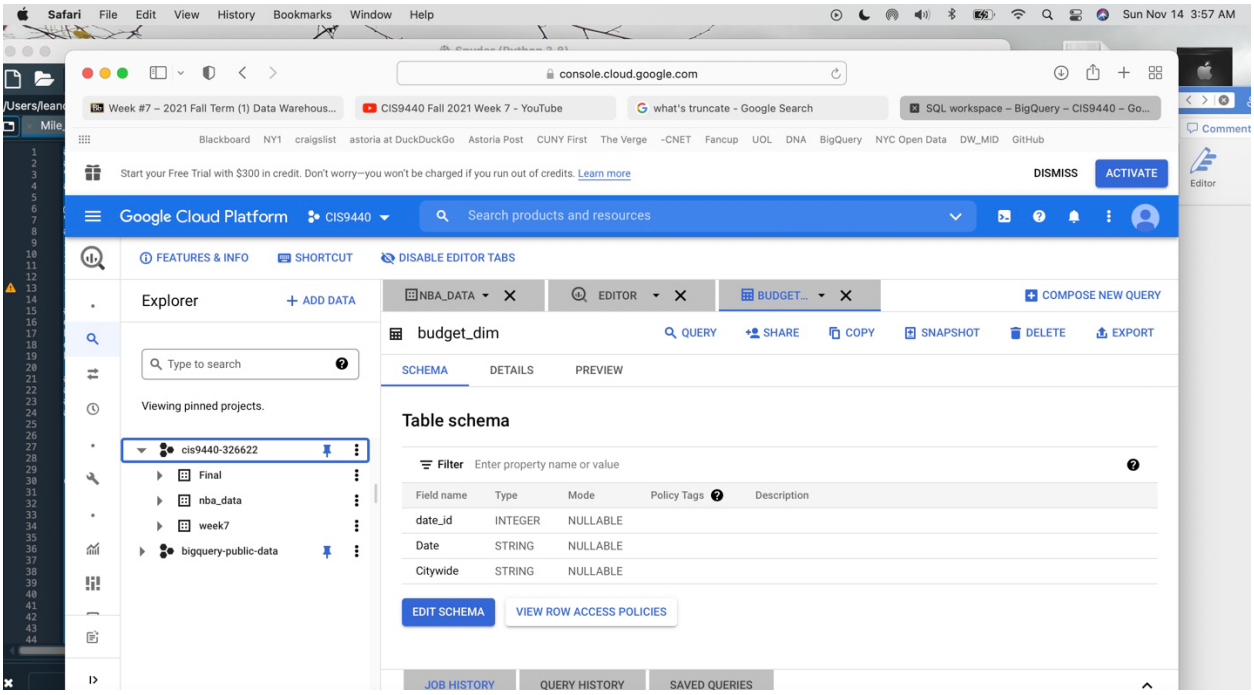
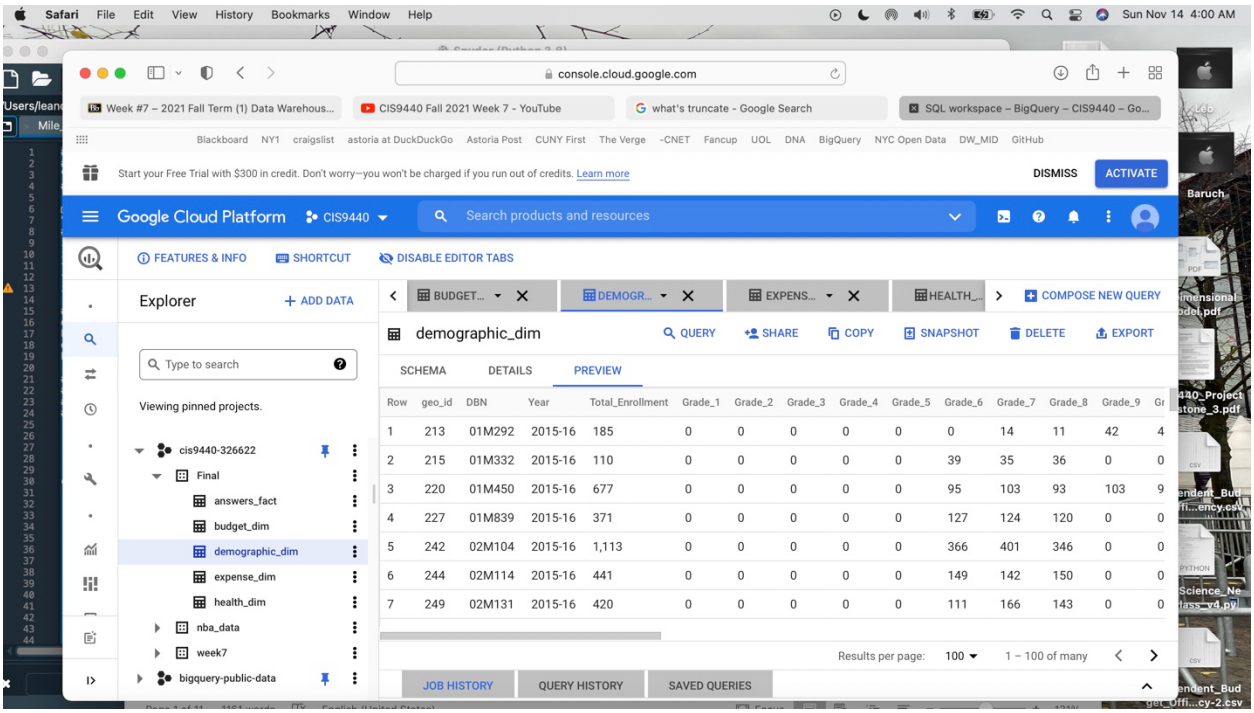
Mile Stone #3
Leandro Coimbra. CIS 9440 Data Warehouse

