ETL Report

Group 2 – Retail/Operations/Commerce

Adam Brewer, Emily Atkinson, Nolan Thomas, Stephanie Leiva

Introduction:

With our team being provided the direction of Retail, Operations, & Commerce, we made the decision to focus on the government's **Paycheck Protection Program (PPP)** first initiated in April of 2020. To conduct research of the PPP, our group accessed information from the Small Business Administration to review details of the individual PPP loans distributed. We then combined this information with the Census Annual Business Survey - Company Summary API to gain a more holistic view of the demographics of the businesses, industries, and individuals who received PPP loans.

To provide context on how the PPP loan affected portions of the economy, our group pulled in Unemployment Data from the Bureau of Labor Statistics aggregated at the state level to view potential correlations between the number of loans being provided and the trends in the Unemployment numbers by state.

Data Sources:

Small Business Administration. (July 4, 2022). Paycheck Protection Program - Freedom of Information Act. Retrieved September 19, 2022 from data.sba.gov website: https://data.sba.gov/dataset/ppp-foia

United States Census Bureau. (October 28, 2021). Annual Business Survey. Retrieved September 20, 2022 from www.census.gov website: https://www.census.gov/data/developers/data-sets/abs.html

United States Department of Labor. (July 7, 2022). Unemployment Insurance Weekly Claims Data. https://oui.doleta.gov/unemploy/claims.asp

Extraction:

Unemployment Data:

1. Download an Excel spreadsheet from the OUI website (https://oui.doleta.gov/unemploy/claims.asp) with the following specifications:
    a. Select = State
    b. Beginning Year = 2019
    c. Ending Year = 2020
    d. Output Format = Spreadsheet
    e. State = Select all
2. Convert the spreadsheet into a CSV.
3. Place the raw CSV file into the relevant storage container.

Annual Business Survey (ABS) 2020 Company Summary:

1. Since this is an API call, the extraction happens with the transformation. See Transformation steps.

Paycheck Protection Program Data:

1. Download all 13 CSV datasets from the following website (https://data.sba.gov/dataset/ppp-foia)
2. Place the raw CSV files into the relevant storage container.

Transformation:

All transformation/cleaning will happen in one ipynb notebook in a databrick. You can find the databrick in its entirety here.

1. Connect to the appropriate storage account and storage container that holds all raw CSVs.
2. Create a mount point to this container.
3. Import the following at the start of the notebook:
    a. import uuid
    b. from confluent_kafka.admin import AdminClient, NewTopic
    c. from pyspark.sql.functions import concat, col, lit,isnan,when,count, udf
    d. from pyspark.sql.types import FloatType, IntegerType
    e. from pyspark.sql.types import StringType
    f. from pyspark.sql.functions import to_date
    g. import pandas as pd
    h. from time import sleep

Annual Business Survey (ABS) 2020 Company Summary:

1. Import the pandas and requests python modules.
    https://pandas.pydata.org/
    https://requests.readthedocs.io/en/latest/

2. Request the API key for the Census API for the Github owner or go to the census website and request a key of your own. Save this as variable ‘apikey’

   https://api.census.gov/data/key_signup.html

3. Request data from the ABS 2020 Company Summary for state data (link to data below) using the request.get() command. Transform the requested data with the .json() requests function so the data in output in a list of lists.

   "https://api.census.gov/data/2018/abscs?get=NAME,NAICS2017,SEX,SEX_LABEL,RACE_GROUP,RACE_GROUP_LABEL,ETH_GROUP,ETH_GROUP_LABEL,FIRMPDEMP&for=state:*&key={apikey}"

4. Turn the requested data from the census into a spark dataframe using command spark.createDataFrame() called df_company_summary_State. Set the first list of the requested data equal to ‘columns’ in the pandas command to create column headers. Then let the remaining lists of data be set to the values associated with the headers.

```
df_company_summary_State =
spark.createDataFrame(company_summary_list_of_lists[1:], columns =
company_summary_list_of_lists[0])
```

5. Drop “SEX” and “state” columns using the spark command:

```
df_company_summary_State = df_company_summary_State.drop(* [“SEX” , ”state”]
```

6. Rename columns in the data frame to more meaningful header titles so the data frame is easier to understand. The column names to change can be seen below, using the spark command:

```
 df_company_summary_State  = df_company_summary_State.withColumnsRenamed
(“old_column_name” ,” new_column_name”).
        "NAME" -> "State"
         "NAICS2017" ->  "NAICSCode"
        "SEX_LABEL" -> "Sex"
         "RACE_GROUP_LABEL" -> "Race"
         "ETH_GROUP_LABEL" -> "Ethnicity"
        "FIRMPDEMP" -> "NumberOfBusinesses"
```

7. Add in a column based on the NAICS code for the associated Industry it defines. This can be done using the Industry_Adder() function (seen below):

```
def Industry_Adder(df):
  # dataframe df must have NAICS codes column named 'NAICSCode'

  NAICS_to_Industry = {"11":"Agriculture, Forestry, Fishing and Hunting",
             "21":"Mining",
             "22": "Utilities",
             "23":"Construction",
             "31":"Manufacturing",
                        "32":"Manufacturing",
                        "33":"Manufacturing",
                        "42":"Wholesale Trade",
                        "44":"Retail Trade",
                        "45":"Retail Trade",
                        "48":"Transportation and Warehousing",
                        "49":"Transportation and Warehousing",
                        "51":"Information",
                        "52":"Finance and Insurance",
                        "53":"Real Estate Rental and Leasing",
                        "54":"Professional, Scientific, and Technical Services",
                        "55":"Management of Companies and Enterprises",
                        "56":"Administrative and Support and Waste Management and
Remediation Services",
                        "61":"Educational Services",
                        "62":"Health Care and Social Assistance",
                        "71":"Arts, Entertainment, and Recreation",
                        "72":"Accommodation and Food Services",
                        "81":"Other Services (except Public Administration)",
                        "92":"Public Administration"}

    udf_slice_2 = udf(lambda x : x.strip()[:2])
    df = df.withColumn("Industry", udf_slice_2(col("NAICSCode")))

      NAICS_Code_values = df[["Industry"]].distinct()
    range_rows = NAICS_Code_values.count()
    keys = list(NAICS_to_Industry.keys())
    remove_list = [str(i).zfill(2) for i in range(0, 100)]
    for k in keys:
        remove_list.remove(k)
    for i in range(1,11):
        remove_list.append(str(i))

    for i in range(range_rows):
        value = NAICS_Code_values.collect()[i][0].strip()
        if(value not in keys):
            remove_list.append(value)

    remove_list = list(set(remove_list))
    for j in remove_list:
        df = df.where(df.Industry != j)

    udf_dict = udf(lambda x : NAICS_to_Industry[x])
    df = df.withColumn("Industry", udf_dict(col("Industry")))
    return df
```

