A surfer riding a wave

Description automatically generated with medium confidence

**Hawaii Climate-Analysis**

The SQLAlchemy assignment required an analysis of Hawaii climate data from a local SQL database.

The database comprised two tables that included precipitation and temperature data from nine weather stations.

The assignment required the use of the following components to conduct the analysis:

* Jupyter Notebook
* SQLAlchemy
* Pandas
* Python
* Matplotlib
* Flask

Jupyter Notebook, SQLAlchemy, Pandas, Python, and Matplotlib were used to conduct the climate analysis and data exploration. Flask was then used to develop an API based on the results of the climate analysis.

Details of the assignment can be found at the following link:

<https://monash.bootcampcontent.com/monash-coding-bootcamp/monu-virt-data-11-2021-u-c/-/tree/master/02-Homework/10-Advanced-Data-Storage-and-Retrieval/Instructions>

The “10\_sqlalchemy\_challenge” directory contains the following files submitted for assessment:

* “10\_readme\_details\_mm” – a step by step, detailed description of the analyses and their results,
* “climate\_starter\_mm” – the jupyter notebook for the Climate Analysis,
* “analysis\_bonus\_1\_mm” - the jupyter notebook for the Analysis Bonus I,
* “analysis\_bonus\_2\_mm” - the jupyter notebook for the Analysis Bonus II,
* “flask\_climate\_starter\_mm” – the Flask Analysis App.

**Author: Mike Murphy**

## Step 1 - Climate Analysis and Exploration

**Requirement:**

Use SQLAlchemy ORM queries, Pandas, Python and Matplotlib to conduct the climate analysis and data exploration of database. Retrieve, analyse, and plot the preceding 12 months of precipitation data.

**Visualisation:**

Chart, histogram

Description automatically generated

## Step 2 - Station Analysis

**Requirement:**

Use SQLAlchemy ORM queries, Pandas, Python and Matplotlib to conduct the climate analysis and data exploration of database. Retrieve, analyse, and plot the preceding 12 months of precipitation data.

**Visualisation:**

## Chart, histogram Description automatically generated

## Step 3 – Flask Climate App

**Overall Flask Requirement:**

Use Flask to create the routes for the queries to the SQL database and list all the routes.

**Routes / Requirements:**

* **/api/v1.0/precipitation**
  + Query for the dates and temperature observations from the preceding year.
  + Convert the query results to a Dictionary using ‘date’ as the key and ‘prcp’ 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 for the most active station for the previous year.
* **/api/v1.0/<start>**
  + When given the start only, calculate TMIN, TAVG, and TMAX for all dates greater than and equal to the start date.
* **/api/v1.0/<start>/<end>**
  + When given the start and the end date, calculate the TMIN, TAVG, and TMAX for dates between the start and end date inclusive.

**Routes / Solutions:**

1. **@app.route("/api/v1.0/precipitation")**

def precipitation():

# Create our session (link) from Python to the DB

    session = Session(engine)

# Query precipitation

    sel = [Measurement.station, Measurement.date, Measurement.tobs]

    precipqry = session.query(\*sel).all()

    session.close()

    # Convert list of tuples into normal list

    precipitation = []

    for station, date, tobs in precipqry:

        dict\_precip = {}

        dict\_precip["Station"] = station

        dict\_precip["Date"] = date

        dict\_precip["Precipitation"] = tobs

        precipitation.append(dict\_precip)

    return jsonify(precipitation)

[

{

"Date": "2010-01-01",

"Precipitation": 65.0,

"Station": "USC00519397"

},

{

"Date": "2010-01-02",

"Precipitation": 63.0,

"Station": "USC00519397"

},

{

"Date": "2010-01-03",

"Precipitation": 74.0,

"Station": "USC00519397"

},

**.**

**.**

**.**

{

"Date": "2017-08-21",

"Precipitation": 76.0,

"Station": "USC00516128"

},

{

"Date": "2017-08-22",

"Precipitation": 76.0,

"Station": "USC00516128"