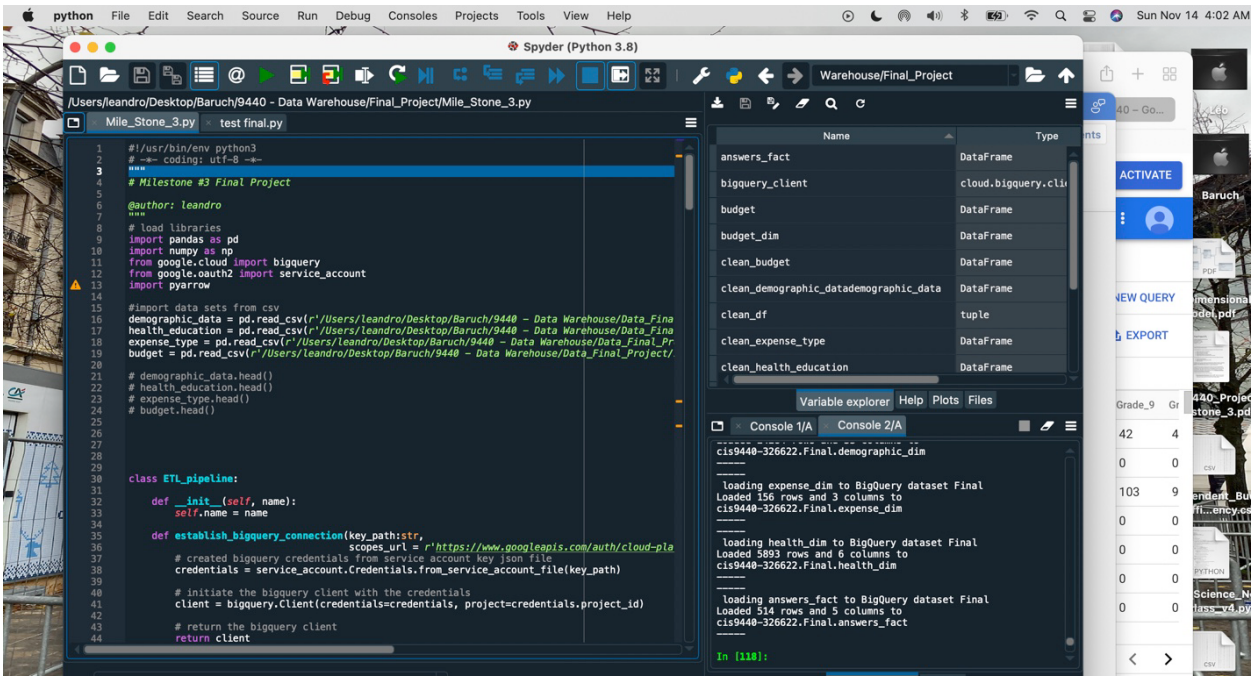
The steps I have taken for the following ETL were:

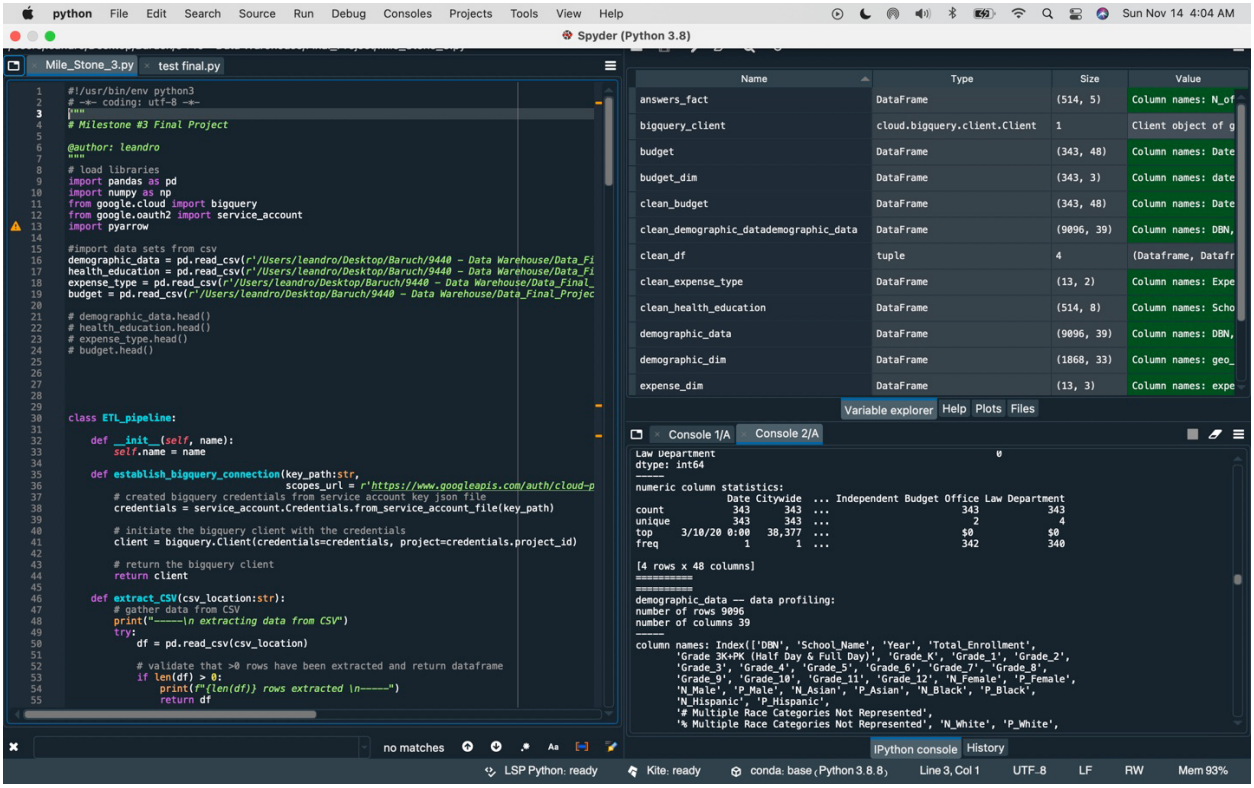
1. Downloaded CSV from NYC open data and saved it on my PC
2. I uploaded the csv into spider using python
3. Established connection with big query client
4. Perform data profiling
5. Perform data cleansing
6. Create dimension
7. Create fact
8. Load to Big Query

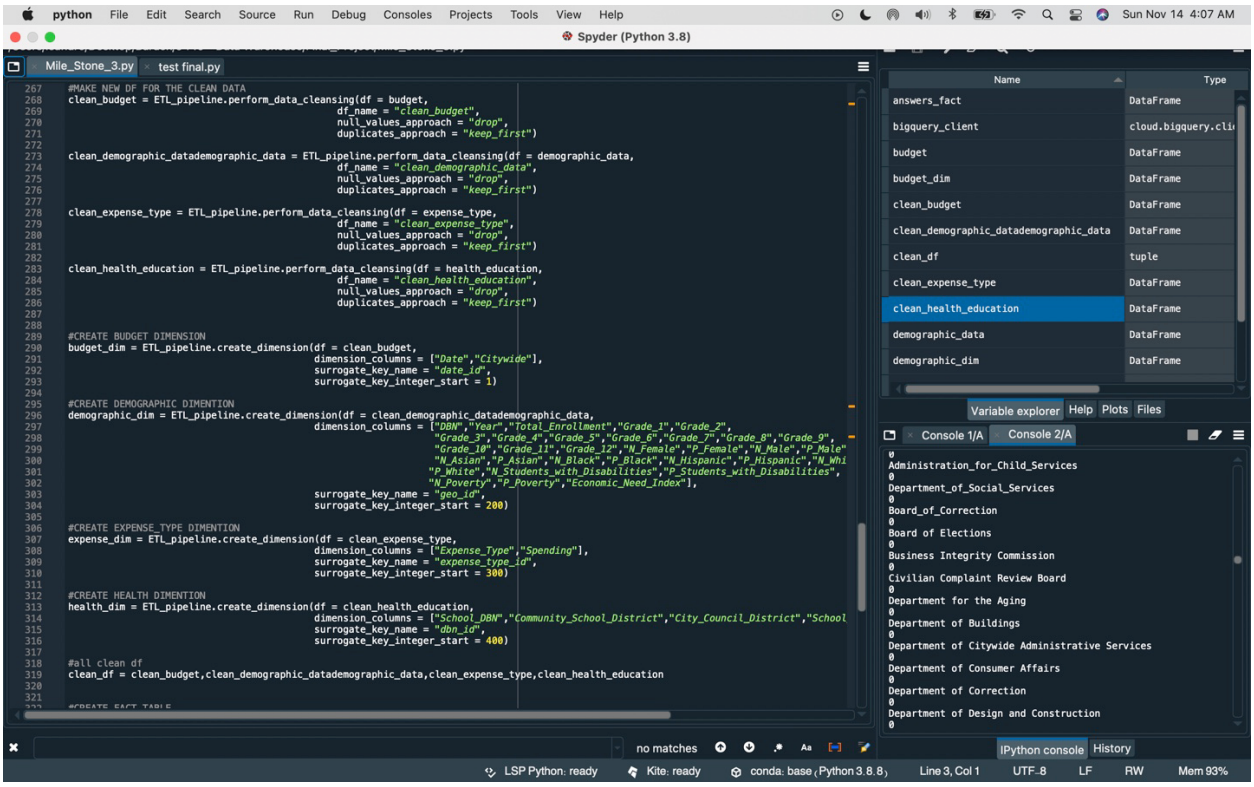
The steps were done using Prof. Michael O' Donell's code as guidance in order to perform the ETL process.