8. Drop the following columns ["RACE_GROUP", "ETH_GROUP", "NAICSCode"] using the spark command format:

    ```
    df_company_summary_State = df_company_summary_State.drop(*
    [list_of_columns_to_remove])
    ```

9. Remove aggregate rows from the remaining columns using the following command format:

    ```
     df_company_summary_State = df_company_summary_State. Where
    (df_company_summary_State.column_name != ‘row_item_to_remove’ )
    ```

    i. Column Name: Sex
        1. Remove rows with name: Total, Equally male/female
    ii. Column Name: Race
        1. Remove rows with name: Total, Minority, Nonminority, Equally minority/nonminority
    iii. Column Name: Ethnicity
        1. Remove rows with name: Total, Equally Hispanic/non-Hispanic

10. Format the data frame by ordering the data columns and sort the values by ascending values. Then reset the index and drop remove the previous index column. This can all be done using the code format below:

    ```
    df_company_summary_State = df_company_summary_State.select(*"State", "Industry",
    "Race", "Ethnicity","Sex", "NumberOfBusinesses")

    df_company_summary_State = df_company_summary_State.sort("State", "Industry",
    "Race", "Ethnicity","Sex", "NumberOfBusinesses")
    ```

11. Reconfigure the "NumberOfBusinesses" column by setting the column type to an integer. This can be done using the code format:

    ```
     df_company_summary_State =
    df_company_summary_State.withColumn(“NumberOfBusinesses”,
    dataframe_name[“NumberOfBusinesses”].cast(IntegerType())
    ```

12. See Exporting All Cleaned CSVs below; last transformation step.

Unemployment Data:

1. Import the ‘unemployment.csv’ using PySpark called ‘unemployment_df’
2. Change the headers to correct header names that are displayed in row 4, with no spaces between words using CamelCasing. EX: ‘Filed week ended’ → ‘FiledWeekEnded’. Create dummy names for the final 3 columns that will be deleted later.
    a. Do so by setting the variable ‘headers’ to equal a list of NoSpaced CamelCased column names (with three dummy names at the end) then using this code:

```
headers = ['State', 'FiledWeekEnded', 'InitialClaims','ReflectingWeekEnded',
'ContinuedClaims', 'CoveredEmployment', 'InsuredUnemploymentRate', 'null1', 'null2',
'null3']

unemployment_df = unemployment_df.toDF(*headers)
```

3. Drop rows in the 'State' column that are null. Drop rows in the 'State' column that contain the word 'State'.

```
unemployment_df = unemployment_df.where(unemployment_df.State.isnotNull())
unemployment_df = unemployment_df.where(unemployment_df.State != ‘State’)
```

4. Set the data frame to exclude the last three columns.
5. Drop the row in the ‘State’ column that contains 'Run Date d/mm/yyyy' in it. This will be a different date depending on when the csv was run so check the 'State' column value_counts to see what date this would be. Only keep rows in the 'State' column that are state names.

```
unemployment_df = unemployment_df.where(unemployment_df.State != ‘Run Date:
9/20/222’)
```

6. Replace all commas (,) in 'Initial Claims', 'Continued Claims', and 'Covered Employment’ by replacing with nothing:

```
from pyspark.sql import DataFrame, SparkSession, types, functions as F
dataframe_name = dataframe_name.withColumn(‘column_name’,
F.regexp_replace(‘column_name’, ‘,’, ‘’)
```

   a. To be clear, it’s (‘column_name’ , ‘ comma_symbol’ , ‘ empty‘)

7. Convert the 'Initial Claims', 'Continued Claims', 'Covered Employment’, and 'InsuredUnemploymentRate' columns to numeric/cast as Floats. Use the following code and replace appropriate column names:

```
unemployment_df = unemployment_df.withColumn("column_name",
unemployment_df["column_name"].cast(FloatType()))
```

8. Convert the 'FiledWeekEnded' and 'ReflectingWeekEnded' to date type.Use the following code and replace appropriate column names:

```
spark.sql(“set spark.sql.legacy.timeParserPolicy = LEGACY”)
unemployment_df = unemployment_df.withColumn(‘column_name’,
to_date(unemployment_df.column_name, ‘d/m/yyyy/’)
```

9. See Exporting All Cleaned CSVs below; last transformation step.

Paycheck Protection Program Data:

1. Import CSV using PySpark. Use a for loop to run through (1-13) for the csv title names, union them together.

```
PPP_ =
spark.read.format("csv").load("/mnt/group-2-capstone/csvCapture/public_150k_plus_2
20703.csv", header=True, inferSchema='True')
for i in range(1,13):
  print(i)
  PPPx =
spark.read.format("csv").load(f"/mnt/group-2-capstone/csvCapture/public_up_to_150k
_{i}_220703.csv", header=True, inferSchema='True')
  PPP_ = PPP_.union(PPPx)
```

2. Make a copy with a slightly different name in case you run into errors:PPP = PPP_.select("*")
3. Set the data frame to exclude the following columns/drop these columns: SBAOfficeCode, ProcessingMethod, Term, SBAGuarantyPercentage, UndisbursedAmount, CD, Veteran, ServicingLenderLocationID, RuralUrbanIndicator, NonProfit, BorrowerAddress, BorrowerCity, BorrowerZip, ServicingLenderAddress, ServicingLenderCity, ServicingLenderZip, ProjectCity, ProjectCountyName, ProjectState, ProjectZip, OriginatingLenderCity, OriginatingLenderID
4. Remove nulls from the NAICSCode column. Null removal code is listed earlier in this ETL, just replace appropriate dataframe name and column name.
5. Convert 'DateApproved', 'LoanStatusDate', 'ForgivenessDate' columns to datetime type. Converting to date type code is listed earlier in this ETL, just replace appropriate dataframe name and column name.
6. Remove any columns with over 50% null values except the PROCEED columns like RENT_PROCEED, PAYROLL_PROCEED, etc. Following function:

```
def dropNullColumns(df):
  """
  This function drops columns containing all null values.
  :param df: A PySpark DataFrame
  """
  # unwanted columns list
  col_to_drop= []
  # count all rows
  total_rows = df.count()
  # counts nulls before running for loop
  null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in
df.columns]).collect()[0].asDict()
  # list of columns
  columns = list(null_counts.keys())

  for column in columns:
        if(column == 'UTILITIES_PROCEED' or column == 'PAYROLL_PROCEED' or column ==
'MORTGAGE_INTEREST_PROCEED' or column == 'RENT_PROCEED' or column ==
'REFINANCE_EIDL_PROCEED' or column == 'HEALTH_CARE_PROCEED' or column ==
'DEBT_INTEREST_PROCEED'):
            continue

        else:
            null_value = null_counts[f"{column}"]
            percent_null = null_value/total_rows
            if(percent_null > .5):
                col_to_drop.append(column)

    df = df.drop(*col_to_drop)  # drop all unwanted columns

    return df
```

