Capital Bikeshare

Table of Contents

1. Introduction
2. Data and Cleaning
   - 2.1 Import Libraries
   - 2.2 Create SQL Database Connections and Table
   - 2.3 Clean CSV Files & Load to SQL
   - 2.4 Load SQL Data to DataFrame & Clean
3. Data Analysis
   - 3.1 Compare Total Rides Over Time
   - 3.2 Compare Total Rides by Time of Day
   - 3.3 Compare Total Rides by Day, Year
   - 3.4 Most Popular Ride Stations
   - 3.5 Number of Rides per Station (2019-Present)
   - 3.6 Compare Rides by Pandemic Period (Non-Just By Station)

0. Introduction:

The purpose of this project is to use data science tools to analyze Capital Bikeshare usage around Washington, DC before and during the COVID-19 pandemic. Specifically, I will be looking at the following questions:

- What is the total number of rides since bikeshare? Since COVID-19?
- How are the patterns in ridership different between before and during COVID-19?
- Are ridership patterns different by time of day, day of the week, or month of the year?
- Which stations have the most ridership? Which stations have the most ridership during the pandemic?
- Are bikeshare trips more for commuting or for recreational purposes?
- Can forecasting models predict future demand during COVID-19 and after?
- Can forecasting models predict future demand during COVID-19 and after?

The data I will need to accomplish this project is Capital Bike Share Data, Weather Data, Holiday Data, and Daily COVID-19 Data. Capital Bike Share Data is available for download here. Weather data is available through the NOAA API. Holiday data is available through the Python "holidays" package. Finally, COVID-19 data is available through the Python "covidcast" package.

Since this pandemic began deep into the analysis, I decided to be inspired by the **FiveThirtyEight** articles and use a blue-to-white-to-red color scheme when creating visualizations. Since it is ongoing for and after a 100- hundred- year, I decided to use a wide palette through all the 100 years of data.

1. Data Input & Cleaning:

Data is provided by Capital Bike Share (https://www.capitalbikeshare.com/system-data) on a monthly basis and provides all trips completed by month since the beginning of Capital Bike in 2010.

