Economic Factors

ETL Project

Extract - Transform - Load

Team Members:

Kasey Wilson

Yuri Groza

Mariam Ahmed

Project Description/Outline:

Utilize two or more datasets to complete ETL and provide an outline of the following:

- The type of transformation needed for this data (cleaning, joining, filtering, aggregating, etc).
- The type of final production database to load the data into (relational or non-relational).
- The final tables or collections that will be used in the production database.

Extract, Transform and Load (ETL) Process:

Extract

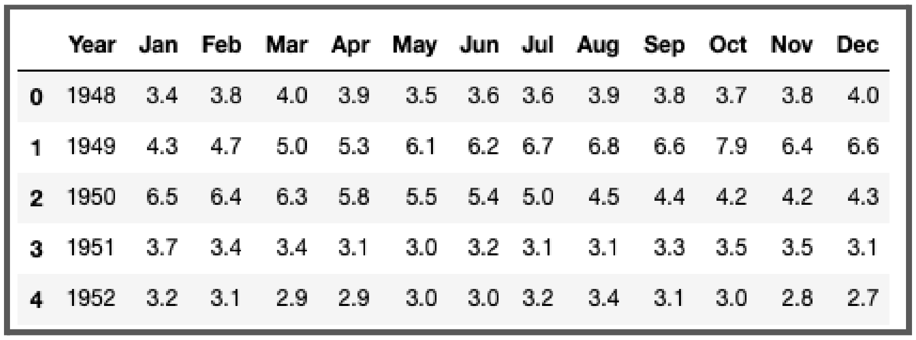
We used the following datasets related to fuel prices and unemployment rate:

- https://www.kaggle.com/mruanova/us-gasoline-and-diesel-retail-prices-19952021
- https://www.kaggle.com/tunguz/us-monthly-unemployment-rate-1948-present

The datasets utilized were both CSV files we found from Kaggle related to fuel prices and unemployment rates in the United States. We selected these files as we were interested in how different economic factors were potentially related. We reviewed different files and ultimately selected these datasets as they both contained month and year data (although in different formats), and found that these both had a large dataset.

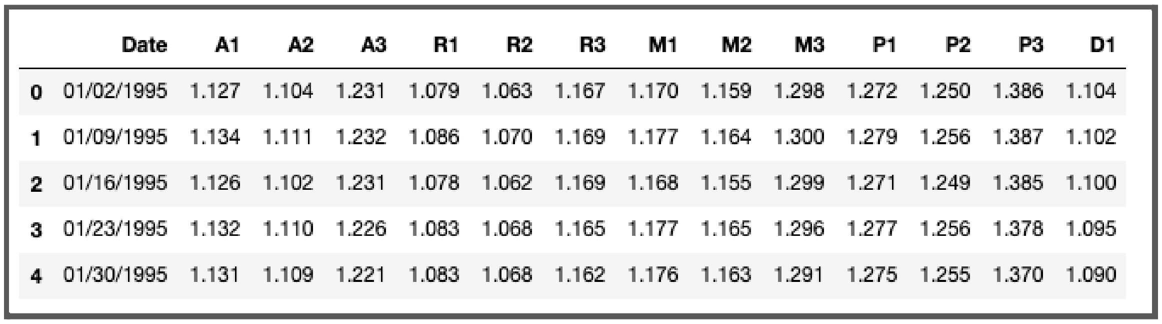
We then imported the CSV files into DataFrames in a jupyter notebook using pandas library to read the CSV files. Our original DataFrames are shown below.