7. Drop the entire 'LoanStatusDate' column.
8. Drop the rows with null values in the following columns but DO NOT drop nulls from any PROCEED columns: ServicingLenderName, ForgivenessAmount, BusinessType, JobsReported, BusinessAgeDescription, LMIIndicator, BorrowerState, BorrowerName.
9. Add in a column based on the NAICS code for the associated Industry it defines. This can be done using the Industry_Adder() function (seen function above in Annual Business Survey).
10. Cast the following columns to FloatType. Converting to float type code is listed earlier in this ETL, just replace appropriate dataframe name and column name: InitialApprovalAmount, CurrentApprovalAmount, JobsReported, UTILITIES_PROCEED, PAYROLL_PROCEED, REFINANCE_EIDL_PROCEED, HEALTH_CARE_PROCEED, DEBT_INTEREST_PROCEED, ForgivenessAmount
11. See Exporting All Cleaned CSVs below; last transformation step.

Exporting All Cleaned CSV's:

1. Connect to the appropriate storage account and storage container that you'd like to store the cleaned CSV's in.
2. Create a new mount point to this container.
3. Export Cleaned_ABS_Company_Summary and Cleaned_Unemployment by overwriting the csvs.

```
df_company_summary_State.write.mode("Overwrite").csv("/mnt/group-2-capstone/csv
Store/Cleaned_ABS_Company_Summary.csv", header = True)
unemployment_df.write.mode("Overwrite").csv("/mnt/group-2-capstone/csvStore/Clea
ned_Unemployment.csv", header = True)
```

4. For PPP, make a copy of the dataframe, split it up into smaller pieces, save each one to a csv using the following while loop.

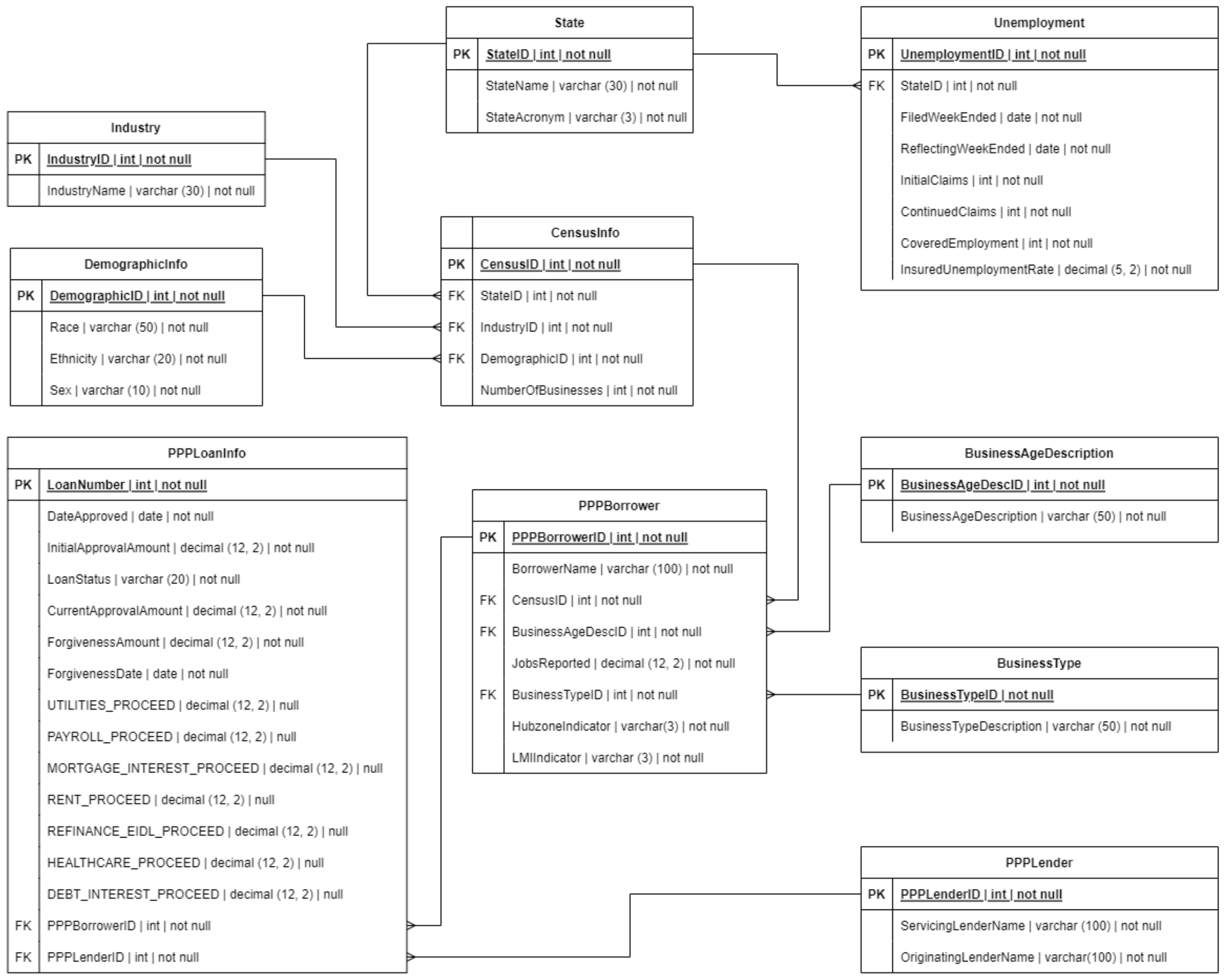
```
dfs_PPP_list = []
n_splits = 2
```

```
# Create a copy of original dataframe
copy_df = PPP.select("*")
length = copy_df.count()
each_len = length // n_splits
print(length)

i = 0
while i < n_splits:
   i+=1
   # Get the top `each_len` number of rows
   temp_df = copy_df.limit(each_len)
   print(f"{i}.{1}")
   sleep(60)
   # Truncate the `copy_df` to remove
   # the contents fetched for `temp_df`
   copy_df = copy_df.subtract(temp_df)
   print(f"{i}.{2}")
   sleep(80)

temp_df.write.mode("Overwrite").csv(f"/mnt/group-2-capstone/csvStore/Cleaned_PPP_
{i}.csv", header = True)
   print(i)
   sleep(80)
i += 1
left_overs = copy_df.select("*")
sleep(80)
left_overs.write.mode("Overwrite").csv(f"/mnt/group-2-capstone/csvStore/Cleaned_PP
P_{i}.csv", header = True)
```