1.1 Import Libraries:

```
# Jupyter Extensions
%load_ext autoreload
%autoreload 2
%load_ext sql
%matplotlib inline

# Numpy and Pandas
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

# Matplotlib
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import matplotlib.colors as mcolors
import matplotlib.gridspec as gridspec, time
%matplotlib inline

# Seaborn
import seaborn as sns

# Calendar, Holidays, Requests, Urllib, Time
import calendar as cal
import holidays
import requests
import urllib.request, urllib.error, urllib.parse, time

# Sqlalchemy
from sqlalchemy.dialects.mysql.pymysql import (BIGINT, BINARY, BLOB, BOOLEAN,
CHAR, DATE, DATETIME, DECIMAL, DECIMAL, DOUBLE, ENUM, FLOAT, INTEGER,
LONGBLOB, LONGTEXT, MEDIUMBLOB, MEDIUMINT, MEDIUMTEXT, NCHAR,
NUMERIC, NVARCHAR, PYMYSQL_CHARSETS, REAL, SMALLINT,
TEXT, TIME, TIMESTAMP, TINYBLOB, TINYINT, TINYTEXT, VARBINARY,
VARCHAR, YEAR)
from sqlalchemy.dialects.mysql.base import (dialect, MariaDB, MariaDBIdentifierPreparer,
MariaDBCompiler, MariaDBDDLCompiler, MariaDBTypeCompiler, MariaDBExecutionContext,
MariaDBIdentifierPreparer, MariaDBIntroSpector, MariaDBDefaultRunner,
MariaDBSchemaDropper, MariaDBSchemaGenerator, MariaDBTypeIdentifier,
MariaDBInformationSchemaDialect, VARBINARY2_5, BIT2_5_BIN,
MariaDBExecutionContext_10_2_mariadb, MariaDBCompiler_10_2_mariadb,
MariaDBIntroSpector_10_2_mariadb, MariaDBDialect_10_2_mariadb,
MariaDBIdentifierPreparer_10_2_mariadb, MariaDBTypeCompiler_10_2_mariadb,
MariaDBDDLCompiler_10_2_mariadb, MariaDBSchemaGenerator_10_2_mariadb,
MariaDBSchemaDropper_10_2_mariadb, MariaDBDefaultRunner_10_2_mariadb,
MariaDBIntroSpector_10_3_mariadb, MariaDBDialect_10_3_mariadb,
MariaDBIdentifierPreparer_10_3_mariadb, MariaDBTypeCompiler_10_3_mariadb,
MariaDBDDLCompiler_10_3_mariadb, MariaDBSchemaGenerator_10_3_mariadb,
MariaDBSchemaDropper_10_3_mariadb, MariaDBDefaultRunner_10_3_mariadb,
MariaDBIdentifierPreparer_10_4_mariadb, MariaDBIntroSpector_10_4_mariadb,
MariaDBDialect_10_4_mariadb, MariaDBExecutionContext_10_4_mariadb,
MariaDBCompiler_10_4_mariadb, MariaDBTypeCompiler_10_4_mariadb,
MariaDBDDLCompiler_10_4_mariadb, MariaDBSchemaGenerator_10_4_mariadb,
MariaDBSchemaDropper_10_4_mariadb, MariaDBDefaultRunner_10_4_mariadb,
VARCHAR2, NVARCHAR2, VARCHAR5, RAW, BOOLEAN5, INTERVAL DAY TO SECOND)
from sqlalchemy.dialects.mysql.mysqlconnector import (MySQLCompiler_mysqlconnector,
MySQLIdentifierPreparer_mysqlconnector, MySQLExecutionContext_mysqlconnector,
MySQLDialect_mysqlconnector)

# Scikit-Learn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_regression import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_val_score, KFold
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils import resample
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin

# Statsmodels
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
import statsmodels.tsa.seasonal as seas
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.arima_model as arima
import statsmodels.tsa.statespace.sarimax as sarimax
import statsmodels.formula.api as smf
import statsmodels.tsa.holtwinters as hw

# Fbprophet
import fbprophet

# Warnings
import warnings
warnings.filterwarnings('ignore')
```

1.2 Create SQL Database Connections and Table

```
# Jupyter Extension
%load_ext autoreload
%autoreload 2
%load_ext sql

# Variable Strings for MySQL Connection
rides_conn = 'mysql+pymysql://root:bikeshare@localhost:3306/rides?charset=utf8mb4'
weather_conn = 'mysql+pymysql://root:bikeshare@localhost:3306/weather?charset=utf8mb4'
covid_conn = 'mysql+pymysql://root:bikeshare@localhost:3306/covid?charset=utf8mb4'

# Create Table for Ridership Data
create table trips (
                    rideid varchar(16),
                    rtype varchar(5),
                    start_time datetime,
                    end_time datetime,
                    scode varchar(5),
                    ecode varchar(5),
                    bike varchar(8),
                    btype varchar(13),
                    zipcode varchar(5)
                    );
```

