climate_starter

November 2, 2019

[1]:
```
%matplotlib inline
from matplotlib import style
style.use('fivethirtyeight')
import matplotlib.pyplot as plt
```

[2]:
```
import numpy as np
import pandas as pd
```

[3]:
```
import datetime as dt
```

1 Reflect Tables into SQLAlchemy ORM

[4]:
```
# Python SQL toolkit and Object Relational Mapper
import sqlalchemy
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine, func
```

[5]:
```
engine = create_engine("sqlite:///Resources/hawaii.sqlite")
```

[6]:
```
# reflect an existing database into a new model
Base = automap_base()
# reflect the tables
Base.prepare(engine, reflect=True)
```

[7]:
```
# We can view all of the classes that automap found
Base.classes.keys()
```

[7]:
```
['measurement', 'station']
```

[8]:
```
# Save references to each table
Measurement = Base.classes.measurement
Station = Base.classes.station
```

[9]:
```
# Create our session (link) from Python to the DB
session = Session(engine)
```

2 Exploratory Climate Analysis

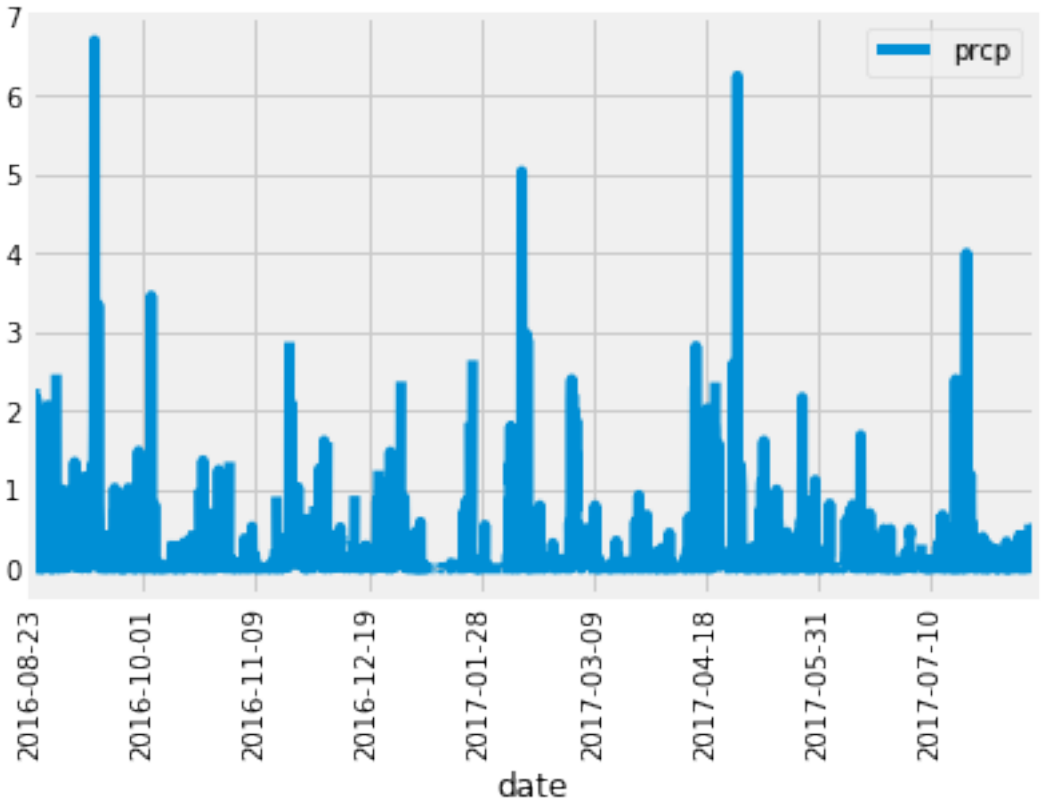
[10]:
```
first_row = session.query(Measurement).first()
first_row.__dict__
```

[10]:
```
{'_sa_instance_state': <sqlalchemy.orm.state.InstanceState at 0x1f49599aeb8>,
 'prcp': 0.08,
 'station': 'USC00519397',
 'tobs': 65.0,
 'date': '2010-01-01',
 'id': 1}
```