Safari File Edit View History Bookmarks Window Help
Sun Nov 14 4:00 AM
console.cloud.google.com
Week #7 – 2021 Fall Term (1) Data Warehous...
CIS9440 Fall 2021 Week 7 - YouTube
what's truncate - Google Search
SQL workspace – BigQuery – CIS9440 – Go...
Blackboard NY1 craigslist astoria at DuckDuckGo Astoria Post CUNY First The Verge -CNET Fancup UOL DNA BigQuery NYC Open Data DW_MID GitHub
Start your Free Trial with $300 in credit. Don't worry--you won't be charged if you run out of credits. Learn more
DISMISS
ACTIVATE
Google Cloud Platform
CIS9440
Search products and resources
FEATURES & INFO
SHORTCUT
DISABLE EDITOR TABS
Explorer
ADD DATA
Type to search
Viewing pinned projects.
cis9440-326622
Final
answers_fact
budget_dim
demographic_dim
expense_dim
health_dim
nba_data
week7
bigquery-public-data
BUDGET...
DEMOGR...
EXPENS...
HEALTH_...
COMPOSE NEW QUERY
demographic_dim
QUERY
SHARE
COPY
SNAPSHOT
DELETE
EXPORT
SCHEMA
DETAILS
PREVIEW
Row geo_id DBN Year Total_Enrollment Grade_1 Grade_2 Grade_3 Grade_4 Grade_5 Grade_6 Grade_7 Grade_8 Grade_9
1 213 01M292 2015-16 185 0 0 0 0 0 0 14 11 42 4
2 215 01M332 2015-16 110 0 0 0 0 0 39 35 36 0 0
3 220 01M450 2015-16 677 0 0 0 0 0 95 103 93 103 9
4 227 01M839 2015-16 371 0 0 0 0 0 127 124 120 0 0
5 242 02M104 2015-16 1,113 0 0 0 0 0 366 401 346 0 0
6 244 02M114 2015-16 441 0 0 0 0 0 149 142 150 0 0
7 249 02M131 2015-16 420 0 0 0 0 0 111 166 143 0 0
Results per page: 100
1 – 100 of many
JOB HISTORY
QUERY HISTORY
SAVED QUERIES


python File Edit Search Source Run Debug Consoles Projects Tools View Help
Sun Nov 14 4:02 AM
Spyder (Python 3.8)
/Users/leandro/Desktop/Baruch/9440 - Data Warehouse/Final_Project/Mile_Stone_3.py
Mile_Stone_3.py
test final.py
Warehouse/Final_Project
Name
Type
answers_fact
DataFrame
bigquery_client
budget
DataFrame
budget_dim
DataFrame
clean_budget
DataFrame
clean_demographic_datademographic_data
DataFrame
clean_df
tuple
clean_expense_type
DataFrame
clean_health_education
DataFrame
Variable explorer
Help
Plots
Files
Console 1/A
Console 2/A
cis9440-326622.Final.demographic_dim
loading expense_dim to BigQuery dataset Final
Loaded 156 rows and 3 columns to
cis9440-326622.Final.expense_dim
loading health_dim to BigQuery dataset Final
Loaded 5893 rows and 6 columns to
cis9440-326622.Final.health_dim
loading answers_fact to BigQuery dataset Final
Loaded 514 rows and 5 columns to
cis9440-326622.Final.answers_fact
In [118]:


python File Edit Search Source Run Debug Consoles Projects Tools View Help
Sun Nov 14 4:04 AM
Spyder (Python 3.8)
Mile_Stone_3.py
test final.py
Name
Type
Size
Value
answers_fact
DataFrame
(514, 5)
Column names: N_of
bigquery_client
cloud.bigquery.client.Client
1
Client object of g
budget
DataFrame
(343, 48)
Column names: Date
budget_dim
DataFrame
(343, 3)
Column names: date
clean_budget
DataFrame
(343, 48)
Column names: Date
clean_demographic_datademographic_data
DataFrame
(9096, 39)
Column names: DBN,
clean_df
tuple
4
(Dataframe, Datafr
clean_expense_type
DataFrame
(13, 2)
Column names: Expe
clean_health_education
DataFrame
(514, 8)
Column names: Scho
demographic_data
DataFrame
(9096, 39)
Column names: DBN,
demographic_dim
DataFrame
(1868, 33)
Column names: geo_
expense_dim
DataFrame
(13, 3)
Column names: expe
Variable explorer
Help
Plots
Files
Console 1/A
Console 2/A
Law Department 0
dtype: int64
numeric column statistics:
Date Citywide ... Independent Budget Office Law Department
count 343 343 ... 343 343
unique 343 343 ... 2 4
top 3/10/20 0:00 38,377 ... $0 $0
freq 1 1 ... 342 340
[4 rows x 48 columns]
demographic_data — data profiling:
number of rows 9096
number of columns 39
column names: Index(['DBN', 'School_Name', 'Year', 'Total_Enrollment',
'Grade 3K+PK (Half Day & Full Day)', 'Grade_K', 'Grade_1', 'Grade_2',
'Grade_3', 'Grade_4', 'Grade_5', 'Grade_6', 'Grade_7', 'Grade_8',
'Grade_9', 'Grade_10', 'Grade_11', 'Grade_12', 'N_Female', 'P_Female',
'N_Male', 'P_Male', 'N_Asian', 'P_Asian', 'N_Black', 'P_Black',
'N_Hispanic', 'P_Hispanic',
'# Multiple Race Categories Not Represented',
'% Multiple Race Categories Not Represented', 'N_White', 'P_White',
IPython console
History
no matches
LSP Python: ready
Kite: ready
conda: base (Python 3.8.8)
Line 3, Col 1
UTF-8
LF
RW
Mem 93%


python File Edit Search Source Run Debug Consoles Projects Tools View Help
Sun Nov 14 4:07 AM
Spyder (Python 3.8)
Mile_Stone_3.py
test final.py
#MAKE NEW DF FOR THE CLEAN DATA
clean_budget = ETL_pipeline.perform_data_cleansing(df = budget,
df_name = "clean_budget",
null_values_approach = "drop",
duplicates_approach = "keep_first")
clean_demographic_datademographic_data = ETL_pipeline.perform_data_cleansing(df = demographic_data,
df_name = "clean_demographic_data",
null_values_approach = "drop",
duplicates_approach = "keep_first")
clean_expense_type = ETL_pipeline.perform_data_cleansing(df = expense_type,
df_name = "clean_expense_type",
null_values_approach = "drop",
duplicates_approach = "keep_first")
clean_health_education = ETL_pipeline.perform_data_cleansing(df = health_education,
df_name = "clean_health_education",
null_values_approach = "drop",
duplicates_approach = "keep_first")
#CREATE BUDGET DIMENSION
budget_dim = ETL_pipeline.create_dimension(df = clean_budget,
dimension_columns = ["Date","Citywide"],
surrogate_key_name = "date_id",
surrogate_key_integer_start = 1)
#CREATE DEMOGRAPHIC DIMENTION
demographic_dim = ETL_pipeline.create_dimension(df = clean_demographic_datademographic_data,
dimension_columns = ["DBN","Year","Total_Enrollment","Grade_1","Grade_2",
"Grade_3","Grade_4","Grade_5","Grade_6","Grade_7","Grade_8","Grade_9",
"Grade_10","Grade_11","Grade_12","N_Female","P_Female","N_Male","P_Male"
"N_Asian","P_Asian","N_Black","P_Black","N_Hispanic","P_Hispanic","N_Whi
"P_White","N_Students_with_Disabilities","P_Students_with_Disabilities",
"N_Poverty","P_Poverty","Economic_Need_Index"],
surrogate_key_name = "geo_id",
surrogate_key_integer_start = 200)
#CREATE EXPENSE_TYPE DIMENTION
expense_dim = ETL_pipeline.create_dimension(df = clean_expense_type,
dimension_columns = ["Expense_Type","Spending"],
surrogate_key_name = "expense_type_id",
surrogate_key_integer_start = 300)
#CREATE HEALTH DIMENTION
health_dim = ETL_pipeline.create_dimension(df = clean_health_education,
dimension_columns = ["School_DBN","Community_School_District","City_Council_District","School
surrogate_key_name = "dbn_id",
surrogate_key_integer_start = 400)
#all clean df
clean_df = clean_budget,clean_demographic_datademographic_data,clean_expense_type,clean_health_education
Name
Type
answers_fact DataFrame
bigquery_client cloud.bigquery.cli
budget DataFrame
budget_dim DataFrame
clean_budget DataFrame
clean_demographic_datademographic_data DataFrame
clean_df tuple
clean_expense_type DataFrame
clean_health_education DataFrame
demographic_data DataFrame
demographic_dim DataFrame
Variable explorer Help Plots Files
Console 1/A Console 2/A
0
Administration_for_Child_Services
0
Department_of_Social_Services
0
Board_of_Correction
0
Board of Elections
0
Business Integrity Commission
0
Civilian Complaint Review Board
0
Department for the Aging
0
Department of Buildings
0
Department of Citywide Administrative Services
0
Department of Consumer Affairs
0
Department of Correction
0
Department of Design and Construction
0
no matches
IPython console History
LSP Python: ready
Kite: ready
conda: base (Python 3.8.8)
Line 3, Col 1
UTF-8
LF
RW
Mem 93%

The Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
# Milestone #3 Final Project

@author: leandro
"""
# load libraries
import pandas as pd
import numpy as np
from google.cloud import bigquery
from google.oauth2 import service_account
import pyarrow

#import data sets from csv
demographic_data = pd.read_csv(r'/Users/leandro/Desktop/Baruch/9440 - Data
Warehouse/Data_Final_Project/2019-20_Demographic_Snapshot_-_School.csv')
health_education = pd.read_csv(r'/Users/leandro/Desktop/Baruch/9440 - Data
Warehouse/Data_Final_Project/2019-2020_Local_Law_14_Health_Education_Report_-
_Final.csv')
expense_type = pd.read_csv(r'/Users/leandro/Desktop/Baruch/9440 - Data
Warehouse/Data_Final_Project/Independent_Budget_Office__NYC_COVID_19_Cumulative_Sp
ending_by_Expense_Type.csv')
budget = pd.read_csv(r'/Users/leandro/Desktop/Baruch/9440 - Data
Warehouse/Data_Final_Project/Independent_Budget_Office__NYC_COVID_19_Spending_by_D
ate_-_Citywide_and_by_Agency.csv')

# demographic_data.head()
# health_education.head()
# expense_type.head()
# budget.head()




class ETL_pipeline:

    def __init__(self, name):
        self.name = name

    def establish_bigquery_connection(key_path:str,
```

```
                        scopes_url = r'https://www.googleapis.com/auth/cloud-platform'):
    # created bigquery credentials from service account key json file
    credentials = service_account.Credentials.from_service_account_file(key_path)

    # initiate the bigquery client with the credentials
    client = bigquery.Client(credentials=credentials, project=credentials.project_id)

    # return the bigquery client
    return client

def extract_CSV(csv_location:str):
    # gather data from CSV
    print("-----\n extracting data from CSV")
    try:
        df = pd.read_csv(csv_location)

        # validate that >0 rows have been extracted and return dataframe
        if len(df) > 0:
            print(f"{len(df)} rows extracted \n-----")
            return df

        # if data extraction fails print failure
        else:
            print(f"{csv_location} has 0 rows of data")

    # if data extraction fails print failure
    except:
        print(f"{csv_location} extraction failed")

def perform_data_profiling(df,
                           df_name:str):
    # --Make more robust!
    # Some basic data profiling steps/ideas here...
    print('='*10)
    print(f"{df_name} -- data profiling:")
    print(f"number of rows {df.shape[0]}")
    print(f"number of columns {df.shape[1]}")
    print('-'*5)
    print(f"column names: {df.columns}")
    print('-'*5)
    print("column data types:")
    for i in df.columns:
        print(f"{i} -- dtype: {df[i].dtypes}")
    print('-'*5)
```

```
    print("Columns with null values:")
    print(df.isnull().sum())
    print('-'*5)
    print("numeric column statistics:")
    print(df.describe())
    print('='*10)

def perform_data_cleansing(df,
                           df_name:str,
                           null_values_approach = "drop",
                           duplicates_approach = "keep_first",
                           outliers_approach = 3,
                           outlier_columns = None):
    print('='*10)
    print(f"cleaning {df_name} \n-----")

    # remove or replace Null values
    print(f"{df_name} Null values cleaning method: {null_values_approach}")
    if null_values_approach == "drop":
        print(f"dropping {len(df[df.isna().any(axis = 1)])} Null rows")
        df = df.dropna()

    elif null_values_approach == "threshold_2":
        df = df.dropna(thresh = 2)

    elif null_values_approach == "fill_0":
        df = df.fillna(0)

    elif null_values_approach == "ffill":
        df = df.fillna(method = 'ffill')

