Authentication - you will need to upload the service account key json file when prompted

```
from google.colab import auth
auth.authenticate_service_account()
```

```
Successfully saved credentials for pipelineauth@weatherlink-404323.iam.gservi
```

+ Code + Text

Testing querying data from GBQ now that we are authenticated:

```
from google.cloud import bigquery
project_name='weatherlink-404323'
client = bigquery.Client(project=project_name)

dataset_name = 'weatherlink_master'
table_name = 'census'

sql_query = (f"SELECT * FROM {dataset_name}.{table_name} LIMIT 10")

df = client.query(sql_query).to_dataframe()

print(df)
```

```
  geo_id income_per_capita median_income  year
0  36079            106871         48417  2021
1  47065             66096         38668  2021
2  12097             60585         26789  2021
3   9003             80069         43636  2021
4  42017            100144         50607  2021
5  34029             75719         39055  2021
6  18089             61443         31785  2021
7   6067             80063         37259  2021
8  49011             93182         38879  2021
9  22071             46942         35587  2021
```

Analyzing some of the census data, I want to see what the number of columns in each year are to see if they differ.

```
accident_client = bigquery.Client()

for i in range(2007,2024):
  table_name = f"bigquery-public-data.census_bureau_acs.county_{i}_1yr"


  sql_query = (f"SELECT * FROM {table_name} LIMIT 1")

  try:
    df = accident_client.query(sql_query).to_dataframe()
    print (f"Table name {table_name} has {(df.shape[1])} cols")

  #print(df)
  except:
    print (f"Table name {table_name} was not found :( ")
```

```
Table name bigquery-public-data.census_bureau_acs.county_2007_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2008_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2009_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2010_1yr has 247 cols
Table name bigquery-public-data.census_bureau_acs.county_2011_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2012_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2013_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2014_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2015_1yr has 246 cols
Table name bigquery-public-data.census_bureau_acs.county_2016_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2017_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2018_1yr has 252 cols
Table name bigquery-public-data.census_bureau_acs.county_2019_1yr has 245 cols
Table name bigquery-public-data.census_bureau_acs.county_2020_1yr was not foun
```

