**Surfs Up!**



Congratulations! You've decided to treat yourself to a long holiday vacation in Honolulu, Hawaii! To help with your trip planning, you decided to do some climate analysis on the area. Because you are such an awesome person, you have decided to share your ninja analytical skills with the community by providing a climate analysis api. The following outlines what you need to do.

**Step 1 - Data Engineering**

The climate data for Hawaii is provided through two CSV files. Start by using Python and Pandas to inspect the content of these files and clean the data.

\* Create a Jupyter Notebook file called `data\_engineering.ipynb` and use this to complete all of your Data Engineering tasks.

\* Use Pandas to read in the measurement and station CSV files as DataFrames.

\* Inspect the data for NaNs and missing values. You must decide what to do with this data.

\* Save your cleaned CSV files with the prefix `clean\_`.

**Step 2 - Database Engineering**

Use SQLAlchemy to model your table schemas and create a sqlite database for your tables. You will need one table for measurements and one for stations.

\* Create a Jupyter Notebook called `database\_engineering.ipynb` and use this to complete all of your Database Engineering work.

\* Use Pandas to read your cleaned measurements and stations CSV data.

\* Use the `engine` and connection string to create a database called `hawaii.sqlite`.

\* Use `declarative\_base` and create ORM classes for each table.

\* You will need a class for `Measurement` and for `Station`.

\* Make sure to define your primary keys.

\* Once you have your ORM classes defined, create the tables in the database using `create\_all`.

**Step 3 - Climate Analysis and Exploration**

You are now ready to use Python and SQLAlchemy to do basic climate analysis and data exploration on your new weather station tables. All of the following analysis should be completed using SQLAlchemy ORM queries, Pandas, and Matplotlib.

\* Create a Jupyter Notebook file called `climate\_analysis.ipynb` and use it to complete your climate analysis and data exporation.

\* Choose a start date and end date for your trip. Make sure that your vacation range is approximately 3-15 days total.

\* Use SQLAlchemy `create\_engine` to connect to your sqlite database.

\* Use SQLAlchemy `automap\_base()` to reflect your tables into classes and save a reference to those classes called `Station` and `Measurement`.

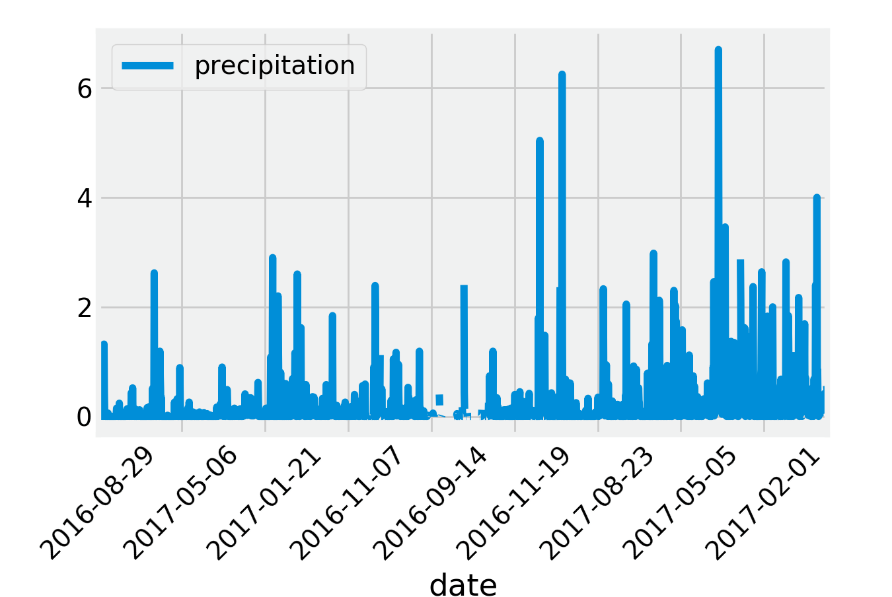
**Precipitation Analysis**

\* Design a query to retrieve the last 12 months of precipitation data.

\* Select only the `date` and `prcp` values.

\* Load the query results into a Pandas DataFrame and set the index to the date column.

\* Plot the results using the DataFrame `plot` method.



\* Use Pandas to print the summary statistics for the precipitation data.

**Station Analysis**

\* Design a query to calculate the total number of stations.

\* Design a query to find the most active stations.

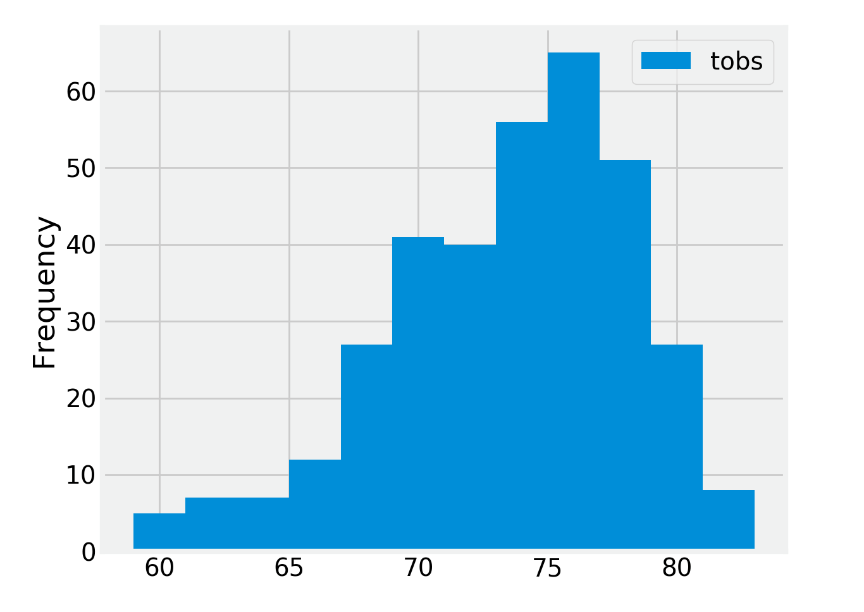
\* List the stations and observation counts in descending order

\* Which station has the highest number of observations?