5. Now all cleaned csvs are stored in the storage container and are ready to be loaded into the appropriate tables in a SQL database.

Load:

The next step is to load all data into a SQL server database. It is important to understand how the tables will interact. Below is our ERD Diagram.


State
PK StateID | int | not null
StateName | varchar (30) | not null
StateAcronym | varchar (3) | not null
Unemployment
PK UnemploymentID | int | not null
FK StateID | int | not null
FiledWeekEnded | date | not null
ReflectingWeekEnded | date | not null
InitialClaims | int | not null
ContinuedClaims | int | not null
CoveredEmployment | int | not null
InsuredUnemploymentRate | decimal (5, 2) | not null
Industry
PK IndustryID | int | not null
IndustryName | varchar (30) | not null
DemographicInfo
PK DemographicID | int | not null
Race | varchar (50) | not null
Ethnicity | varchar (20) | not null
Sex | varchar (10) | not null
CensusInfo
PK CensusID | int | not null
FK StateID | int | not null
FK IndustryID | int | not null
FK DemographicID | int | not null
NumberOfBusinesses | int | not null
PPPLoanInfo
PK LoanNumber | int | not null
DateApproved | date | not null
InitialApprovalAmount | decimal (12, 2) | not null
LoanStatus | varchar (20) | not null
CurrentApprovalAmount | decimal (12, 2) | not null
ForgivenessAmount | decimal (12, 2) | not null
ForgivenessDate | date | not null
UTILITIES_PROCEED | decimal (12, 2) | null
PAYROLL_PROCEED | decimal (12, 2) | null
MORTGAGE_INTEREST_PROCEED | decimal (12, 2) | null
RENT_PROCEED | decimal (12, 2) | null
REFINANCE_EIDL_PROCEED | decimal (12, 2) | null
HEALTHCARE_PROCEED | decimal (12, 2) | null
DEBT_INTEREST_PROCEED | decimal (12, 2) | null
FK PPPBorrowerID | int | not null
FK PPPLenderID | int | not null
PPPBorrower
PK PPPBorrowerID | int | not null
BorrowerName | varchar (100) | not null
FK CensusID | int | not null
FK BusinessAgeDescID | int | not null
JobsReported | decimal (12, 2) | not null
FK BusinessTypeID | int | not null
HubzoneIndicator | varchar(3) | not null
LMIIndicator | varchar (3) | not null
BusinessAgeDescription
PK BusinessAgeDescID | int | not null
BusinessAgeDescription | varchar (50) | not null
BusinessType
PK BusinessTypeID | not null
BusinessTypeDescription | varchar (50) | not null
PPPLender
PK PPPLenderID | int | not null
ServicingLenderName | varchar (100) | not null
OriginatingLenderName | varchar(100) | not null

1. Create the SQL database
    a. Create SQL database in appropriate Azure resource group
    b. Create SQL database and add users and roles.
    c. Build the SQL tables with a DDL script based on the ERD. Exact DDL script is found here.
2. Load all tables into SQL using databricks. Since there are tables with foreign keys, the order of which tables are loaded into SQL is important. You must ensure that all foreign keys needed for a table are already in SQL before adding in a table.
    a. Below is our data flow diagram which shows the order in which tables need to be loaded in/databricks need to be run.

3. Loading State, Industry, and Demographic tables Databrick
    a. Since these tables do not have any foreign keys and their primary keys are foreign keys in other tables, these tables need to be created first.
    b. The loading databrick in its entirety can be found here.
    c. Imports:

```
import uuid
from confluent_kafka.admin import AdminClient, NewTopic
from pyspark.sql.functions import col
from pyspark.sql.functions import lit
from pyspark.sql.types import FloatType
from pyspark.sql.types import StringType
import pandas as pd
```

d. Connect to appropriate Azure storage account and storage container that holds all cleaned CSVs from the transformation steps.
e. Create a mount point to this container.
f. Connect to appropriate SQL database, username, password, server, and port that were previously made/given by instructors.
g. Read in the Cleaned ABS csv data into a spark dataframe

```
ABS_df =
spark.read.format("csv").load("/mnt/adambrew/ABS/Cleaned_ABS_Company_S
ummary.csv", header=True, inferSchema='True')
```

   i. Now you have the entire ABS data which now needs to be broken down into the individual tables
h. Define helper function SaveToTable which will push to SQL. Function will make it easier since this process will happen multiple times.

```
def saveToTable(df, table, change='append'):
  df.write.format('jdbc').option("url",
f"jdbc:sqlserver://{server}:1433;databaseName={database};") \
        .mode(change) \
        .option("dbtable", table) \
        .option("user", user) \
        .option("password", password) \
        .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
        .save()
```

i. State table
   i. Create a dictionary named us_state_to_abbrev for each state's name as the key and each state's abbreviation as the value.
   ii. Assign the State_df dataframe to take just distinct values from the 'State' column from ABS_df.
   iii. Create a second column named 'StateAcronym' with all full state names.
   iv. Add a row of 'Unanswered' to both columns.
   v. Using a lambda function, rewrite the StateAcronym' column to each state's abbreviation.
   vi. Save this as a spark dataframe.
   vii. Full code here:

```
State_df = ABS_df[["State"]].distinct().sort(col("State"))
State_df = State_df.withColumn("StateAcronym", col("State"))
State_df_pd = State_df.toPandas()
State_df_pd["StateAcronym"] =
State_df_pd["StateAcronym"].apply(lambda x : us_state_to_abbrev[x])
State_df_pd = State_df_pd[State_df_pd["StateAcronym"] != "DC"]
State_df_pd["StateName"] = State_df_pd["State"]
State_df_pd = State_df_pd.drop("State", axis = 1)
State_df = spark.createDataFrame(State_df_pd)
State_df = State_df.select("StateName", "StateAcronym")
new_row = spark.createDataFrame([("Unanswered", "Unanswered")],
State_df.columns)
State_df = State_df.union(new_row)
```

viii. Write the State_df dataframe to SQL using the SaveToTable helper function.

