Powering Through: Exploring the Dynamics of Electricity Demand in Relation to Urban Weather Patterns

Austin DeVore
Clayton Knight
Gabby Kruger
Wanderson Oliveira

Overview

Our project was to look into the effects of weather climates of a city and compare their demand of electricity in a given hour from July 2018 to May 2020.

We took the ETL route to create DataFrames that could be used to show what the electricity demand was for a city at a certain time and what the weather conditions were for that date and time.

We initially thought of doing the visual route, however from expert advice we were informed that might be a tad difficult.

Cleaning the Data

We received our original data from Kaggle. The data contained the electricity demand in the top 10 US cities by population and their weather conditions for each day by the hour.

This data contained multiple excels and jsons. Our first step was to combine the electricity information from the excels to contain all the electricity data of the 10 cities in one file.

After we had this, we continued to clean further in pandas.

Pandas Data Cleaning

Imported Libraries

```
# importing libraries
import pandas as pd
import numpy as np
pd.set_option('max_colwidth', 400)
from pytz import timezone
from datetime import datetime
```

Formatted the Date/Time

```
#format date/time
electric_data_df["Date/Time"]=pd.to_datetime(electric_data_df["Date/Time"]).dt.strftime("%Y-%m-%d %H:%M:%S")
electric_data_df.head(-20)
```

Panda Data Cleaning Cont.

Defined Time Zones of each city

```
# define local time zones for each city
city_timezones = {
    'nyc': 'America/New_York',
    'la': 'America/Los_Angeles',
    'dallas': 'America/Chicago',
    'houston': 'America/Chicago',
    'philadelphia': 'America/New_York',
    'phoenix': 'America/Phoenix',
    'san antonio': 'America/Chicago',
    'san diego': 'America/Los_Angeles',
    'san jose': 'America/Los_Angeles',
    'seattle': 'America/Los_Angeles'
}
```

Convert the date/times to central time

zone

```
#convert datetime to Central Time
def convert_to_central(dt_str, city):
    local_tz = timezone(city_timezones[city])
    dt = datetime.strptime(dt_str, '%Y-%m-%d %H:%M:%S')
    dt_local = local_tz.localize(dt)
    dt_central = dt_local.astimezone(timezone('America/Chicago'))
    return dt_central
```

Panda Data Cleaning Cont.

Convert those times to central time zone

```
#convert 'Date/Time' column to Central Time
electric_data_df['Central Time'] = electric_data_df.apply(lambda row: convert_to_central(row['Date/Time'], row['City']), axis=1)
electric_data_df.head(-20)
```

Convert date/time column to Unix Time format

```
#convert 'Date/Time' column to Unix time
electric_data_df['Unix Time'] = pd.to_datetime(electric_data_df['Central Time']).astype('int64') // 10**9
electric_data_df.head(-20)
```

Create city dataframe (x10)

```
# create nyc data frame
nyc_df = electric_data_df[electric_data_df["city"]=="nyc"]
nyc_df.reset_index(drop=True)
```

Panda Data Cleaning Cont.

- Remove DST times
- Remove duplicate dates

```python
# dst date/times
dates_to_remove = ['2018-03-11 01:00:00', '2018-11-04 01:00:00', '2019-03-10 01:00:00', '2019-11-03 01:00:00', '2020-03-08 01:00:00']
```

```python
# drop dst dates to remove from cities
nyc_df = nyc_df[~nyc_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
la_df = la_df[~la_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
dallas_df = dallas_df[~dallas_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
houston_df = houston_df[~houston_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
philadelphia_df = philadelphia_df[~philadelphia_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
san_antonio_df = san_antonio_df[~san_antonio_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
san_diego_df = san_diego_df[~san_diego_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
san_jose_df = san_jose_df[~san_jose_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
seattle_df = seattle_df[~seattle_df['date_time'].isin(dates_to_remove)].reset_index(drop=True)
```

```python
# dedup
nyc_df = nyc_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
la_df = la_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
dallas_df = dallas_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
houston_df = houston_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
philadelphia_df = philadelphia_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
phoenix_df = phoenix_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
san_antonio_df = san_antonio_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
san_diego_df = san_diego_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
san_jose_df = san_jose_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
seattle_df = seattle_df.drop_duplicates(subset=['date_time']).reset_index(drop=True)
```