Unemployment DataFrame



	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	1948	3.4	3.8	4.0	3.9	3.5	3.6	3.6	3.9	3.8	3.7	3.8	4.0
1	1949	4.3	4.7	5.0	5.3	6.1	6.2	6.7	6.8	6.6	7.9	6.4	6.6
2	1950	6.5	6.4	6.3	5.8	5.5	5.4	5.0	4.5	4.4	4.2	4.2	4.3
3	1951	3.7	3.4	3.4	3.1	3.0	3.2	3.1	3.1	3.3	3.5	3.5	3.1
4	1952	3.2	3.1	2.9	2.9	3.0	3.0	3.2	3.4	3.1	3.0	2.8	2.7

Fuel Prices DataFrame



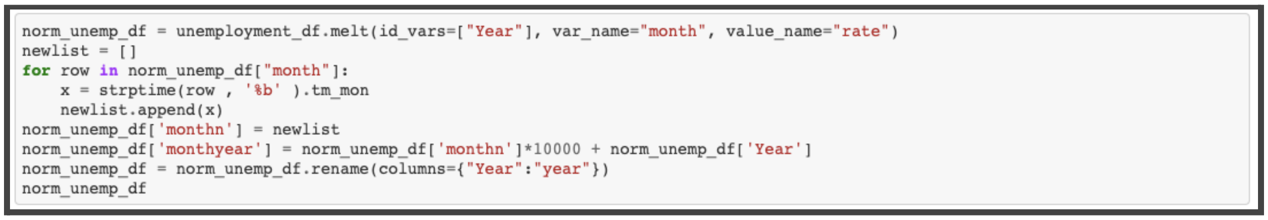
	Date	A1	A2	A3	R1	R2	R3	M1	M2	M3	P1	P2	P3	D1
0	01/02/1995	1.127	1.104	1.231	1.079	1.063	1.167	1.170	1.159	1.298	1.272	1.250	1.386	1.104
1	01/09/1995	1.134	1.111	1.232	1.086	1.070	1.169	1.177	1.164	1.300	1.279	1.256	1.387	1.102
2	01/16/1995	1.126	1.102	1.231	1.078	1.062	1.169	1.168	1.155	1.299	1.271	1.249	1.385	1.100
3	01/23/1995	1.132	1.110	1.226	1.083	1.068	1.165	1.177	1.165	1.296	1.277	1.256	1.378	1.095
4	01/30/1995	1.131	1.109	1.221	1.083	1.068	1.162	1.176	1.163	1.291	1.275	1.255	1.370	1.090

<u>Transform:</u>

Unemployment DataFrame:

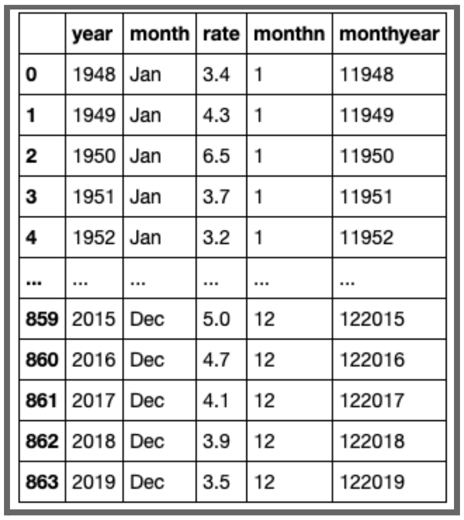
For the Unemployment DataFrame we needed to convert the formatting to match the Fuel DataFrame and wanted to have the unemployment rate to be displayed vertically with both month and year columns rather than horizontally. We also created a for loop to go through all of the string values for month and convert them to a numeric format (i.e. Jan to 1, Dec to 12) for a new column. Once converted we used a formula utilizing the numeric month and year columns to combine into a unique identifier for values for a specific month and year that was still descriptive (ex: January 1948 became 11948).

Code for End Result Unemployment DataFrame



```
norm_unemp_df = unemployment_df.melt(id_vars=["Year"], var_name="month", value_name="rate")
newlist = []
for row in norm_unemp_df["month"]:
    x = strptime(row , '%b' ).tm_mon
    newlist.append(x)
norm_unemp_df['monthn'] = newlist
norm_unemp_df['monthyear'] = norm_unemp_df['monthn']*10000 + norm_unemp_df['Year']
norm_unemp_df = norm_unemp_df.rename(columns={"Year":"year"})
norm_unemp_df
```