j. Industry Table
   i. Assign the Industry_df dataframe to take just distinct values from the 'Industry' column from ABS_df.
   ii. Create a duplicate column of 'Industry' named 'IndustryName'. Drop Industry column.
   iii. Append a row of 'Unanswered' to the dataframe.
   iv. Save this as a spark dataframe. Write the Industry_df dataframe to SQL using the SaveToTable helper function.

k. Demographic table
   i. Assign the Demographic_df dataframe to take the distinct values from 'Race', 'Ethnicity', and 'Sex' columns from ABS_df.
   ii. Using a for loop, add an instance/new row to each distinct row where every outcome has 'Unanswered' in one of the columns. Code here:

```
df_list_of_dict = [row.asDict() for row in Demographic_df.collect()]

cols = list(df_list_of_dict[0].keys())
num_cols = len(cols)
num_rows = len(df_list_of_dict)

for i in range(num_cols):
  copy_l = [row.asDict() for row in Demographic_df.collect()]
  for j in range(num_rows):
    copy_l[j][cols[i]] = 'Unanswered'
  for k in copy_l:
    df_list_of_dict.append(k)

df_list_of_dict = [dict(t) for t in {tuple(d.items()) for d in df_list_of_dict}]
```

iii. Add an instance/new row to each distinct row where every outcome has 'Unanswered' in one of the columns for each demographic. EX: {'Race': 'Unanswered', 'Ethnicity': 'Unanswered', 'Sex': 'Female'}

iv. Save this as a spark dataframe. Write the Demographic_df dataframe to SQL using the SaveToTable helper function.

4. Load all tables that pull data from the PPP csvs.
    a. The loading databrick in its entirety can be found here.
    b. Imports:

```
import uuid
from confluent_kafka.admin import AdminClient, NewTopic
from pyspark.sql.functions import concat, col, lit,isnan,when,count, udf
from pyspark.sql.types import FloatType
from pyspark.sql.types import StringType
import pandas as pd
from time import sleep
```

    c. Connect to appropriate Azure storage account and storage container that holds all cleaned CSVs from the transformation steps.
    d. Create a mount point to this container.
    e. Connect to appropriate SQL database, username, password, server, and port that were previously made/given by instructors.
    f. Define same saveToTable function as illustrated earlier.
    g. Read in the Cleaned PPP csv data into a sparks dataframe using a for loop to unionize them together.

```
PPP_df =
spark.read.format("csv").load("/mnt/adambrew/PPP/Cleaned_PPP_1.csv",
header=True, inferSchema='True')

for i in range(2,4):
   PPP__df =
spark.read.format("csv").load(f"/mnt/adambrew/PPP/Cleaned_PPP_{i}.csv",
header=True, inferSchema='True')
   PPP_df = PPP_df.union(PPP__df)

print(f"Shape: Columns: {len(PPP_df.columns)}, Rows: {PPP_df.count()}, Rows
distinct: {PPP_df.distinct().count()}")
```

    h. Remove pyspark added column '_c0'
    i. Create all main dataframes used for making the tables.

```
PPPLender_df = PPP_df[["ServicingLenderName",
"OriginatingLender"]].distinct()
```

```
BusinessType_df = PPP_df[["BusinessType"]].distinct()

BusinessAgeDescription_df = PPP_df[["BusinessAgeDescription"]].distinct()

PPPLoanInfo_df = PPP_df[["DateApproved", "InitialApprovalAmount",
"LoanStatus", "CurrentApprovalAmount", "ForgivenessAmount",
"ForgivenessDate", "UTILITIES_PROCEED", "PAYROLL_PROCEED",
"MORTGAGE_INTEREST_PROCEED", "RENT_PROCEED",
"REFINANCE_EIDL_PROCEED", "HEALTH_CARE_PROCEED",
"DEBT_INTEREST_PROCEED", "BorrowerName",
“ServicingLenderName”]].distinct()

PPPBorrower_df = PPP_df[["BorrowerName", "JobsReported",
"HubzoneIndicator", "LMIIndicator", "BusinessType",
"BusinessAgeDescription"]].distinct()
```

j. Create PPPLender table and write it to SQL
    i. In PPPLender_df, rename column “OriginatingLender” to “OriginatingLenderName”
    ii. Write the PPPLender_df dataframe to SQL using the SaveToTable helper function.
k. Create BusinessType table write it to SQL
    i. Ensure there are no nulls in BusinesssType_df, if there are, get rid of them
    ii. Delete all rows that contain ‘0.0’ or the word ‘null’. Conditional row removal code is listed earlier in this ETL, just replace appropriate dataframe name and column name.
    iii. Rename column “BusinessType” to “BusinessTypeDescription”
    iv. Write BusinessType_df dataframe to SQL using the SaveToTable helper function.
l. Create BusinessAgeDescription table and write it to SQL
    i. Write BusinessAgeDescription_df dataframe to SQL as-is using the SaveToTable helper function.
m. Create readInTable helper function:

```
def readInTable(table_name):
   df = spark.read.format("jdbc") \
      .option("url",
f"jdbc:sqlserver://{server}:{port};databaseName={database};") \
      .option("dbtable", table_name).option("user", user).option("password",
password) \
      .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
      .load()
   return df
```

n. Create Foreign Key Matcher function:

```
def foreign_key_matcher(main_table , main_column, foreign_table,
foreign_column, foreign_ID):
  # tables should be in the form of dataframes from SQL
  columns_wanted = list(main_table.columns)
  columns_wanted.remove(main_column)
  columns_wanted.append(foreign_ID)
  main_table = main_table.join(foreign_table, main_table[main_column] ==
foreign_table[foreign_column], "inner").distinct()
  main_table = main_table.select(columns_wanted)
  return main_table
```

