ETL Report Guide

Christopher Ng, Jake Thompson, Mikhaela Anderson, Nick Kartschoke, Ryan Lippe

5/16/2022

Introduction

We are trying to predict unemployment rates using a combination of local government finances from the US census, small business administration disaster loans, and S&P 500 data. To do this, we are using a machine learning algorithm. The data needed to be transformed into a uniform format and have nulls removed to be useful for machine learning. After that, it was merged into a single table to be fed into a kafka server and ultimately be stored in a relational SQL database which we'll refer to for machine learning and other analysis.

Data Sources

Ravaliya, Jay. *US Unemployment Rate by County*, 1990-2016.
https://www.kaggle.com/datasets/jayrav13/unemployment-by-county-us

U.S. Small Business Administration. Disaster Loan Data. 2008-2015.
https://data.sba.gov/dataset/disaster-loan-data

United States Census Bureau. *State & Local Government Finance Historical Datasets and Tables.* https://www.census.gov/programs-surveys/gov-finances/data/datasets.html.

Willden, Chase. *Sp500.* Accessed 2022.
https://data.world/chasewillden/stock-market-from-a-high-level

World Population Review. *List of State Abbreviations.* Accessed 2022.
https://worldpopulationreview.com/states/state-abbreviations

Census Data Extraction

The census data used was for state and local government finances, and we were interested in the years 2008-2015. We saw that the data was in 2 distinct formats, split before and after 2013. Pre-2013 data was in multiple csv files, with a set of 3 for each year. Post-2013 data was in one fixed-width file for each year, with an additional fixed-width file containing county and state code definitions matched with names. Pre-2013 data had each different feature as a separate column, while post-2013 data had these columns encoded as “ItemCode”, with the meaning of the dollars column being different based on the ItemCode. This needed to be broken out into separate columns in order to merge the datasets.

Census Data Transformation

1. Import pandas
2. Read in the fixed-width data for 2013 as str, using column widths [2,1,3,3,5,3,12,4,1] as seen in the state and local government finances technical documentation pdf
3. Label columns ['StateCode', 'Type', 'County', 'UnitIdentifier', 'AnotherGov', 'ItemCode', 'ThousandsOfDollars', 'Year, 'ImputationType'] based on the technical documentation pdf
4. Repeat 3 for 2014 and 2015, and concat into one dataframe called jdf
5. Read in GID.txt, the fixed width file with state and county codes and names, as str using column widths [14,64,35,2,3,5,9,2,7,2,2,2,4,2] as jdf2, and name columns ['IDCode', 'IDName', 'County Name', 'StateCode', 'CountyCode', 'PlaceCode', 'Population', 'Population Year', 'Enrollment', 'Enrollment Year', 'Function Code for Special Districts', 'School Level Code', 'Fiscal Year Ending', 'Survey Year']
6. Create a csv containing FIPS state codes, state names, and state abbreviations.
7. Read in the FIPS csv as a dataframe called fips
8. Read in the 3 csvs for 2008 as str, and concat into one dataframe
9. Iterate over years 2008-2012, and concat all of these into a dataframe jdf8
10. For jdf, jdf2, and jdf8, create column 'State/County' equal to 'StateCode' + 'County'
11. For jdf8 rename columns 'Year4' to 'Year' and 'ID' to 'IDCode'
12. Iterate over 'IDCode' in jdf2 and jdf8, set each cell equal to the 3rd character in 'IDCode'
13. From jdf, drop all rows where 'County' == '000'
14. From jdf2 and jdf8, drop all rows where 'IDCode' != '1'
15. From jdf2, drop all columns but 'County Name', 'IDName', and 'State/County'
16. Find itemcodes for columns of interest in the technical documentation. {'T01': 'Property Tax', 'Z00' : 'Total Salaries & Wages', 'U20' : 'Interest Revenue', 'U30' : 'Fines and Forfeits'}
17. Make df called sub1 equal to jdf where jdf['ItemCode'] == 'T01', with columns ['StateCode', 'County', 'ThousandsOfDollars', 'Year', 'State/County']
18. Rename 'ThousandsOfDollars' to 'Property Tax'
19. Cast 'Property Tax' to int
20. Group 'Property Tax' by 'State/County' and 'Year'
21. Aggregate as sum
22. Reset the index, assign back to sub1
23. Repeat 17-22 with a new df subx for each column of interest
24. Set df master equal to sub1.merge(sub2, on = ['Year', 'State/County'])
25. Iterate through all subx dfs.
26. For jdf8 drop all columns but ['Year', 'Property Tax', 'Total Salaries & Wages', 'Interest Revenue', 'Fines and Forfeits', 'State/County']
27. For jdf8, dropna
28. Concat master with jdf8, assign to master
29. Reset the index of master
30. Create empty column in master called 'StateCode'
31. Iterate through master, set 'StateCode' to 'State/County'[0:2]
32. Merge master with jdf2 on 'State/County', assign back to master