\* Design a query to retrieve the last 12 months of temperature observation data (tobs).

\* Filter by the station with the highest number of observations.

\* Plot the results as a histogram with `bins=12`.



**Temperature Analysis**

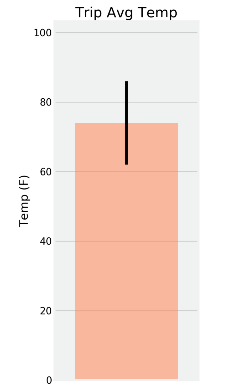
\* Write a function called `calc\_temps` that will accept a start date and end date in the format `%Y-%m-%d` and return the minimum, average, and maximum temperatures for that range of dates.

\* Use the `calc\_temps` function to calculate the min, avg, and max temperatures for your trip using the matching dates from the previous year (i.e. use "2017-01-01" if your trip start date was "2018-01-01")

\* Plot the min, avg, and max temperature from your previous query as a bar chart.

\* Use the average temperature as the bar height.

\* Use the peak-to-peak (tmax-tmin) value as the y error bar (yerr).



**Optional Recommended Analysis**

\* The following are optional challenge queries. These are highly recommended to attempt, but not required for the homework.

\* Calculate the rainfall per weather station using the previous year's matching dates.

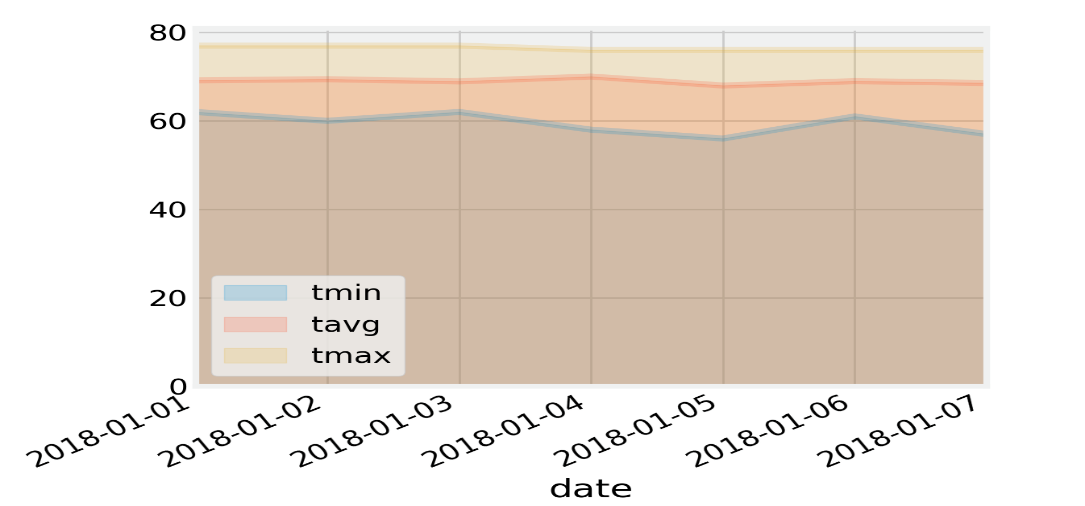
\* Calculate the daily normals. Normals are the averages for min, avg, and max temperatures.

\* Create a function called `daily\_normals` that will calculate the daily normals for a specific date. This date string will be in the format `%m-%d`. Be sure to use all historic tobs that match that date string.

\* Create a list of dates for your trip in the format `%m-%d`. Use the `daily\_normals` function to calculate the normals for each date string and append the results to a list.

\* Load the list of daily normals into a Pandas DataFrame and set the index equal to the date.

\* Use Pandas to plot an area plot (`stacked=False`) for the daily normals.



**Step 4 - Climate App**

Now that you have completed your initial analysis, design a Flask api based on the queries that you have just developed.

\* Use FLASK to create your routes.

**Routes**

\* `/api/v1.0/precipitation`

\* Query for the dates and temperature observations from the last year.

\* Convert the query results to a Dictionary using `date` as the key and `tobs` as the value.

\* Return the json representation of your dictionary.

\* `/api/v1.0/stations`

\* Return a json list of stations from the dataset.

\* `/api/v1.0/tobs`

\* Return a json list of Temperature Observations (tobs) for the previous year

\* `/api/v1.0/<start>` and `/api/v1.0/<start>/<end>`

\* Return a json list of the minimum temperature, the average temperature, and the max temperature for a given start or start-end range.

\* When given the start only, calculate `TMIN`, `TAVG`, and `TMAX` for all dates greater than and equal to the start date.

\* When given the start and the end date, calculate the `TMIN`, `TAVG`, and `TMAX` for dates between the start and end date inclusive.

**Hints**

\* You will need to join the station and measurement tables for some of the analysis queries.

\* Use Flask `jsonify` to convert your api data into a valid json response object.