o. Create CensusInfo table and write it to SQL
   i. Load in the Cleaned_ABS_Company_Summary csv into a spark dataframe called CensusInfo_df
   ii. Convert to pandas.
   iii. Remove ‘District of Columbia’ and from append ‘Unanswered’ State column.
   iv. Add an instance/new row to each distinct row where every outcome has ‘Unanswered’ in one of the columns for each demographic. EX: 'Race': 'Unanswered', 'Ethnicity': 'Unanswered', 'Sex': 'Female', "NumberOfBusinesses": 0}
   v. Using a for loop to append, appended the “Unanswered” rows with every combination of state and industry. EX:

```
for state_id in states_list:
  for Industry in Industry_list:
    df_list_of_dict.append({'State': f"{state_id}", 'Industry':
f"{Industry}", 'Race': 'Unanswered', 'Ethnicity': 'Unanswered', 'Sex':
'Female', "NumberOfBusinesses": 0})
```

   vi. Again for race. EX:

```
for state_id in states_list:
  for Industry in Industry_list:
    for race in Race_unanswered_list:
      df_list_of_dict.append({'State': f"{state_id}", 'Industry':
f"{Industry}", 'Race': f"{race}", 'Ethnicity': 'Unanswered', 'Sex': 'Female',
"NumberOfBusinesses": 0})
```

   vii. Turn into spark dataframe.
   viii. Use readInTable function to read in dbo.DemographicInfo, dbo.Industry, dbo.State
   ix. Use foreign_key_matcher function to match ID’s for State with State df, and Industry with Industry df
   x. Join Census df and Demograhic df on distinct Race, Sex, and Ethnicity.
   xi. Select the following columns for Census df: StateID, IndustryID, DemographicID, NumberofBusinesses.

```
CensusInfo_df_IDs = CensusInfo_df.select("*")

State_SQL_df = readInTable("dbo.State")
print(CensusInfo_df_IDs.count())

CensusInfo_df_IDs = foreign_key_matcher(CensusInfo_df_IDs , "State",
State_SQL_df, "StateName", "StateID")
print(CensusInfo_df_IDs.count())

CensusInfo_df_IDs = foreign_key_matcher(CensusInfo_df_IDs ,
"Industry", Industry_SQL_df, "IndustryName", "IndustryID")
print(CensusInfo_df_IDs.count())

CensusInfo_df_IDs = CensusInfo_df_IDs.join(DemographicInfo_SQL_df,[
   (CensusInfo_df_IDs["Race"] == DemographicInfo_SQL_df["Race"]) &\
   (CensusInfo_df_IDs["Sex"] == DemographicInfo_SQL_df["Sex"]) &\
   (CensusInfo_df_IDs["Ethnicity"] ==
DemographicInfo_SQL_df["Ethnicity"])],"inner").distinct()
print(CensusInfo_df_IDs.count())

CensusInfo_df_IDs = CensusInfo_df_IDs.select("StateID", "IndustryID",
"DemographicID", "NumberOfBusinesses").distinct()
print(CensusInfo_df_IDs.count())
display(CensusInfo_df_IDs)
```

xii. Write dataframe to SQL with the table name CensusInfo using the SaveToTable helper function.

p. Create PPP_Borrower table and write it to SQL
   i. Use readInTable function to read in dbo.DemographicInfo, dbo.Industry, dbo.State, and dbo.CensusInfo
   ii. Let PPP_df_ be a copy of PPP_df from earlier..
   iii. Update PPP_df_ dataframe to exclude “Eskimo & Aluet”, “Multi Group” and “Puerto Rican” from the Race column.
   iv. Edit column entries as such:
      1. Ethnicity column: “Unknown/NotStated’ ⟶ ‘Unanswered’
      2. Ethnicity column: ‘Hispanic or Latino’ ⟶ ‘Hispanic’
      3. Ethnicity column: ‘Not Hispanic or Latino’ ⟶ ‘Non-Hispanic’
      4. Gender column: ‘Male Owned’ ⟶ ‘Male’
      5. Gender column: ‘Female Owned’ ⟶ ‘Female’
      6. Append ‘Unanswered’ to Ethnicity and Gender column
   v. Inner join PPP_df_ and the Industry data frame on the ‘Industry’ and ‘IndustryName’ columns, respectively
   vi. Inner join PPP_df_ and State data frame on the ‘BorrowerState’ and ‘StateAcronym’ column, respectively.

vii. Inner join PPP_df_ and Demographic data frame on the ‘Race’, ‘Gender’/’Sex’, and ‘Ethnicity’

viii. Inner join PPP_df_ and Census data frame on ‘StateID’, ‘IndustryID’, and ‘DemographicID’

ix. Select the following columns columns and save it toPPP_df_: BorrowerName, Jobs Reported, HubzoneIndicator, LMIIndicator, BusinessType, BusinessAgeDescription, CensusID

x. Read in dbo.BusinessType and dbo.BusinessAgeDescription tables from SQL

xi. Use foreign_key_matcher function to match each of them on their respective IDs

    PPP_df_ = foreign_key_matcher(PPP_df_ , "BusinessType", BusinessType_SQL_df, "BusinessTypeDescription", "BusinessTypeID")

    PPP_df_ = foreign_key_matcher(PPP_df_ , "BusinessAgeDescription", BusinessAgeDescription_SQL_df, "BusinessAgeDescription", "BusinessAgeDescID")

xii. Save table into SQL to table PPPBorrower using the saveToTable helper function.

q. Create PPPLoanInfo table and write it to SQL

i. Set PPP_LoanInfo_df to be a copy of PPP_df from earlier.

ii. Update PPP_LoanInfo_df to exclude “Eskimo & Aluet”, “Multi Group” and “Puerto Rican” from the Race column.

iii. Update PPP_LoanInfo_df columns as such:

1. Ethnicity column: “Unknown/NotStated’ → ‘Unanswered’
2. Ethnicity column: ‘Hispanic or Latino’ → ‘Hispanic’
3. Ethnicity column: ‘Not Hispanic or Latino’ → ‘Non-Hispanic’
4. Gender column: ‘Male Owned’ → ‘Male’
5. Gender column: ‘Female Owned’ → ‘Female’
6. Append ‘Unanswered’ to Ethnicity and Gender column

iv. Use readInTable function to read in DemographicInfo, dbo.Industry, dbo.State

v. Join PPP_LoanInfo_df and Industry df on ‘Industry’ and ‘IndustryName’, respectively

vi. Join PPP_LoanInfo_df and State df on ‘BorrowerState’ and ‘StateAcronym’, respectively.

