611000	20160801
12	98.0	129.289993	97.629997	124.870003	274097800	20161001
13	8.707143	10.068571	8.671429	9.784286	499411500	20120601
14	58.842857	65.398567	56.715714	63.66143	317384200	20140201
15	9.824286	12.378571	8.02	8.121429	988045100	20120701
16	490.859985	549.039978	466.549988	529.559998	116269200	20200801
17	117.519997	129.070007	113.949997	123.800003	128528700	20161201
18	91.790001	97.480003	79.949997	93.410004	389268900	20160201
19	52.401428	58.914288	45.581429	58.475716	600418700	20140101
20	267.350006	308.75	257.01001	287.410004	231556400	20191001



# Unemployment\_fact

 QUERY TABLE

[Schema](#) [Details](#) [Preview](#)

Field name	Type	Mode	Policy tags 	Description
LOCATION	STRING	NULLABLE		
VALUE	FLOAT	NULLABLE		
date_id	STRING	NULLABLE		
location_id	INTEGER	NULLABLE		

Edit schema

# Unemployment\_fact

[Schema](#) [Details](#) [Preview](#)

Row	LOCATION	VALUE	date_id	location_id
1	AUS	6.351635	20210101	3
2	AUS	6.351635	20210101	34
3	AUS	6.351635	20210101	69
4	AUS	6.351635	20210101	107
5	AUS	6.351635	20210101	146
6	AUS	6.351635	20210101	185
7	AUS	6.351635	20210101	224
8	AUS	6.351635	20210101	263
9	AUS	6.351635	20210101	302
10	AUS	6.351635	20210101	341
11	AUS	6.351635	20210101	380
12	AUS	6.351635	20210101	419
13	AUS	6.351635	20210101	458
14	AUS	6.351635	20210101	498
15	AUS	6.351635	20210101	538
16	AUS	6.351635	20210101	578
17	AUS	6.351635	20210101	618
18	AUS	6.351635	20210101	658
19	AUS	6.351635	20210101	698
20	AUS	6.351635	20210101	738

date\_dim

Schema Details Preview

Field name	Type	Mode	Policy tags ⓘ
date_id	STRING	NULLABLE	
full_date	DATE	NULLABLE	
year	INTEGER	NULLABLE	
year_week	INTEGER	NULLABLE	
year_day	INTEGER	NULLABLE	
fiscal_year	INTEGER	NULLABLE	
fiscal_qtr	STRING	NULLABLE	
month	INTEGER	NULLABLE	
month_name	STRING	NULLABLE	
week_day	STRING	NULLABLE	
day_name	STRING	NULLABLE	
day_is_weekday	INTEGER	NULLABLE	

date\_dim

QUERY TABLE

SHARE TABLE


COPY TABLE

DELETE TABLE


Schema Details Preview

Row	date_id	full_date	year	year_week	year_day	fiscal_year	fiscal_qtr	month	month_name	week_day	day_name	day_is_weekday
1	20110102	2011-01-02	2011	1	2	2011	1	1	January	0	Sunday	0
2	20110109	2011-01-09	2011	2	9	2011	1	1	January	0	Sunday	0
3	20110116	2011-01-16	2011	3	16	2011	1	1	January	0	Sunday	0
4	20110123	2011-01-23	2011	4	23	2011	1	1	January	0	Sunday	0
5	20110130	2011-01-30	2011	5	30	2011	1	1	January	0	Sunday	0
6	20120101	2012-01-01	2012	1	1	2012	1	1	January	0	Sunday	0
7	20120108	2012-01-08	2012	2	8	2012	1	1	January	0	Sunday	0
8	20120115	2012-01-15	2012	3	15	2012	1	1	January	0	Sunday	0
9	20120122	2012-01-22	2012	4	22	2012	1	1	January	0	Sunday	0
10	20120129	2012-01-29	2012	5	29	2012	1	1	January	0	Sunday	0
11	20130106	2013-01-06	2013	1	6	2013	1	1	January	0	Sunday	0
12	20130113	2013-01-13	2013	2	13	2013	1	1	January	0	Sunday	0
13	20130120	2013-01-20	2013	3	20	2013	1	1	January	0	Sunday	0
14	20130127	2013-01-27	2013	4	27	2013	1	1	January	0	Sunday	0
15	20140105	2014-01-05	2014	1	5	2014	1	1	January	0	Sunday	0
16	20140112	2014-01-12	2014	2	12	2014	1	1	January	0	Sunday	0
17	20140119	2014-01-19	2014	3	19	2014	1	1	January	0	Sunday	0
18	20140126	2014-01-26	2014	4	26	2014	1	1	January	0	Sunday	0
19	20150104	2015-01-04	2015	1	4	2015	1	1	January	0	Sunday	0
20	20150111	2015-01-11	2015	2	11	2015	1	1	January	0	Sunday	0

location\_dim

 QUERY TABLE

[Schema](#) [Details](#) [Preview](#)

Field name	Type	Mode	Policy tags 	Description
location_id	INTEGER	NULLABLE		
country_abbreviation	STRING	NULLABLE		

Edit schema

location\_dim

[Schema](#) [Details](#) [Preview](#)

Row	location_id	country_abbreviation
1	3	AUS
2	34	AUS
3	69	AUS
4	107	AUS
5	146	AUS
6	185	AUS
7	224	AUS
8	263	AUS
9	302	AUS
10	341	AUS
11	380	AUS
12	419	AUS
13	458	AUS
14	498	AUS
15	538	AUS
16	578	AUS
17	618	AUS
18	658	AUS
19	698	AUS
20	738	AUS

OPTIONAL – include code used for ETL

dsn\_functions\_project.py

[https://d.docs.live.net/797771a85995cc48/Desktop/Academics/Spring%20Semester/CIS%209440/dsn\\_functions\\_project.py](https://d.docs.live.net/797771a85995cc48/Desktop/Academics/Spring%20Semester/CIS%209440/dsn_functions_project.py)

ETL\_Project\_Milestone\_3.py

[https://d.docs.live.net/797771a85995cc48/Desktop/Academics/Spring%20Semester/CIS%209440/ETL\\_Project\\_Milestone\\_3.py](https://d.docs.live.net/797771a85995cc48/Desktop/Academics/Spring%20Semester/CIS%209440/ETL_Project_Milestone_3.py)

or Google Drive folder:

[https://drive.google.com/drive/folders/1pahFZA\\_BX9FlykbanizmFkCeXqxoyM1w?usp=sharing](https://drive.google.com/drive/folders/1pahFZA_BX9FlykbanizmFkCeXqxoyM1w?usp=sharing)