End Result Unemployment DataFrame

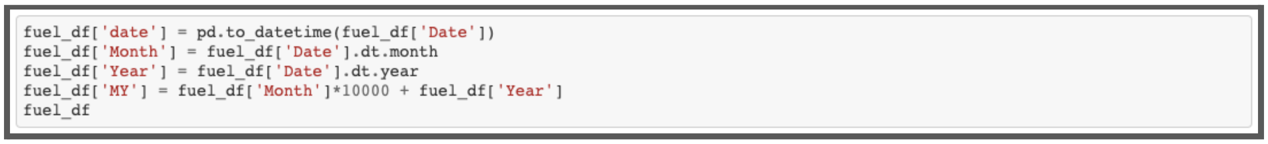


	year	month	rate	monthn	monthyear
0	1948	Jan	3.4	1	11948
1	1949	Jan	4.3	1	11949
2	1950	Jan	6.5	1	11950
3	1951	Jan	3.7	1	11951
4	1952	Jan	3.2	1	11952
...	...	...	...	...	...
859	2015	Dec	5.0	12	122015
860	2016	Dec	4.7	12	122016
861	2017	Dec	4.1	12	122017
862	2018	Dec	3.9	12	122018
863	2019	Dec	3.5	12	122019

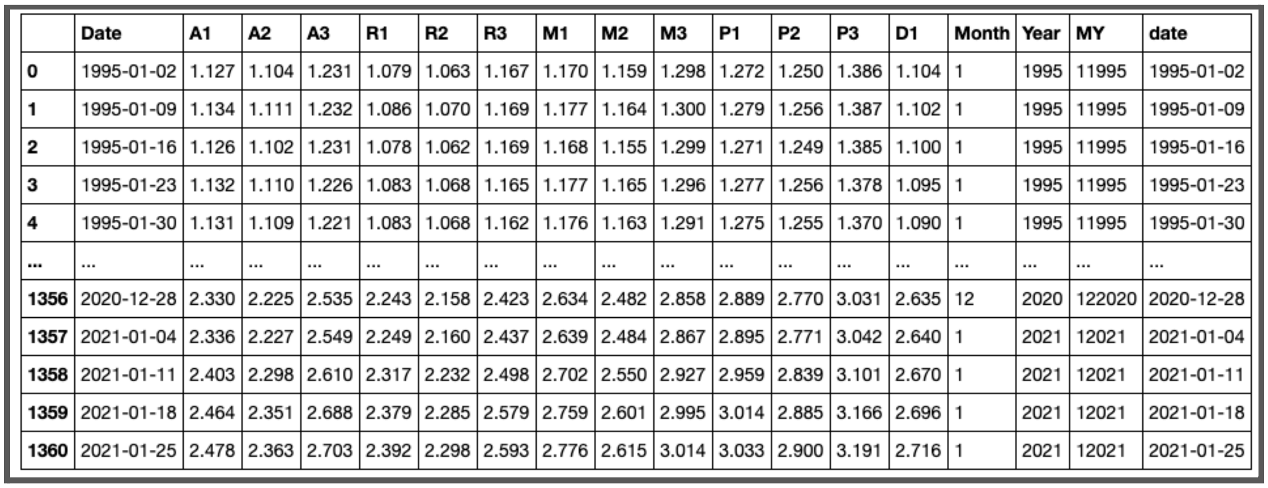
Fuel DataFrame:

For the fuel DataFrame we separated month and year into separate columns from the original date column in order to create the Month/Year(MY) column similar to the Unemployment DataFrame. This ultimately created a unique identifier for values that fall into a certain corresponding month and year. This also allowed us to group by the month/year values since we had multiple data points that fell into the same month and year.