vii. Rename ‘Gender’ column in PPP_LoanInfo_df to ‘Sex’

viii. Join PPP_LoanInfo_df and Demographic df with distinct entries from ‘Race’,’Sex’, and ‘Ethnicity’.

ix. Select following columns and save as PPP_LoanInfo_df: LoanNumber, DateApproved, BorrowerName, LoanStatus, InitialApprovalAmount, CurrentApprovalAmount, HubzoneIndicator, LMIIndicator, JobsReported, UTILITIES_PROCEED, PAYROLL_PROCEED, MORTAGE_INTEREST_PROCEED, RENT_PROCEED, REFINANCE_EIDL_PROCEED, HEALTH_CARE_PROCEED, DEBT_INTEREST_PROCEED, BusinessType, BusinessAgeDescription, ForgivenessAmount, ForgivenessDate, IndustryID, StateID, DemographicID, ServicingLenderName., OriginatingLender

x. Join PPP_LoanInfo_df and Census df on StateID, IndustryID, and DemographicID
xi. Drop duplicates from 'LoanNumber' column
xii. Select following columns and save as PPP_LoanInfo_df: LoanNumber, DateApproved, BorrowerName, LoanStatus, InitialApprovalAmount, CurrentApprovalAmount, HubzoneIndicator, LMIIndicator, JobsReported, UTILITIES_PROCEED, PAYROLL_PROCEED, MORTAGE_INTEREST_PROCEED, RENT_PROCEED, REFINANCE_EIDL_PROCEED, HEALTH_CARE_PROCEED, DEBT_INTEREST_PROCEED, BusinessType, BusinessAgeDescription, ForgivenessAmount, ForgivenessDate, IndustryID, StateID, DemographicID, ServicingLenderName, OriginatingLender
xiii. Use readInTable function to read in dbo.BusinessAgeDescription, dbo.BusinessType
xiv. Join LoanInfo and BusinessType on BusinessTypeID.
xv. Join LoanInfo and BusinessAgeDescription on BusinessAgeDescID
xvi. Read in dbo.PPPBorrower
xvii. Join PPP_LoanInfo_df and Borrower df on BorrowerName, JobsReported, HubzoneIndicator, LMIIndicator, CensusID, BusinessTypeID, BusinessAgeDescID.
xviii. Select following columns and save as PPP_LoanInfo_df: LoanNumber, DateApproved, BorrowerName, LoanStatus, InitialApprovalAmount, CurrentApprovalAmount, HubzoneIndicator, LMIIndicator, JobsReported, UTILITIES_PROCEED, PAYROLL_PROCEED, MORTAGE_INTEREST_PROCEED, RENT_PROCEED, REFINANCE_EIDL_PROCEED, HEALTH_CARE_PROCEED, DEBT_INTEREST_PROCEED, BusinessType, BusinessAgeDescription, ForgivenessAmount, ForgivenessDate, IndustryID, StateID, DemographicID, ServicingLenderName, OriginatingLender
xix. Read in dbo.PPPLender table.
xx. Join LoanInfo and PPPLender on ServicingLenderName and OriginatingLenderName
xxi. Select following columns and save as PPP_LoanInfo_df: LoanNumber, DateApproved, InitialApprovalAmount, LoanStatus, CurrentApprovalAmount, ForgivenessAmount, ForgivenessDate, UTILITIES_PROCEED, PAYROLL_PROCEED, MORTGAGE_INTEREST_PROCEED, RENT_PROCEED, REFINANCE_EIDL_PROCEED, HEALTH_CARE_PROCEED, DEBT_INTEREST_PROCEED, PPPBorrowerID, PPPLenderID
xxii. Cast the following columns as StringType: DateApproved, ForgivenessDtae
xxiii. Cast the following columns as FloatType: InitialApprovalAmount, CurrentApprovalAmount, ForgivenessAmount, UTILITIES_PROCEED, PAYROLL_PROCEED
xxiv. Rename HEALTH_CARE_PROCEED to HEALTHCARE_PROCEED
xxv. Drop 'LoanNumber'
xxvi. Write df into SQL to PPPLoanInto table using the saveToTable helper function.

5. Use Kafka producer to send Unemployment messages off.
   a. The producer and consumer databricks can be found at the following sites:

        i. Producer:
           https://github.com/emilyatk13/dev-10-capstone/blob/bcefc01a209159cfc3998fb109155de9341ec93c/code/Unemployment_Producer.ipynb
        ii. Consumer:
        iii. https://github.com/emilyatk13/dev-10-capstone/blob/bcefc01a209159cfc3998fb109155de9341ec93c/code/Unemployment_Consumer.ipynb

   b. Connect to appropriate Azure storage account and storage container that holds all cleaned CSVs from the transformation steps.
   c. Create a mount point to this container.
   d. Read in the Cleaned_Unemployment.csv into a spark dataframe df.
   e. Briefly convert dataframe to pandas dataframe called df2 to fun following lambda function on FiledWeekEdned and ReflectingWeekEnded columns to convert entries to string type. Kafka/JSON does not like datetime

```
df2[‘column_name’]=df2[‘column_name’].apply(lambda x:
str(x.strftime('%m/%d/%Y'))
```

   f. Convert df2 back to spark dataframe called Unemployment
   g. Set up all appropriate Kafka variables. All should be provided for you but the topic name. Name the topic name ‘group2trial’.
   h. Set up all error message functions that have been provided.
   i. Set up Kafka Class Setups for Producer and AdminClient that have been provided.
   j. Convert Unemployment df to a list of JSON dictionaries. Following code:

```
templist = []
for i in range(Unemployment.count()):
  Unemployment2 = Unemployment.toJSON().map(lambda j:
json.loads(j)).collect()[i]
  templist.append(Unemployment2)
```

   k. Send the messages to the Kafka producer. Following code:

```
i=0
while(i<len(templist)-1):
  i=i+1
  p.produce(confluentTopicName,json.dumps(templist[i]))
  p.flush()
  print(f'Current step: {i}')
  print(templist[i])

  sleep(5)
```

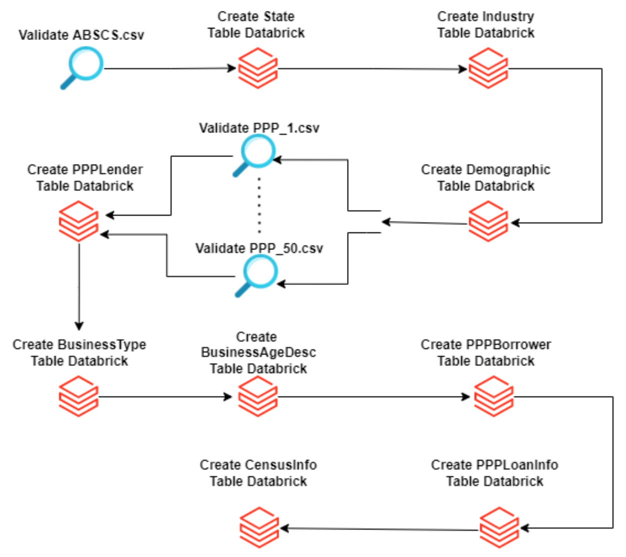
6. Use Kafka consumer to read in messages from unemployment and write table to SQL.
   a. Set up all error message functions that have been provided.
   b. Set up all appropriate Kafka variables. All should be provided for you but the topic name. Make sure to use the same topic name from the producer, ‘group2trial’.