    # handle duplicates
    print(f"-----\n{df_name} duplicate values cleaning method: {duplicates_approach}")
    if duplicates_approach == "keep_first":
        print(f"dropping {df.duplicated().sum()} duplicate rows")
        df.drop_duplicates(keep = "first")

    elif duplicates_approach == "keep_last":
        df.drop_duplicates(keep = "last")

    # handle outliers
    if type(outliers_approach) == int and outlier_columns is not None:
        print(f"{df_name} outliers cleaning method: abs {outliers_approach}")
        from scipy import stats
```

```
        for c in outlier_columns:
            if df[c].dtypes == "int64" or df[c].dtypes == "float64":
                df = df[(np.abs(stats.zscore(df[c])) < outliers_approach)]
    print('='*10)
    # return cleaned dataframe
    return df

  def create_dimension(df,
                dimension_columns:list,
                surrogate_key_name:str,
                surrogate_key_integer_start:int):

    print(f"-----\n creating dimension: {dimension_columns[0]}_dim")
    # copy full dataframe to create dimension from subset
    dim = df.copy()
    dim = dim[dimension_columns]

    # drop unneeded rows in hierarchy
    dim = dim.drop_duplicates(subset=[dimension_columns[0]], keep = "first")

    # add surrogate key
    dim.insert(0, surrogate_key_name, range(surrogate_key_integer_start,
                                surrogate_key_integer_start+len(dim)))

    print(f"dimension {dimension_columns[0]}_dim created with {len(dim)} rows \n-----")
    # return the dimension as a dataframe
    return dim

  def create_date_dimension(client,
                    start_year:int,
                    end_year:int):

    date_query = f"""SELECT
              d AS full_date,
              CONCAT
(FORMAT_DATE("%Y",d),FORMAT_DATE("%m",d),FORMAT_DATE("%d",d)) as date_id,
              EXTRACT(YEAR FROM d) AS year,
              EXTRACT(WEEK FROM d) AS year_week,
              EXTRACT(DAY FROM d) AS year_day,
              CONCAT (EXTRACT(YEAR FROM d), EXTRACT(WEEK FROM d)) AS
year_week_concat,
              CONCAT (EXTRACT(YEAR FROM d), EXTRACT(MONTH FROM d)) AS
year_month_concat,
              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('{start_year}-01-01', '{end_year}-01-01',
INTERVAL 1 DAY)) AS d )"""

    # gather data from bigquery to dataframe
    try:
      dim = client.query(date_query).to_dataframe()

    # if data extraction fails print failure
    except:
      print("creating date dimension failed")

    print(f"-----\n creating date dimension from {start_year} to {end_year}")
    print(f"date dimension created with {len(dim)} rows \n-----")
    # return the dimension as a dataframe
    return dim

  def create_fact(df,
          dimensions:list,
          date_column = None):

    print(f"-----\n creating fact table")
    # copy full dataframe to create fact table
    fact = df.copy()

    # for every dimension, add the FK to the Fact table and remove hierarchy
    for d in dimensions:

      fact = fact.merge(d[[d.columns[0], d.columns[1]]],
                left_on = d.columns[1],
                right_on = d.columns[1],
                how = 'left')

      for c in range(1, len(d.columns)):
```

```
            fact = fact.drop(d.columns[c], 1)

        # create date_id column in fact table to match date_dim
        if date_column != None:

            import datetime as dt
            fact["date_id"] = df[date_column].dt.strftime("%Y%m%d")

            if date_column != "date_id":

                fact = fact.drop(date_column, 1)

        # return fact table as a dataframe
        print(f"fact table created with {len(fact)} rows \n-----")
        return fact

    def load_table_to_bigquery(bq_client,
                               table,
                               dataset_name:str,
                               table_name:str):

        print(f"-----\n loading {table_name} to BigQuery dataset {dataset_name}")

        # define bigquery client
        client = bq_client

        # define location you will upload table to in bigquery
        table_ref = client.dataset(dataset_name).table(table_name)

        # configure load job settings
        job_config = bigquery.LoadJobConfig()
        job_config.autodetect = True
        job_config.source_format = bigquery.SourceFormat.CSV
        #job_config.write_disposition = "WRITE_TRUNCATE"

        # initiate load job
        load_job = client.load_table_from_dataframe(table, table_ref,
                                        job_config=job_config)
        load_job.result()

        # Make a BigQuery API request to check if new table was loaded successfully
        validate_table = client.get_table(table_ref)  # Make an API request.
        print(f"Loaded {validate_table.num_rows} rows and {len(validate_table.schema)} columns
to {table_ref} \n-----")
```