Kick it over to SQL

Created tables for electricity and weather conditions

```sql
CREATE TABLE "dallas" (
    "demand" DECIMAL,
    "date_time" TIMESTAMP NOT NULL,
    "city" VARCHAR NOT NULL,
    "unix_time" INTEGER PRIMARY KEY
);
```

```sql
CREATE TABLE "dallas_climate" (
    "time" INTEGER PRIMARY KEY,
    "summary" VARCHAR(255),
    "icon" VARCHAR(50),
    "precip_intensity" FLOAT,
    "precip_probability" FLOAT,
    "temperature" DECIMAL,
    "apparent_temperature" DECIMAL,
    "dewpoint" DECIMAL,
    "humidity" DECIMAL,
    "pressure" DECIMAL,
    "wind_speed" DECIMAL,
    "wind_gust" DECIMAL,
    "wind_bearing" DECIMAL,
    "cloud_cover" DECIMAL,
    "uv_index" DECIMAL,
    "visibility" DECIMAL,
    "precip_type" VARCHAR(50),
    "ozone" DECIMAL,
    "precip_accumulation" DECIMAL
);
```

SQL Cont.

Joined the two SQL queries based on the primary key “unix_time” and then exported to CSV

Query | Query History

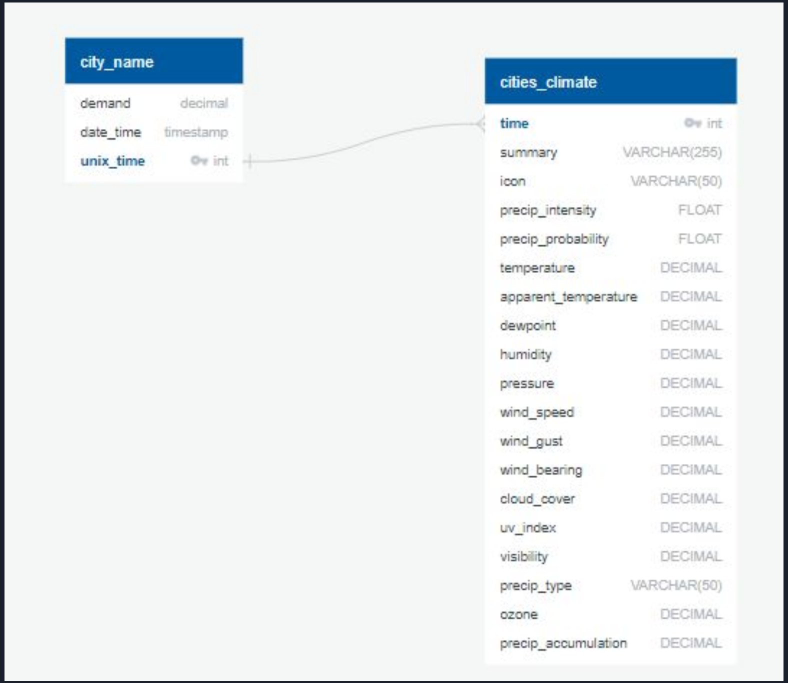
```sql
Select * from philadelphia
join philadelphia_climate
on unix_time = time
```