[11]:
```
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates

# Design a query to retrieve the last 12 months of precipitation data and plot␣
 ↪the results

# Calculate the date 1 year ago from the last data point in the database
total = session.query(func.count(Measurement.date)).all()
# total = session.query(Measurement).filter(Measurement.date >= '2010-01-01').
 ↪count()
# annual = session.query(Measurement).filter(Measurement.date.
 ↪between('2010-01-01','2011-01-01')).count()
# years = total/annual
#years =7.00 so adding 7 to 2010 gives us the 2017 year

# Perform a query to retrieve the date and precipitation scores
# guessing and checking in the following query gives 2017-08-23 as the last day␣
 ↪in the dataset
last_day = session.query(Measurement.date).order_by(Measurement.date.desc()).
 ↪first()
rows = []
for row in session.query(Measurement.date, Measurement.prcp).\
    filter(Measurement.date.between('2016-08-23','2017-08-23')).all():  #.
 ↪limit(15)
    rows.append(row)

    # Save the query results as a Pandas DataFrame and set the index to the␣
 ↪date column
row_df = pd.DataFrame.from_dict(rows)
row_df = row_df.set_index("date")
row_df = row_df.sort_index()
# row_df
row_df.plot(rot=90)
# Sort the dataframe by date
# row_df.sort_values(by='date', ascending=False)
```

```
# row_df

# Use Pandas Plotting with Matplotlib to plot the data
# fig, ax = plt.subplots(figsize=(12, 8))
# ax.set(xlabel = "Date",ylabel = "Precipitation", title="Daily Precipitation
 ↪in Honolulu from Aug 2016 - Aug 2017")
# # ax.plot(row_df.index.values, row_df['prcp'],ls='-', marker='o')
# # ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
# # ax.xaxis.set_major_formatter(DateFormatter("%m-%d"))
# plt.show()
```

[11]: `<matplotlib.axes._subplots.AxesSubplot at 0x1f4959eeeb8>`



precipitation

[12]:
```
# Use Pandas to calcualte the summary statistics for the precipitation data
row_df.describe()
```

[12]:

	prcp
count	2021.000000
mean	0.177279

```
std       0.461190
min       0.000000
25%       0.000000
50%       0.020000
75%       0.130000
max       6.700000
```

describe

[13]:
```
first_row = session.query(Station).first()
first_row.__dict__
```

[13]:
```
{'_sa_instance_state': <sqlalchemy.orm.state.InstanceState at 0x1f496ee1940>,
 'name': 'WAIKIKI 717.2, HI US',
 'id': 1,
 'elevation': 3.0,
 'latitude': 21.2716,
 'station': 'USC00519397',
 'longitude': -157.8168}
```

[14]:
```
# Design a query to show how many stations are available in this dataset?
session.query(Station.station).count()
session.query(Station.station, func.count(Station.station)).all()
```

[14]:
```
[('USC00519397', 9)]
```

[24]:
```
# What are the most active stations? (i.e. what stations have the most rows)?
# List the stations and the counts in descending order.
# session.query(Measurement.station, Measurement.tobs).group_by(Measurement.
 ↪station).order_by(Measurement.tobs).all()
#sel = [Measurement.prcp, Measurement.station, Measurement.tobs, Measurement.
 ↪date, Measurement.id]
rows = []
for row in session.query(Measurement.prcp, Measurement.station, Measurement.
 ↪tobs, Measurement.date, Measurement.id).all():
    rows.append(row)
row_df = pd.DataFrame.from_dict(rows)
row_df.station.value_counts()
session.query(Measurement.station,func.count(Measurement.station)).
 ↪group_by(Measurement.station).\
    order_by(func.count(Measurement.station).desc()).all()
```

[24]:
```
[('USC00519281', 2772),
 ('USC00519397', 2724),
 ('USC00513117', 2709),
 ('USC00519523', 2669),
 ('USC00516128', 2612),
 ('USC00514830', 2202),
```

```
 ('USC00511918', 1979),
 ('USC00517948', 1372),
 ('USC00518838', 511)]
```

[16]:
```
# Using the station id from the previous query, calculate the lowest␣
 ↪temperature recorded,
# highest temperature recorded, and average temperature of the most active␣
 ↪station?
most_active = row_df.station.value_counts().idxmax()
session.query(func.min(Measurement.tobs), func.max(Measurement.tobs), func.
 ↪avg(Measurement.tobs)).\
    filter(Measurement.station == most_active).all()
```

[16]:
```
[(54.0, 85.0, 71.66378066378067)]
```

[17]:
```
# Filter by the station with the highest number of temperature observations.
# Query the last 12 months of temperature observation data for this station and␣
 ↪plot the results as a histogram (bins=12)
# year_ago_day = dt.date(2017,8,23)-dt.timedelta(days=365)
# year_ago_day
# session.query(Measurement.tobs).filter(Measurement.station == most_active).\
#     filter(Measurement.date>year_ago_day).limit(5).all()
row_df['date'] = pd.to_datetime(row_df['date'])
last_year_df = row_df.loc[row_df['date']>year_ago_day]
big_station_df = last_year_df.loc[last_year_df['station']==most_active]
hist = big_station_df['tobs'].hist(bins=12)
plt.xlabel("Temperature")
plt.ylabel("# of Observations")
plt.title("Observed Temperatures at Station USC00519281")
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-17-d7f057e3e440> in <module>
         6 #     filter(Measurement.date>year_ago_day).limit(5).all()
         7 row_df['date'] = pd.to_datetime(row_df['date'])
   ----> 8 last_year_df = row_df.loc[row_df['date']>year_ago_day]
         9 big_station_df = last_year_df.
↪loc[last_year_df['station']==most_active]
        10 hist = big_station_df['tobs'].hist(bins=12)


       NameError: name 'year_ago_day' is not defined
```