```
#CONNECTED TO BIG QUERY
bigquery_client = ETL_pipeline.establish_bigquery_connection(key_path =
r'/Users/leandro/Desktop/Baruch/9440 - Data Warehouse/Final_Project/cis9440-326622-
5b7c41b08b65.json')

#PERFORM DATA PROFILING
ETL_pipeline.perform_data_profiling(df = budget,
                        df_name = "budget")

ETL_pipeline.perform_data_profiling(df = demographic_data,
                        df_name = "demographic_data")

ETL_pipeline.perform_data_profiling(df = expense_type,
                        df_name = "expense_type")

ETL_pipeline.perform_data_profiling(df = health_education,
                        df_name = "health_education")

#MAKE NEW DF FOR THE CLEAN DATA
clean_budget = ETL_pipeline.perform_data_cleansing(df = budget,
                                df_name = "clean_budget",
                                null_values_approach = "drop",
                                duplicates_approach = "keep_first")

clean_demographic_datademographic_data = ETL_pipeline.perform_data_cleansing(df =
demographic_data,
                                df_name = "clean_demographic_data",
                                null_values_approach = "drop",
                                duplicates_approach = "keep_first")

clean_expense_type = ETL_pipeline.perform_data_cleansing(df = expense_type,
                                df_name = "clean_expense_type",
                                null_values_approach = "drop",
                                duplicates_approach = "keep_first")

clean_health_education = ETL_pipeline.perform_data_cleansing(df = health_education,
                                df_name = "clean_health_education",
                                null_values_approach = "drop",
                                duplicates_approach = "keep_first")


#CREATE BUDGET DIMENSION
budget_dim = ETL_pipeline.create_dimension(df = clean_budget,
```

```
                        dimension_columns = ["Date","Citywide"],
                        surrogate_key_name = "date_id",
                        surrogate_key_integer_start = 1)

#CREATE DEMOGRAPHIC DIMENTION
demographic_dim = ETL_pipeline.create_dimension(df =
clean_demographic_datademographic_data,
                        dimension_columns =
["DBN","Year","Total_Enrollment","Grade_1","Grade_2",

"Grade_3","Grade_4","Grade_5","Grade_6","Grade_7","Grade_8","Grade_9",

"Grade_10","Grade_11","Grade_12","N_Female","P_Female","N_Male","P_Male",

"N_Asian","P_Asian","N_Black","P_Black","N_Hispanic","P_Hispanic","N_White",

"P_White","N_Students_with_Disabilities","P_Students_with_Disabilities",
                                    "N_Poverty","P_Poverty","Economic_Need_Index"],
                        surrogate_key_name = "geo_id",
                        surrogate_key_integer_start = 200)

#CREATE EXPENSE_TYPE DIMENTION
expense_dim = ETL_pipeline.create_dimension(df = clean_expense_type,
                        dimension_columns = ["Expense_Type","Spending"],
                        surrogate_key_name = "expense_type_id",
                        surrogate_key_integer_start = 300)

#CREATE HEALTH DIMENTION
health_dim = ETL_pipeline.create_dimension(df = clean_health_education,
                        dimension_columns =
["School_DBN","Community_School_District","City_Council_District","School_Name",],
                        surrogate_key_name = "dbn_id",
                        surrogate_key_integer_start = 400)

#all clean df
clean_df =
clean_budget,clean_demographic_datademographic_data,clean_expense_type,clean_health_e
ducation


#CREATE FACT TABLE
answers_fact = ETL_pipeline.create_fact (df = clean_budget,
                        dimensions=[budget_dim])
```

```
answers_fact = ETL_pipeline.create_fact (df = clean_demographic_datademographic_data,
                        dimensions=[demographic_dim])

answers_fact = ETL_pipeline.create_fact (df = clean_expense_type,
                        dimensions=[expense_dim])

answers_fact = ETL_pipeline.create_fact (df = clean_health_education,
                        dimensions=[health_dim])


#load the budget_dim to BigQuery
ETL_pipeline.load_table_to_bigquery(bigquery_client,
                        budget_dim,
                        dataset_name = "Final",
                        table_name = "budget_dim")

#load the demographic_dim to BigQuery
ETL_pipeline.load_table_to_bigquery(bigquery_client,
                        demographic_dim,
                        dataset_name = "Final",
                        table_name = "demographic_dim")

#load the expense_dim to BigQuery
ETL_pipeline.load_table_to_bigquery(bigquery_client,
                        expense_dim,
                        dataset_name = "Final",
                        table_name = "expense_dim")

#load the health_dim to BigQuery
ETL_pipeline.load_table_to_bigquery(bigquery_client,
                        health_dim,
                        dataset_name = "Final",
                        table_name = "health_dim")

#load the FACT answers_fact to BigQuery
ETL_pipeline.load_table_to_bigquery(bigquery_client,
                        answers_fact,
                        dataset_name = "Final",
                        table_name = "answers_fact")
```