Okay, now I want to see WHICH columns everything does not have, so I can see what we would be missing out on if we just ignore columns that are not common across all data sets.

```
import pandas as pd

year_start = 2007
year_stop = 2025

# Initialize a dictionary to store the schema for each table
table_schemas = {}
```

```
# Initialize a set to store common columns
common_columns_set = None

# Initialize a dictionary to store unique columns for each table
unique_columns_dict = {}

# Initialize a list to store DataFrames for each year
dfs = []

for i in range(year_start, year_stop):
    table_name = f"bigquery-public-data.census_bureau_acs.county_{i}_1yr"

    try:
        # Fetch the schema (column information) for each table
        if table_name not in table_schemas:
            table = accident_client.get_table(table_name)
            table_schemas[table_name] = set([field.name for field in table.schema

        # If this is the first DataFrame, initialize the set with its columns
        if common_columns_set is None:
            common_columns_set = set(table_schemas[table_name])
        else:
            # Update the set to include only columns present in both DataFrames
            common_columns_set.intersection_update(table_schemas[table_name])

        # Update the set to include only columns not present in other DataFrames
        unique_columns_dict[table_name] = table_schemas[table_name].difference(co

        print(f"Table name {table_name} has {len(table_schemas[table_name])} cols

        # Fetch the data and add it to a DataFrame
        query = f"SELECT * FROM {table_name}"
        df = accident_client.query(query).to_dataframe()

        # Add the 'year' column to the DataFrame
        df['year'] = i

        # Keep only the common columns
        df = df[list(common_columns_set) + ['year']]

        # Append the DataFrame to the list
        dfs.append(df)

    except:
        print(f"Table {table_name} not found")

# Print unique columns for each table
for i in range(year_start, year_stop):
    table_name = f"bigquery-public-data.census_bureau_acs.county_{i}_1yr"
    try:
        print(f"Unique columns in {table_name}: {unique_columns_dict[table_name]}
    except:
        print(f"Table {table_name} not found")

# Concatenate all DataFrames into a master DataFrame
master_df = pd.concat(dfs, ignore_index=True)

# Print the master DataFrame
print("\nMaster DataFrame:")
print(master_df)
```

```
Table name bigquery-public-data.census_bureau_acs.county_2007_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2008_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2009_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2010_1yr has 247 cc
Table name bigquery-public-data.census_bureau_acs.county_2011_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2012_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2013_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2014_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2015_1yr has 246 cc
Table name bigquery-public-data.census_bureau_acs.county_2016_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2017_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2018_1yr has 252 cc
Table name bigquery-public-data.census_bureau_acs.county_2019_1yr has 245 cc
Table bigquery-public-data.census_bureau_acs.county_2020_1yr not found
Table name bigquery-public-data.census_bureau_acs.county_2021_1yr has 245 cc
Table bigquery-public-data.census_bureau_acs.county_2022_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2023_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2024_1yr not found
Unique columns in bigquery-public-data.census_bureau_acs.county_2007_1yr: se
Unique columns in bigquery-public-data.census_bureau_acs.county_2008_1yr: se
Unique columns in bigquery-public-data.census_bureau_acs.county_2009_1yr: se
Unique columns in bigquery-public-data.census_bureau_acs.county_2010_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2011_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2012_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2013_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2014_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2015_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2016_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2017_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2018_1yr: {'
Unique columns in bigquery-public-data.census_bureau_acs.county_2019_1yr: {'
Table bigquery-public-data.census_bureau_acs.county_2020_1yr not found
Unique columns in bigquery-public-data.census_bureau_acs.county_2021_1yr: {'
Table bigquery-public-data.census_bureau_acs.county_2022_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2023_1yr not found
Table bigquery-public-data.census_bureau_acs.county_2024_1yr not found

Master DataFrame:
      income_per_capita  two_cars pop_5_years_over occupation_services  \
0               24635.0   94225.0              NaN             48502.0
1               22853.0   68027.0              NaN             37994.0
2               37651.0  125047.0              NaN             74770.0
3               21887.0   50863.0              NaN             60063.0
4               27268.0  106823.0              NaN             62354.0
...                 ...       ...              ...                 ...
11503           41636.0   55639.0              NaN             39743.0
11504           30473.0   60722.0              NaN             35711.0
11505           20943.0   51464.0              NaN             38167.0
11506           37681.0   89537.0              NaN             42335.0
11507           35375.0  113759.0              NaN             55175.0

      owner_occupied_housing_units female_75_to_79   no_car  pop_divorced  \
0                         162734.0          7125.0   6003.0       48781.0
1                         130322.0          7678.0   5750.0       40576.0
2                         213786.0          7548.0  31819.0       83851.0
3                         119820.0          8177.0  44853.0       50839.0
4                         189723.0         10108.0   6811.0       75202.0
...                            ...             ...      ...           ...
```

Okay, so we can see there are indeed some columns that are NOT in all of the data sets. We now have a master dataframe of ALL the census data, with only the columns from every set. (There is a gap from the year 2020, where no census data was generated due to the COVID-19 pandemic)

Since 253! has 500 digits, we are not going to be making scatter plots of every column vs every other column. Instead, we are going to upload this dataframe to Google Big Query in our 'Data Warehouse', and then select a number of columns to compare for analysis.

```
pip install pandas_gbq
```

```
from pandas_gbq import to_gbq
from pandas_gbq.schema import generate_bq_schema
from io import StringIO

client = bigquery.Client()

project_id='weatherlink-404323'
dataset_id = 'weatherlink_master'
table_id = 'census_master'

# This is a weird work around to get the dataframe acceptable for upload
# temporarily store the dataframe as a csv in a string variable
temp_csv_string = master_df.to_csv(sep=";", index=False)
temp_csv_string_IO = StringIO(temp_csv_string)
# create new dataframe from string variable
new_df = pd.read_csv(temp_csv_string_IO, sep=";")

to_gbq(new_df, f"{dataset_id}.{table_id}", project_id=project_id, if_exists='repl
```

```
    100%|██████████| 1/1 [00:00<00:00, 1348.22it/s]
```

Lets test by querying our new master table

```
sql = f" SELECT COUNT(*) FROM {project_id}.{dataset_id}.{table_id}"

result = client.query(sql)

print(result)
```

```
    QueryJob<project=weatherlink-404323, location=US, id=a28bd2f3-62ce-40c4-9140-
```