33. Merge master with fips on ‘StateCode’, assign back to master
34. Drop all columns from master except ['Year', 'Property Tax', 'Total Salaries & Wages', 'Interest Revenue', 'Fines and Forfeits', 'County Name', 'State Name']
35. In master, rename ‘County Name’ to ‘County’ and ‘State Name’ to ‘State
36. Cast ‘State’ and ‘County’ to uppercase str
37. Write master to csv

Unemployment Data Extraction

For the unemployment dataset, there were two files found in our source from kaggle, and we wound it was necessary to work with the JSON file, as the CSV file was missing data for three states. The primary ETL for this data source was done in a Jupyter Notebook using Pandas and Numpy. The following steps were taken in order to transform the data into how it would be merged with our other sources and eventually our SQL schema.

Unemployment Data Transformation

1. Import pandas as pd, json, and numpy as np, as we will use these libraries.
2. We then open our JSON file which we placed in the same directory with the name Unemployement.json. The code was:
   ```
   with open(‘Unemployement.json’) as f:
   df = json.load(f)
   ```
3. The next step is to normalize it, using df = pd.json_normalize(df)
4. Next, it is necessary to reformat the data for dropping unnecessary values. To do this, we first make a new dataframe:
   ```
   df1 = pd.DataFrame(data=None, index=None, columns=['Year','Timeframe',
   'State','Unemployement','County'], dtype=None, copy=False)
   ```
5. Next we append to it, by iterating through each column, and splitting on each period in order to add each new row. This uses the following code:
   ```
   i = 0
   while i < len(df.columns):
       data = df.columns[i].split('.')
       value = df[df.columns[i]][0]
       newRow = pd.Series([data[0],data[1],data[2],value,data[4]], index = df1.columns)
       df1 = df1.append(newRow, ignore_index=True)
       i = i + 1
   ```
6. Once reformatted, we begin by dropping all date where the timeframe is not annual, as we only care about the datapoints for each year. This is done with:
   ```
   df1 = df1[df1.Timeframe == 'Annual']
   ```
7. Next we filter to get the year range we desire, from 2008-2015. This is done with:
   ```
   df1 = df1[df1.Year <= '2015']
   df1 = df1[df1.Year >= '2008']
   ```

8. Next we reset the index:
   df1.reset_index(inplace=True)
9. Next we drop columns that will not be used:
   df1.drop(['Timeframe', 'index'], axis=1, inplace=True)
10. To prepare for the merge with the other datasets, it's necessary to drop the word 'County' from each row of the county column. This is done in two lines with:
    df1.County = df1.County.str.rstrip('County')
    df1.County = df1.County.str.rstrip(' ')
11. Next we need to capitalize each entry within the state and county columns:
    df1['State'] = df1['State'].str.upper()
    df1['County'] = df1['County'].str.upper()
12. The final step is to write the cleaned dataframe to a CSV
    df1.to_csv('CleanedUnemployement.csv')

Disaster Loan Extraction

After accessing the Disaster Loan data on the U.S. Small Business Administration website, we found that the data was split up by year in separate csv files. The first step in the process was to load each dataset into one Jupyter Notebook. Once the data was loaded we had to keep our ERD chart in mind to specify which columns we needed and in what format we needed the data based off of the merges we were going to perform on the multiple datasets we had in the end. When loading each year we performed the following transformation process in Pandas for each dataset represented by the years 2008-2015:

Disaster Loan Transformation

1. Read in csv
2. Drop "_id" & "Fema Disaster Number" column (we dropped the Fema Disaster number because it had over 900 null values and we did not want to lose the data in those rows and did not need the Fema Disaster Number in our report).
3. Create a "Year" column and for each row, add the corresponding year of the dataset

Once all of the years were read in and a dataframe was created for each csv file, we executed the following steps to merge the years into one dataframe and cleaned the data.