Initial Code and Updated Fuel DataFrame



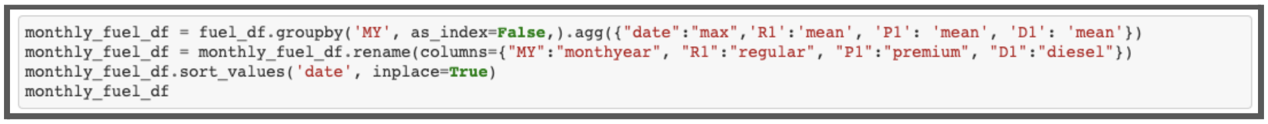
```
fuel_df['date'] = pd.to_datetime(fuel_df['Date'])
fuel_df['Month'] = fuel_df['Date'].dt.month
fuel_df['Year'] = fuel_df['Date'].dt.year
fuel_df['MY'] = fuel_df['Month']*10000 + fuel_df['Year']
fuel_df
```



	Date	A1	A2	A3	R1	R2	R3	M1	M2	M3	P1	P2	P3	D1	Month	Year	MY	date
0	1995-01-02	1.127	1.104	1.231	1.079	1.063	1.167	1.170	1.159	1.298	1.272	1.250	1.386	1.104	1	1995	11995	1995-01-02
1	1995-01-09	1.134	1.111	1.232	1.086	1.070	1.169	1.177	1.164	1.300	1.279	1.256	1.387	1.102	1	1995	11995	1995-01-09
2	1995-01-16	1.126	1.102	1.231	1.078	1.062	1.169	1.168	1.155	1.299	1.271	1.249	1.385	1.100	1	1995	11995	1995-01-16
3	1995-01-23	1.132	1.110	1.226	1.083	1.068	1.165	1.177	1.165	1.296	1.277	1.256	1.378	1.095	1	1995	11995	1995-01-23
4	1995-01-30	1.131	1.109	1.221	1.083	1.068	1.162	1.176	1.163	1.291	1.275	1.255	1.370	1.090	1	1995	11995	1995-01-30
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1356	2020-12-28	2.330	2.225	2.535	2.243	2.158	2.423	2.634	2.482	2.858	2.889	2.770	3.031	2.635	12	2020	122020	2020-12-28
1357	2021-01-04	2.336	2.227	2.549	2.249	2.160	2.437	2.639	2.484	2.867	2.895	2.771	3.042	2.640	1	2021	12021	2021-01-04
1358	2021-01-11	2.403	2.298	2.610	2.317	2.232	2.498	2.702	2.550	2.927	2.959	2.839	3.101	2.670	1	2021	12021	2021-01-11
1359	2021-01-18	2.464	2.351	2.688	2.379	2.285	2.579	2.759	2.601	2.995	3.014	2.885	3.166	2.696	1	2021	12021	2021-01-18
1360	2021-01-25	2.478	2.363	2.703	2.392	2.298	2.593	2.776	2.615	3.014	3.033	2.900	3.191	2.716	1	2021	12021	2021-01-25

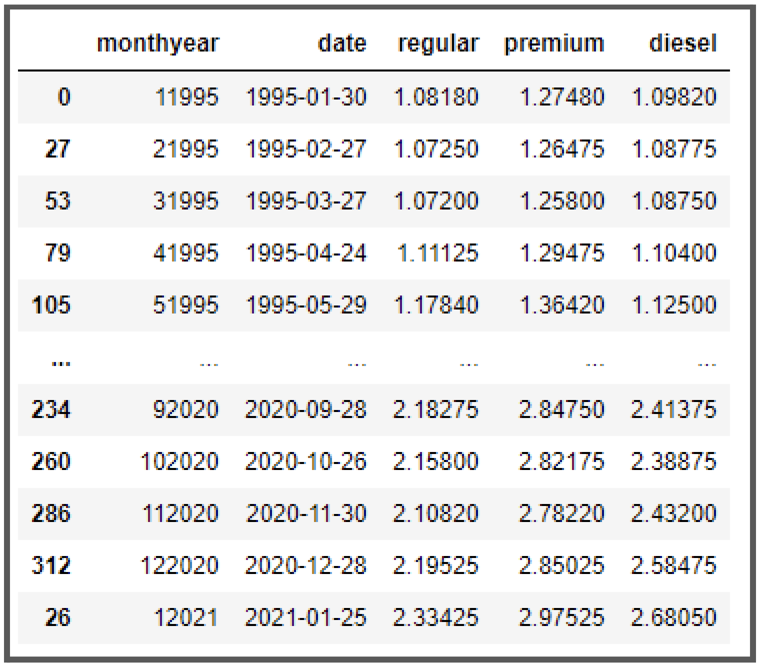
We grouped by the combined “MY” (month/year) field to aggregate the mean for the following: R1 (Regular fuel), P1 (Premium fuel) and D1 (Diesel) for each date value. We updated the columns to be more descriptive with “regular”, “premium”, “diesel” and “monthyear.” We then sorted the values by the date column in order to display the data in chronological order.