},

{

"Date": "2017-08-23",

"Precipitation": 76.0,

"Station": "USC00516128"

}

]

1. **@app.route("/api/v1.0/station")**

def station():

# Create our session (link) from Python to the DB

    session = Session(engine)

# Query station table

    sel = [Station.station, Station.name, Station.latitude, Station.longitude, Station.elevation]

    stationqry = session.query(\*sel).all()

    session.close()

# Convert list of tuples into normal list

    stations = []

    for station, name, lat, lon, elev in stationqry:

        dict\_stations = {}

        dict\_stations["Station"] = station

        dict\_stations["Name"] = name

        dict\_stations["Latitude"] = lat

        dict\_stations["Longitude"] = lon

        dict\_stations["Elevation"] = elev

        stations.append(dict\_stations)

    return jsonify(stations)

[

{

"Elevation": 3.0,

"Latitude": 21.2716,

"Longitude": -157.8168,

"Name": "WAIKIKI 717.2, HI US",

"Station": "USC00519397"

},

{

"Elevation": 14.6,

"Latitude": 21.4234,

"Longitude": -157.8015,

"Name": "KANEOHE 838.1, HI US",

"Station": "USC00513117"

},

{

"Elevation": 7.0,

"Latitude": 21.5213,

"Longitude": -157.8374,

"Name": "KUALOA RANCH HEADQUARTERS 886.9, HI US",

"Station": "USC00514830"

},

{

"Elevation": 11.9,

"Latitude": 21.3934,

"Longitude": -157.9751,

"Name": "PEARL CITY, HI US",

"Station": "USC00517948"

},

{

"Elevation": 306.6,

"Latitude": 21.4992,

"Longitude": -158.0111,

"Name": "UPPER WAHIAWA 874.3, HI US",

"Station": "USC00518838"

},

{

"Elevation": 19.5,

"Latitude": 21.33556,

"Longitude": -157.71139,

"Name": "WAIMANALO EXPERIMENTAL FARM, HI US",

"Station": "USC00519523"

},

{

"Elevation": 32.9,

"Latitude": 21.45167,

"Longitude": -157.84888999999998,

"Name": "WAIHEE 837.5, HI US",

"Station": "USC00519281"

},

{

"Elevation": 0.9,

"Latitude": 21.3152,

"Longitude": -157.9992,

"Name": "HONOLULU OBSERVATORY 702.2, HI US",

"Station": "USC00511918"

},

{

"Elevation": 152.4,

"Latitude": 21.3331,

"Longitude": -157.8025,

"Name": "MANOA LYON ARBO 785.2, HI US",

"Station": "USC00516128"

}

]

1. **@app.route("/api/v1.0/tobs")**

def tobs():

# Create our session (link) from Python to the DB

    session = Session(engine)

# find the most active station (i.e., what station has the most rows?

    session.query(Measurement.station, func.count(Measurement.date)).group\_by(Measurement.station).\

    order\_by(func.count(Measurement.date).desc()).all()

    most\_active = session.query(Measurement.station).first

# Retrieve and convert date, calculate 1-year prior date

    rec\_date = session.query(Measurement).order\_by(Measurement.date.desc()).first()

    most\_rec\_date = dt.datetime.strptime(rec\_date.date, '%Y-%m-%d')

    yr\_date = most\_rec\_date - dt.timedelta(days=365)

# Query 1 year's observations for the most active station

    for station in most\_active():

        sel = [Measurement.date, Measurement.tobs]

        yr\_qry = session.query(\*sel).filter(Measurement.date >= yr\_date).all()

    session.close()

# Convert list of tuples into normal list

    all\_tobs = []

    for date, tobs in yr\_qry:

        dict\_tobs = {}

        dict\_tobs["Station"] = station

        dict\_tobs["Date"] = date

        dict\_tobs["Tobs"] = tobs

        all\_tobs.append(dict\_tobs)

    return jsonify(all\_tobs)