precipitation

[ ]:
```
# This function called `calc_temps` will accept start date and end date in the␣
 ↪format '%Y-%m-%d'
# and return the minimum, average, and maximum temperatures for that range of␣
 ↪dates
def calc_temps(start_date, end_date):
    """TMIN, TAVG, and TMAX for a list of dates.

    Args:
        start_date (string): A date string in the format %Y-%m-%d
        end_date (string): A date string in the format %Y-%m-%d

    Returns:
        TMIN, TAVE, and TMAX
    """

    return session.query(func.min(Measurement.tobs), func.avg(Measurement.
 ↪tobs), func.max(Measurement.tobs)).\
        filter(Measurement.date >= start_date).filter(Measurement.date <=␣
 ↪end_date).all()

# function usage example
print(calc_temps('2012-02-28', '2012-03-05'))
```

[ ]:
```
# Use your previous function `calc_temps` to calculate the tmin, tavg, and tmax
# for your trip using the previous year's data for those same dates.
print(calc_temps('2017-01-01', '2017-01-07'))
tmin, tave, tmax = calc_temps('2017-01-01', '2017-01-07')[0]
```

[ ]:
```
# Plot the results from your previous query as a bar chart.
# Use "Trip Avg Temp" as your Title
# Use the average temperature for the y value
# Use the peak-to-peak (tmax-tmin) value as the y error bar (yerr)
fig, ax = plt.subplots(figsize=plt.figaspect(2.))
xpos = 1
yerr = tmax - tmin
bar = ax.bar(xpos, tmax, yerr=yerr, alpha = 0.5, color = 'coral', align =␣
 ↪'center')
ax.set(ylabel = "Temp (F)", xticks = range(xpos), xticklabels = 'a',␣
 ↪title="Trip Average Temp")
ax.margins(.2,.2)
fig.tight_layout()
# ax.plot(row_df.index.values, row_df['prcp'],ls='-', marker='o')
# ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
# ax.xaxis.set_major_formatter(DateFormatter("%m-%d"))
fig.show()
```

```
# trip_dates_df = row_df.loc[row_df['date'].between('2016-12-23',␣
 ↪'2017-01-08')]
# plt.bar(trip_dates_df['date'],trip_dates_df['tobs'], color='r', alpha=0.5,␣
 ↪align="center")
# threshold = 70.14
# plt.axhline(y=threshold,linewidth=1, color='k')
```

[ ]:
```
# Calculate the total amount of rainfall per weather station for your trip␣
 ↪dates using the previous year's matching dates.
# Sort this in descending order by precipitation amount and list the station,␣
 ↪name, latitude, longitude, and elevation
sel = [Station.station,
       Station.name,
       Station.latitude,
       Station.longitude,
       Station.elevation,
       func.sum(Measurement.prcp)]

rainfall = session.query(*sel).filter(Station.station == Measurement.station).\
    filter(Measurement.date.between('2017-01-01','2017-01-08')).\
    group_by(Station.name).order_by(func.sum(Measurement.prcp).desc()).all()
rainfall
```

2.1 Optional Challenge Assignment

[ ]:
```
# Create a query that will calculate the daily normals
# (i.e. the averages for tmin, tmax, and tavg for all historic data matching a␣
 ↪specific month and day)

def daily_normals(date):
    """Daily Normals.

    Args:
        date (str): A date string in the format '%m-%d'

    Returns:
        A list of tuples containing the daily normals, tmin, tavg, and tmax

    """

    sel = [func.min(Measurement.tobs), func.avg(Measurement.tobs), func.
 ↪max(Measurement.tobs)]
    return session.query(*sel).filter(func.strftime("%m-%d", Measurement.date)␣
 ↪== date).all()
```

```
daily_normals("01-01")
```

[ ]:
```
# calculate the daily normals for your trip
# push each tuple of calculations into a list called `normals`

# Set the start and end date of the trip

# Use the start and end date to create a range of dates

# Stip off the year and save a list of %m-%d strings

# Loop through the list of %m-%d strings and calculate the normals for each␣
↪date
```

[ ]:
```
# Load the previous query results into a Pandas DataFrame and add the␣
↪`trip_dates` range as the `date` index
```

[ ]:
```
# Plot the daily normals as an area plot with `stacked=False`
```