Code for Final Version of Fuel DataFrame



```
monthly_fuel_df = fuel_df.groupby('MY', as_index=False,).agg({"date":"max",'R1':'mean', 'P1': 'mean', 'D1': 'mean'})
monthly_fuel_df = monthly_fuel_df.rename(columns={"MY":"monthyear", "R1":"regular", "P1":"premium", "D1":"diesel"})
monthly_fuel_df.sort_values('date', inplace=True)
monthly_fuel_df
```

End Result Fuel DataFrame

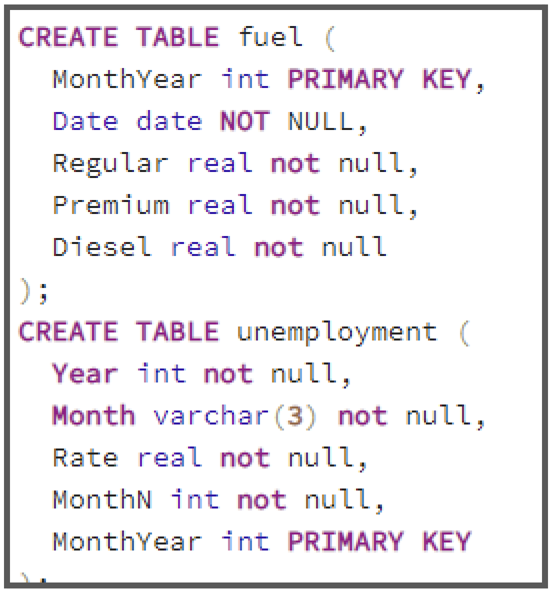


	monthyear	date	regular	premium	diesel
0	11995	1995-01-30	1.08180	1.27480	1.09820
27	21995	1995-02-27	1.07250	1.26475	1.08775
53	31995	1995-03-27	1.07200	1.25800	1.08750
79	41995	1995-04-24	1.11125	1.29475	1.10400
105	51995	1995-05-29	1.17840	1.36420	1.12500
...	...	...	...	...	...
234	92020	2020-09-28	2.18275	2.84750	2.41375
260	102020	2020-10-26	2.15800	2.82175	2.38875
286	112020	2020-11-30	2.10820	2.78220	2.43200
312	122020	2020-12-28	2.19525	2.85025	2.58475
26	12021	2021-01-25	2.33425	2.97525	2.68050

<u>Load:</u>

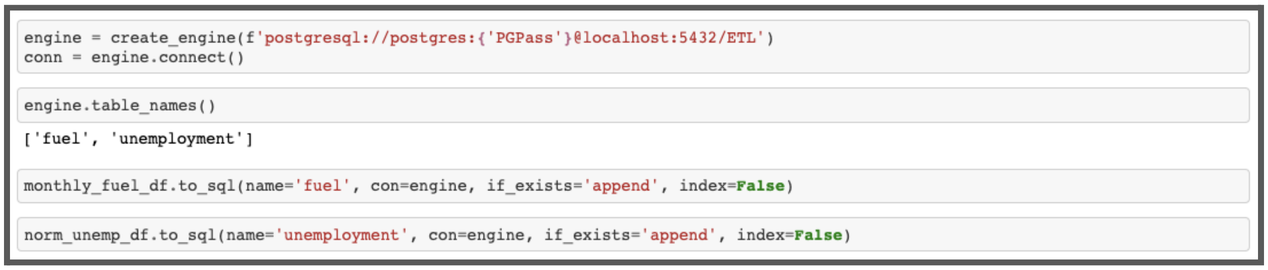
We decided to use a relational database as we had common factors between the two datasets (month/year). We utilized PgAdmin to create the Database and Tables that correspond with the DataFrames we built in our jupyter notebook, and then established the connection.