1.3 Clean CSV Files & Load to SQL

```
# Python Extension for cell magic definition
def cell_time(line, cell):
    ipy = get_ipython()
    ipy.ast_run_cell(cell)
    return 'Time: {:.2f} s'.format(ipy.user_ns['_'].total_seconds())

# Define Variables
dir = '/Data/Capital Bikeshare/'
ridefiles = ['2010-capitalbikeshare-tripdata.csv', '2011-capitalbikeshare-tripdata.csv',
             '2012-capitalbikeshare-tripdata.csv', '2013-capitalbikeshare-tripdata.csv',
             '2014-capitalbikeshare-tripdata.csv', '2015-capitalbikeshare-tripdata.zip',
             '2016-capitalbikeshare-tripdata.zip', '2017-capitalbikeshare-tripdata.zip',
             '2018-capitalbikeshare-tripdata.zip', '201801_capitalbikeshare_tripdata.zip',
             '201802_capitalbikeshare_tripdata.zip', '201803_capitalbikeshare_tripdata.zip',
             '201804_capitalbikeshare_tripdata.zip', '201805_capitalbikeshare_tripdata.zip',
             '201806_capitalbikeshare_tripdata.zip', '201807_capitalbikeshare_tripdata.zip',
             '201808_capitalbikeshare_tripdata.zip', '201809_capitalbikeshare_tripdata.zip',
             '201810_capitalbikeshare_tripdata.zip', '201811_capitalbikeshare_tripdata.zip',
             '201812_capitalbikeshare_tripdata.zip']
header = ['rideid', 'rtype', 'start_time', 'end_time', 'scode', 'ecode', 'bike', 'btype', 'zipcode']

# Create Function to Clean Ride Files and Write to Temp CSV
def cleanfiles(ridefile, header):
    ipy = get_ipython()
    print('Reading {}'.format(ridefile))
    ...
    df = pd.read_csv(dir + ridefile, ...)
    ...
    if len(df.columns) == 9:
        df = df[['rideid', 'rtype', 'start_time', 'end_time', 'scode', 'ecode', 'bike', 'btype', 'zipcode']]
    ...
    else:
        raise ...
    ...
    print('Writing {}_temp.csv'.format(ridefile))
    ...
    df.to_csv(dir + ridefile + '_temp.csv', ...)
    ...
    return df

# Create Function to Write Temp CSV Files to SQL Database
def writesql(ridefile):
    ipy = get_ipython()
    ...
    print('Writing {}_temp.csv to SQL Database'.format(ridefile))
    ...
    ipy.run_line_magic('sql', '{} --dsn=rides:root:bikeshare@localhost:3306/rides --persist --table trips --dsn_connect_timeout=600 --dsn_connect_options="local_infile=1" --dsn_connect_options="connect_timeout=600" --dsn_connect_options="max_allowed_packet=1073741824" --dsn_connect_options="interactive_timeout=600" --dsn_connect_options="net_read_timeout=600" --dsn_connect_options="net_write_timeout=600" --dsn_connect_options="wait_timeout=600" --dsn_connect_options="max_allowed_packet=1073741824"
```

Change in Rides by Station (Mar-Aug 2019 vs. Mar-Aug 2020)

Lincoln Park
Near North
Near West
Near South
Loop
Near West
Near South
Loop
Near North
Near West
Near South
Loop
Near West
Near South
Loop
Near North
Near West
Near South
Loop
Near West
Near South
Loop

```
[INFO] Executing: /home/ec2-user/anaconda3/envs/python3/bin/jupyter-nbconvert --to notebook --execute --
[NbConvertApp] Converting notebook notebook.ipynb to notebook
[NbConvertApp] Writing 1.1.1.1 MB to /home/ec2-user/SageMaker/project-bikeshare/notebook.nbconvert.ipynb
[NbConvertApp] Executed notebook.ipynb in 0:00:00.000000
```

	count	0_2	RMSE	MAE
0	mae(frequency_days)	0.000000	0.000000	0.000000
1	mae(holiday)	0.000000	0.000000	0.000000
2	mae(temp)	0.000000	0.000000	0.000000
3	mae(weekend)	0.000000	0.000000	0.000000

3.0 Compare All Models

The following table shows the metrics for each model.

```
model_comparison
```

	count	0_2	RMSE	MAE
0	boosted(count, cat_cols, lag_cols)	0.000000	0.000000	0.000000
1	boosted(count, cat_cols, lag_cols)_tuned	0.000000	0.000000	0.000000
2	boosted(count, cat_cols)	0.000000	0.000000	0.000000
3	boosted(count, cat_cols)_tuned	0.000000	0.000000	0.000000

4. Data Pipeline Using Docker, Airflow, AWS S3, AWS RDS

4.0 Summary

As a capstone effort within this project, a data pipeline was created to automate ETL process for bikeshare rides. The steps of the ETL pipeline include:

- create a database for rides and stations
- create a database table for weather (using weather forecast API)
- create a database table for holidays (using holiday API) and load the holiday database
- create a table to get the ID values from the station database
- ETL the bikeshare ride data
- ETL the weather data

The below Directed Acyclic Graph (DAG) displays the dependencies of files and their name.