1. Create one dataframe for all years and append it with each separate year dataframe
2. Check for null values (most of the rows had only 8, so we decided to drop nulls)
3. Drop null values

The other datasets that we needed to merge the disaster loan data with had state names represented by the full name, however this dataset had the states represented only by their abbreviations. To combat this, we found a separate dataset that held state names and their corresponding abbreviations and merged this dataset with the loan dataset and dropped the original abbreviation columns. Here are the steps we took to do this:

1. Read in state & abbreviations table
2. Rename the column with the state code to “state code” in state dataframe & loan dataframe
3. Use inner merge to merge loan dataframe & state dataframe on “state code” column
4. Drop the columns we were not using for our analysis: State Code, Abbrev, Total Approved Loan Amount, Total Verified Loss, Damaged Property City Name
5. Rename the “Damaged Property County/Parish Name” to “County Name and the “State” column to “State Name” to match the column names of the other datasets
6. Use str.upper() on the “State Name” column to match the format of the other datasets
7. Export a csv of the merged & cleaned final dataset

S&P500 Transformation

1. Read in the csv
2. Replace the values in the Date columns to just the last 4 characters
3. Change the date values to an integer data type
4. Replace entries in the date column with “if” statements to Nan if they aren’t in [2008-2015]
5. Remove all rows with Nan
6. Then do a groupby for the Date and aggregate the values
    a. df2.groupby('Date')['Market Value'].agg('median')
7. Change the date column name to “Year_SP” and value column to “Market Values”
8. Change the df to a csv file

Load

The csvs produced in each transformation step were loaded into a blob container so they could be accessed by our data factory. Our kafka producer loads each csv and does an inner join on state, county, and year. This is order insensitive except for the last step. The merge brings in the disaster loan data with a left join, this is because not all counties we analyzed were ever awarded disaster loans. Nulls were replaced with zeroes. This merged table is sent to the kafka server to simulate a datastream, and is read by a kafka consumer. This consumer writes the data to azure data lake, and then the data is loaded into a SQL database.

For our SQL schema, we followed our ERD, which splits the dataset into 3 separate tables. Below are the table names, table constraints, columns, and type for each column. Nulls were excluded from every column except SBA EIDL and SBA Pysical.

1. DisasterData
   Columns: Disaster ID, SBAPhysical, SBAEIDL, SBADisaster
   Types: int, float, float, varchar(50)
   Constraints: Unique SBADisaster.

2. LocationData
   Columns: LocationID, County, StateName
   Types: int, varchar(100), varchar(100)

3. PredictorData
   Columns: PlaceTimeID, DisasterID, LocationID, Year, UnemployementRate, MarketValue, VerifiedLossRealEstate, VerifiedLossContent, AmountApprovedRealEstate, ApprovedAmountEIDL, PropertyTax, TotalSalariesWages, InterestRevenue, FinesForfeits
   Types: int, int, int, float for the rest
   Constraints: foreign key DisasterID, foreign key LocationID.

Now that the schema is set, we can insert our columns. For the DisasterData table, we inserted SBAPhysical, SBAEIDL, and SBADisaster. It is important to select the distinct values from each column from the merged table imported in previous steps.

Next we insert County and StateName into the LocationData table, selecting the unique County and State combinations from the merged table.

Lastly, we populated our PredictorData table by inserting DisasterID, LocationID, and all remaining columns. It is necessary to specify that DisasterID comes from DisasterData, and LocationID comes from LoactionData. In addition, in order for these insert statements to work, the merged table must be joined with DisasterData on SBADisaster, and the LocationData table must be joined with the merged table on County as well as State. The code for these joins was:

```sql
join DisasterData on M.[SBA Disaster Number] = DisasterData.SBADisaster
join LocationData on M.County = LocationData.County and M.State = LocationData.StateName
```

```
(The M is abbreviated for the denormalized merged table)
```

Conclusion

The data was successfully transformed, merged, and loaded into a relational SQL database, and is now ready to be used for machine learning and our final analysis.