Creation of Tables in the Database



```
CREATE TABLE fuel (
  MonthYear int PRIMARY KEY,
  Date date NOT NULL,
  Regular real not null,
  Premium real not null,
  Diesel real not null
);
CREATE TABLE unemployment (
  Year int not null,
  Month varchar(3) not null,
  Rate real not null,
  MonthN int not null,
  MonthYear int PRIMARY KEY
```

Established Connection to Database and Load Data to Tables



```
engine = create_engine(f'postgresql://postgres:{'PGPass'}@localhost:5432/ETL')
conn = engine.connect()
```

```
engine.table_names()
```

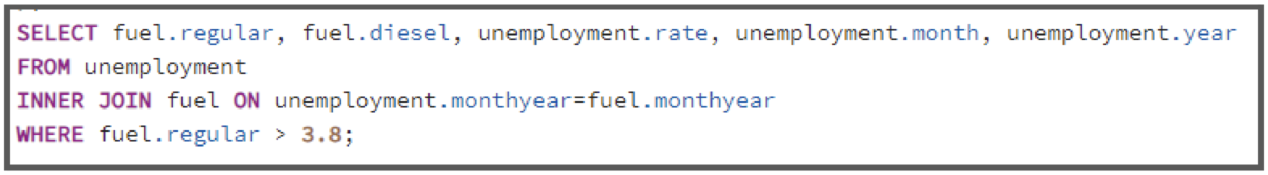
```
['fuel', 'unemployment']
```

```
monthly_fuel_df.to_sql(name='fuel', con=engine, if_exists='append', index=False)
```

```
norm_unemp_df.to_sql(name='unemployment', con=engine, if_exists='append', index=False)
```

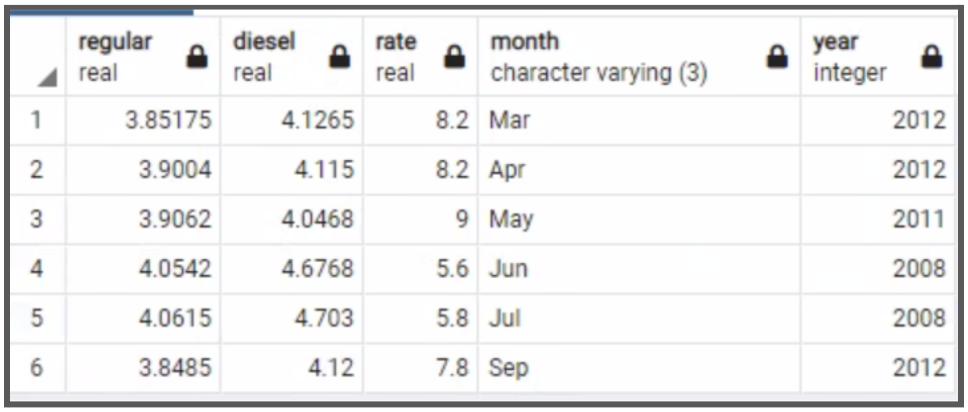
Once we'd established the tables and loaded the data, we performed a few queries to join the tables using our primary key (date/year). A couple example queries we ran to demonstrate the power of PostgreSQL include the below. One query was to see the unemployment rate, month, year and price of regular and diesel fuel for data points where regular fuel was greater than $3.80. Another query we ran was to display the price of regular and diesel fuel along with the corresponding month and year where unemployment was greater than 9.5%