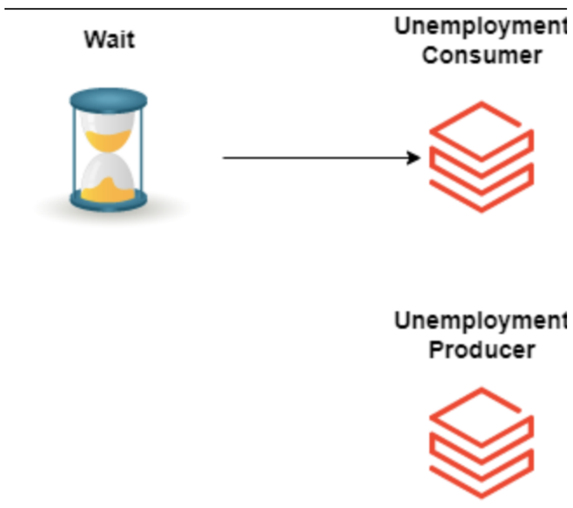
c. Set up Kafka Class Setup for Consumer that has been provided.
d. Must subscribe to topic name variable → c.subscribe([confluentTopicName])
e. Consume all messages and print out the KafkaListDictionaries. Following code:

```
aString = ""
kafkaListDictionaries = []

for i in range(5512):
    try:
        msg = c.poll(timeout=15)
        print(msg)
        if msg is None:
            break
        elif msg.error():
            print("Consumer error: {}".format(msg.error()))
            break
        else:
            aString=json.loads('{}'.format(msg.value().decode('utf-8')))
            kafkaListDictionaries.append(aString)
            c.commit(asynchronous=False)
    except Exception as e:
        print(e)
print(json.dumps(kafkaListDictionaries, indent=4, sort_keys=True))
```

f. Save kafkaListDictionaries into a spark dataframe called Unemployment_df
g. Select the following columns for Unemployment_df → State, FiledWeekEnded, ReflectingWeekEnded, InitialClaims, ContinuedClaims, CoveredEmployment, InsuredUnemploymentRate
h. Connect to appropriate Azure storage account and storage container.
i. Create a mount point to this container.
j. Connect to appropriate SQL database, username, password, server, and port that were previously made/given by instructors.
k. Use the readInTable function to read in dbo.State.
l. Use foreign_key_matcher function to match on their respective State column
m. Write the Unemployment dataframe to SQL using the SaveToTable helper function.

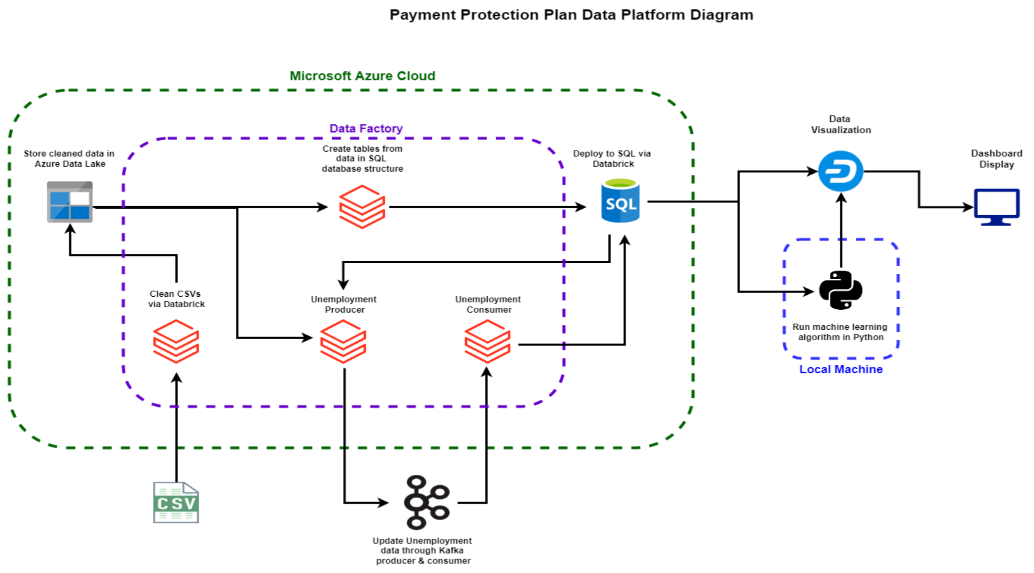
7. And finally, all tables are loaded into SQL with the appropriate foreign keys!
8. Set up the data factory to populate the database through a pipeline with the databricks.
    a. The data factory for the static data should look like the image on the left, and the Streaming kafka producer/consumer datafactory on the right


Validate ABSCS.csv
Create State Table Databrick
Create Industry Table Databrick
Validate PPP_1.csv
Create PPPLender Table Databrick
Create Demographic Table Databrick
Validate PPP_50.csv
Create BusinessType Table Databrick
Create BusinessAgeDesc Table Databrick
Create PPPBorrower Table Databrick
Create CensusInfo Table Databrick
Create PPPLoanInfo Table Databrick



Conclusion:

After completion of this process, this is the structure of the network diagram.


Payment Protection Plan Data Platform Diagram
Microsoft Azure Cloud
Data Factory
Store cleaned data in Azure Data Lake
Create tables from data in SQL database structure
Deploy to SQL via Databrick
SQL
Clean CSVs via Databrick
Unemployment Producer
Unemployment Consumer
CSV
Update Unemployment data through Kafka producer & consumer
Data Visualization
Dashboard Display
Run machine learning algorithm in Python
Local Machine