Data Output | Messages | Notifications

	demand numeric	date_time timestamp without time zone	city character varying	unix_time integer	time integer	summary character varying (255)	icon character varying (50)	precip_intensity double precision	precip_probability double precision	temperature numeric	apparent_temperature numeric	dewpoint numeric
1	407074.0	2018-07-02 00:00:00	philadelphia	1530504000	1530504000	Clear	clear-night	0	0	80.27	83.79	70.79
2	43907.0	2018-07-02 01:00:00	philadelphia	1530507600	1530507600	Clear	clear-night	0	0	79.74	82.67	69.59
3	4423.0	2018-07-02 02:00:00	philadelphia	1530511200	1530511200	Clear	clear-night	0	0	78.14	79.19	69.7
4	40743.0	2018-07-02 03:00:00	philadelphia	1530514800	1530514800	Clear	clear-night	0	0	76.94	78.02	69.75
5	5230.0	2018-07-02 04:00:00	philadelphia	1530518400	1530518400	Clear	clear-night	0	0	76.07	77.2	69.96
6	50752.0	2018-07-02 05:00:00	philadelphia	1530522000	1530522000	Clear	clear-night	0	0	74.97	76.09	69.58
7	6254.0	2018-07-02 06:00:00	philadelphia	1530525600	1530525600	Clear	clear-day	0	0	73.54	74.71	69.61
8	6669.0	2018-07-02 07:00:00	philadelphia	1530529200	1530529200	Clear	clear-day	0	0	75.07	76.33	70.69
9	6964.0	2018-07-02 08:00:00	philadelphia	1530532800	1530532800	Humid	clear-day	0	0	79.62	83.36	72.64
10	7228.0	2018-07-02 09:00:00	philadelphia	1530536400	1530536400	Humid	clear-day	0	0	83.67	90.23	73.47
11	7380.0	2018-07-02 10:00:00	philadelphia	1530540000	1530540000	Humid	clear-day	0.0004	0.01	86.65	94.5	73.57
12	7480.0	2018-07-02 11:00:00	philadelphia	1530543600	1530543600	Humid	clear-day	0.0009000000000000001	0.01	90.16	99.1	73.54
13	7559.0	2018-07-02 12:00:00	philadelphia	1530547200	1530547200	Clear	clear-day	0	0	92.05	98.42	70.5
14	7564.0	2018-07-02 13:00:00	philadelphia	1530550800	1530550800	Clear	clear-day	0	0	93.71	100.93	71.0
15	7454.0	2018-07-02 14:00:00	philadelphia	1530554400	1530554400	Clear	clear-day	0.0027	0.01	94.92	101.89	70.45
16	7260.0	2018-07-02 15:00:00	philadelphia	1530558000	1530558000	Clear	clear-day	0	0	96.1	103.25	70.32
17	7043.0	2018-07-02 16:00:00	philadelphia	1530561600	1530561600	Clear	clear-day	0	0	96.56	103.75	70.25
18	68607.0	2018-07-02 17:00:00	philadelphia	1530565200	1530565200	Clear	clear-day	0.0016	0.01	96.8	103.98	70.16
19	6436.0	2018-07-02 18:00:00	philadelphia	1530568800	1530568800	Clear	clear-day	0.0009000000000000001	0.01	96.42	103.38	70.04
20	59907.0	2018-07-02 19:00:00	philadelphia	1530572400	1530572400	Clear	clear-day	0.0006000000000000001	0.01	94.48	101.68	70.79
21	55407.0	2018-07-02 20:00:00	philadelphia	1530576000	1530576000	Humid	clear-day	0.0007	0.01	91.01	99.29	72.71
22	5225.0	2018-07-02 21:00:00	philadelphia	1530579600	1530579600	Humid	clear-night	0.0006000000000000001	0.01	87.27	95.37	73.61
23	49076.0	2018-07-02 22:00:00	philadelphia	1530583200	1530583200	Humid	clear-night	0.0004	0.01	84.58	91.18	73.03
24	4832.0	2018-07-02 23:00:00	philadelphia	1530586800	1530586800	Humid	clear-night	0	0	82.49	87.95	72.69
25	407074.0	2018-07-03 00:00:00	philadelphia	1530590400	1530590400	Humid	clear-night					

✓ Successfully run. Total query runtime: 641 msec. 16473 rows affected.

ERD


city_name
demand decimal
date_time timestamp
unix_time int
cities_climate
time int
summary VARCHAR(255)
icon VARCHAR(50)
precip_intensity FLOAT
precip_probability FLOAT
temperature DECIMAL
apparent_temperature DECIMAL
dewpoint DECIMAL
humidity DECIMAL
pressure DECIMAL
wind_speed DECIMAL
wind_gust DECIMAL
wind_bearing DECIMAL
cloud_cover DECIMAL
uv_index DECIMAL
visibility DECIMAL
precip_type VARCHAR(50)
ozone DECIMAL
precip_accumulation DECIMAL

Issues Faced

- Formatting of dates: JSON was in Unix Time, where as the excels were in yyyy-mm-dd h:mm
- Getting correct Unix Time: we had to convert all times to central time zone in order to get correct Unix Time
- Daylight Savings: There were duplicate dates because of daylight savings causing complications importing into SQL
- Importing the CSV into SQL: SQL was not accepting the CSV as their were NULL values in the “demand” column

Ways This Data Could Be Used

The following are just a few ways that a power company could use this type of data.

- Plans for their resources
- Energy efficiency programs
- Grid infrastructure