[

{

"Date": "2016-08-24",

"Station": "USC00519397",

"Tobs": 79.0

},

{

"Date": "2016-08-25",

"Station": "USC00519397",

"Tobs": 80.0

},

{

"Date": "2016-08-26",

"Station": "USC00519397",

"Tobs": 79.0

},

**.**

**.**

**.**

{

"Date": "2017-08-21",

"Station": "USC00519397",

"Tobs": 76.0

},

{

"Date": "2017-08-22",

"Station": "USC00519397",

"Tobs": 76.0

},

{

"Date": "2017-08-23",

"Station": "USC00519397",

"Tobs": 76.0

}

]

1. **@app.route("/api/v1.0//<start>/<stop>")**

**NOTE: Used earliest and latest observation dates for this query and for verification of results.**

def get\_temp\_start\_stop(start, stop):

# Create our session (link) from Python to the DB

    session = Session(engine)

# Select the first and last dates of the data set for the range of dates

    start\_date = session.query(func.min(Measurement.date)).first()[0]

    end\_date = session.query(func.max(Measurement.date)).first()[0]

# Query min, avg, max observations

    mamqry = 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()

    session.close()

# Convert list of tuples into normal list

    all\_tobs = []

    for min, avg, max in mamqry:

        dict\_tobs = {}

        dict\_tobs["Min"] = min

        dict\_tobs["Avg"] = avg

        dict\_tobs["Max"] = max

        all\_tobs.append(dict\_tobs)

    return jsonify(all\_tobs)

[

{

"Avg": 73.09795396419437,

"Max": 87.0,

"Min": 53.0

}

]

1. **@app.route("/api/v1.0//<start>")**

**NOTE: Used earliest observation date for this query and for verification of results.**

[

{

"Avg": 73.09795396419437,

"Max": 87.0,

"Min": 53.0

}

]

## Step 3 – Bonus: Other Recommended Analysis - Temperature Analysis I

**Requirement:**

Use Pandas to determine if there is a meaningful difference in temperature between the months of June and December.

**Result:**

The unpaired t-test was used to compare the difference in temperature for June and December.

The unpaired t-test is used to compare the means of two samples of data when each individual in one sample is independent of every individual in the other sample.

As opposed to the paired t-test which is used to compare the means of two samples when each individual in one sample also appears in the other sample.

**The results of the unpaired t-test indicate the following:**

* The t-value of 0.0 indicates that the two samples are statistically very similar.
* The p-value of 1.0 indicates that there is no difference statistically between the two samples.

## Step 4 – Bonus: Other Recommended Analysis - Temperature Analysis II

**Overall Requirement:**

Use Pandas to determine if the weather for the period August 1-7, 2017, is likely to be conducive for a vacation based on analysis of historical data.

**Requirement 1: Temperature Analysis**

Use the `calc\_temps` function to calculate the min, avg, and max temperatures for your trip using the matching dates from a previous year (i.e., use "2017-08-01").

**Visualisation:**

## Chart, box and whisker chart Description automatically generated

**Requirement 2: Daily Rainfall Average**

Calculate the rainfall per weather station using the previous year's matching dates and sort into descending sequence.

**Result:**

**Station Name Latitude Longitude Elevation PrcpSum**

0 USC00516128 MANOA LYON ARBO 785.2, HI US 21.33310 -157.80250 152.4 0.36

1 USC00514830 KUALOA RANCH HEADQUARTERS 886.9, HI US

21.52130 -157.83740 7.0 0.22

2 USC00519281 WAIHEE 837.5, HI US 21.45167 -157.84889 32.9 0.06

3 USC00519397 WAIKIKI 717.2, HI US 21.27160 -157.81680 3.0 0.04

4 USC00519523 WAIMANALO EXPERIMENTAL FARM, HI US

21.33556 -157.71139 19.5 0.00

**Requirement 3: Daily Temperature Normals**

Calculate the daily normals for the duration of your trip. Normals are the averages for the min, avg, and max temperatures. You are provided with 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.

**Visualisation:**

Chart

Description automatically generated