4.1 Dockerfile and Docker Compose

The Dockerfile and Docker Compose files are used to create the files within the docker folder, copying requirements.txt, copying a bikeshare file (to create bikeshare.py), the use of environmental variables, and running requirements file within the container. The docker compose spin-ups configures the docker application as follows: NGYK: (webs) the webserver, Redis: (broker), Flower: (monitor), Postgres: (database), Scheduler: (schedules jobs), Worker: (runs the jobs). The Dockerfile is more detailed and includes AWS CLI and ECS-CLI in order to interface with AWS.

4.2 Holidays DAG

```
# [START def_add_one]
def add_one(x):
    print("Mapping {}".format(x))
    return x + 1
# [END def_add_one]
# [START def_add_sum]
def add_sum(x, y):
    print("Reducing {} and {}".format(x, y))
    return x + y
# [END def_add_sum]
# [START def_print_result]
def print_result(sum):
    print("Result: {}".format(sum))
    return sum
# [END def_print_result]

# [START dag_definition]
with DAG('airflow_tutorial_v01',
         default_args=default_args,
         schedule_interval='0 0 * * *',
         dsrunid = '{{ dag_run.id }}',
         dsowner = '{{ dag_run.conf.owner }}',
         dsemail = '{{ dag_run.conf.email }}',
         dsmodel = '{{ dag_run.conf.model }}',
         dsstart = '{{ dag_run.conf.start }}',
         dsstop = '{{ dag_run.conf.stop }}',
         catchup=False,
         max_active_runs=1,
         concurrency=2
         ) as dag:
# [END dag_definition]

# [START dag_variables]
    home = Variable.get("home")
    conf = Variable.get("airflow_tutorial_conf")
    docker_image = "{{ dag_run.conf.docker_image }}"
    docker_network = "{{ dag_run.conf.docker_network }}"
    docker_command_prefix = "/usr/local/bin/docker run -t --rm -v /var/run/docker.sock:/var/run/docker.sock:ro --
    docker_command_holiday = docker_command_prefix + "holiday"
# [END dag_variables]

# [START dag_objects_holiday]
    create_database = BashOperator(task_id='create_database',
                                   bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/create_db
    create_holiday_table = BashOperator(task_id='create_holiday_table',
                                        bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/cre
    load_holiday_table = BashOperator(task_id='load_holiday_table',
                                      bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/load
# [END dag_objects_holiday]

# [START dag_objects_weather]
    create_weather_table = BashOperator(task_id='create_weather_table',
                                        bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/cre
    load_weather_table = BashOperator(task_id='load_weather_table',
                                      bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/load
# [END dag_objects_weather]

# [START dag_objects_station]
    create_station_table = BashOperator(task_id='create_station_table',
                                        bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/cre
    create_station_id_table = BashOperator(task_id='create_station_id_table',
                                           bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/
    load_station_id_table = BashOperator(task_id='load_station_id_table',
                                         bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/
# [END dag_objects_station]

# [START dag_objects_rides]
    create_rides_table = BashOperator(task_id='create_rides_table',
                                      bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/cre
    load_rides_table = BashOperator(task_id='load_rides_table',
                                    bash_command='cd {{ dag_run.conf.project_dir }}/docker/bikeshare/load
# [END dag_objects_rides]

# [START dag_dependencies]
    create_database >> create_holiday_table
    create_holiday_table >> load_holiday_table
    create_weather_table >> load_weather_table
    create_station_table >> create_station_id_table
    create_station_id_table >> load_station_id_table
    create_rides_table >> load_rides_table
    load_station_id_table >> load_rides_table
# [END dag_dependencies]
```

4.3 WriteConsole1 Class

```
# [START def_write_to_db_one]
def write_to_db(conn):
    cur = conn.cursor()
    cur.execute("""INSERT INTO ride_id(ride_id,rental_id,bike_id,rental_date,rental_hour,rental_month,duration
                   VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s
                   ON CONFLICT (ride_id) DO UPDATE
                   SET (ride_id,rental_id,bike_id,rental_date,rental_hour,rental_month,duration,duration_
                   WHERE ride_id = %s;""",(ride[0],ride[1],ride[2],ride[3],ride[4],ride[5],ride[6],ride[7],ride[8],ride[9],
                                           
```