Query Language for Regular Fuel > $3.8



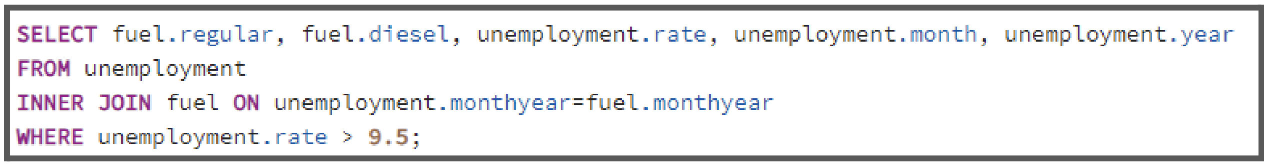
```
SELECT fuel.regular, fuel.diesel, unemployment.rate, unemployment.month, unemployment.year
FROM unemployment
INNER JOIN fuel ON unemployment.monthyear=fuel.monthyear
WHERE fuel.regular > 3.8;
```

Query Output



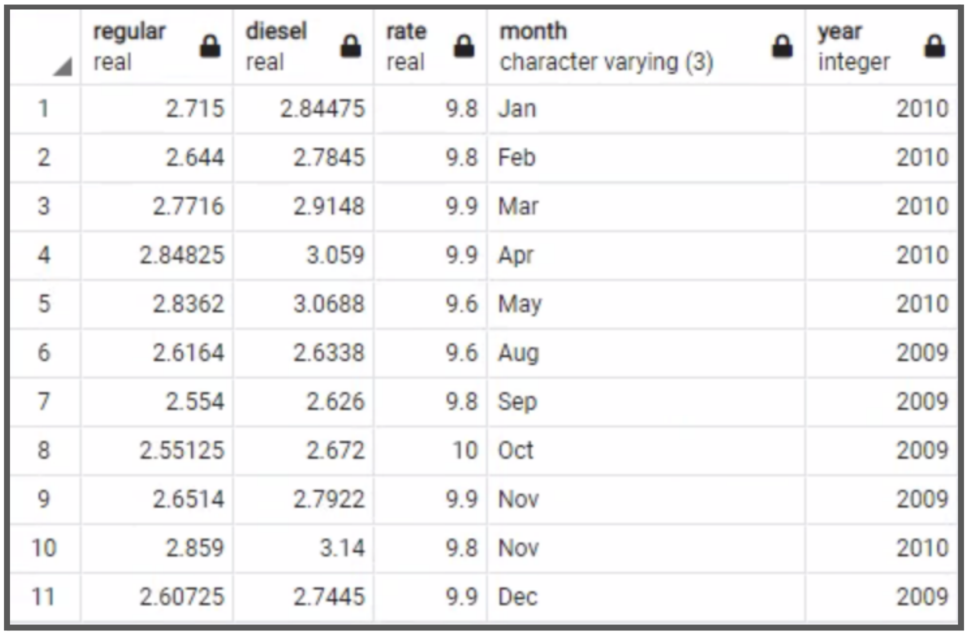
	regular real	diesel real	rate real	month character varying (3)	year integer
1	3.85175	4.1265	8.2	Mar	2012
2	3.9004	4.115	8.2	Apr	2012
3	3.9062	4.0468	9	May	2011
4	4.0542	4.6768	5.6	Jun	2008
5	4.0615	4.703	5.8	Jul	2008
6	3.8485	4.12	7.8	Sep	2012

Query Language for Unemployment Rate > 9.5%



```
SELECT fuel.regular, fuel.diesel, unemployment.rate, unemployment.month, unemployment.year
FROM unemployment
INNER JOIN fuel ON unemployment.monthyear=fuel.monthyear
WHERE unemployment.rate > 9.5;
```

Query Output



	regular real	diesel real	rate real	month character varying (3)	year integer
1	2.715	2.84475	9.8	Jan	2010
2	2.644	2.7845	9.8	Feb	2010
3	2.7716	2.9148	9.9	Mar	2010
4	2.84825	3.059	9.9	Apr	2010
5	2.8362	3.0688	9.6	May	2010
6	2.6164	2.6338	9.6	Aug	2009
7	2.554	2.626	9.8	Sep	2009
8	2.55125	2.672	10	Oct	2009
9	2.6514	2.7922	9.9	Nov	2009
10	2.859	3.14	9.8	Nov	2010
11	2.60725	2.7445	9.9	Dec	2009