In [1]:

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

In [2]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
import datetime as dt
```

# Reflect Tables into SQLAlchemy ORM

In [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, inspect
```

In [5]:

```
engine = create_engine("sqlite:///hawaii.sqlite")
```

In [6]:

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

In [7]:

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

Out[7]:

```
['measurement', 'station']
```

In [8]:

```
# Save references to each table
Measurement = Base.classes.measurement
Station = Base.classes.station
```

In [9]:

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

# Exploratory Climate Analysis

In [ ]:

```
# 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

# Perform a query to retrieve the data and precipitation scores

# Save the query results as a Pandas DataFrame and set the index to the date column

# Sort the dataframe by date

# Use Pandas Plotting with Matplotlib to plot the data
```

In [10]:

```
#First of all, it is needed the max date
max_date = session.query(func.max(func.strftime("%Y-%m-%d", Measurement.date))).limit(5).all()
max_date[0][0]
```

Out[10]:

```
'2017-08-23'
```

In [11]:

```
precipitation_data = session.query(func.strftime("%Y-%m-%d", Measurement.date),Measurement.prcp).\
    filter(func.strftime("%Y-%m-$d",Measurement.date) >= dt.date(2016, 8, 23)).all()
precipitation_df = pd.DataFrame(precipitation_data, columns = ["date", "precipitation"])
precipitation_df.set_index("date", inplace = True)
```

In [12]:

```
#Now, sorting the dataframe by date
precipitation_df = precipitation_df.sort_values(by = "date")
precipitation_df.head()
```

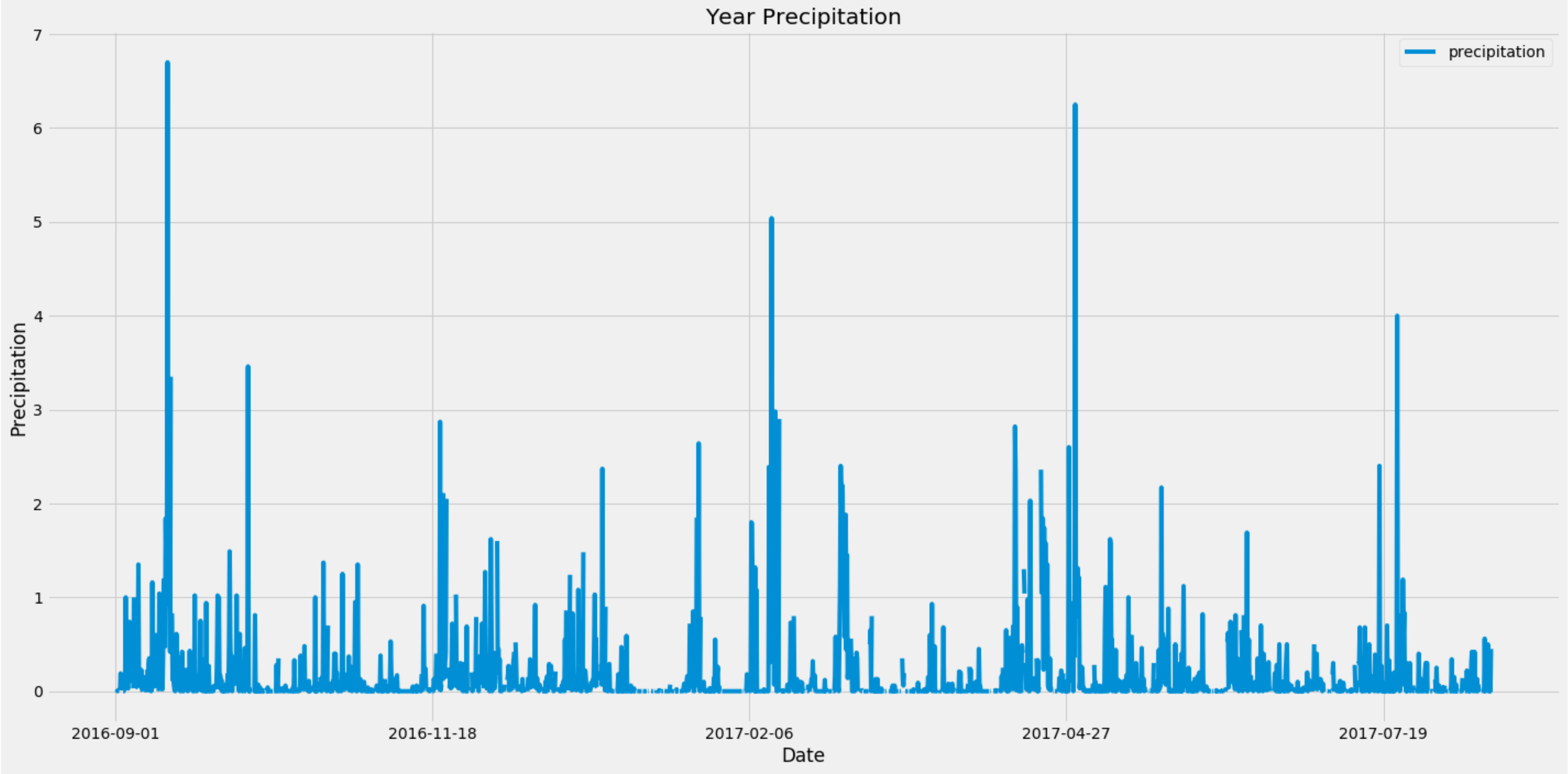
Out[12]:

| | precipitation |
|---|---|
| **date** | |
| **2016-09-01** | 0.00 |
| **2016-09-01** | NaN |
| **2016-09-01** | 0.02 |
| **2016-09-01** | 0.00 |
| **2016-09-01** | 0.01 |

In [13]:

```
# And now plotting
fig, ax = plt.subplots(figsize = (20,10))
precipitation_df.plot(ax = ax, x_compat = True)
ax.set_xlabel("Date")
ax.set_ylabel("Precipitation")
ax.set_title("Year Precipitation")

plt.tight_layout()
plt.show()
```



In [14]:

```
# Use Pandas to calcualte the summary statistics for the precipitation data
precipitation_df.describe()
```

Out[14]:

| | precipitation |
|---|---|
| **count** | 1968.000000 |
| **mean** | 0.171479 |
| **std** | 0.451817 |
| **min** | 0.000000 |
| **25%** | 0.000000 |
| **50%** | 0.020000 |
| **75%** | 0.130000 |
| **max** | 6.700000 |

In [15]:

```
# Design a query to show how many stations are available in this dataset?
stations = session.query(Station.id).distinct().count()
stations
```

Out[15]:

```
9
```

In [16]:

```
# What are the most active stations? (i.e. what stations have the most rows)?
# List the stations and the counts in descending order.
station_counts = (session.query(Measurement.station, func.count(Measurement.station))
                  .group_by(Measurement.station)
                  .order_by(func.count(Measurement.station).desc())
                  .all())
station_counts
```

Out[16]:

```
[('USC00519281', 2772),
 ('USC00519397', 2724),
 ('USC00513117', 2709),
 ('USC00519523', 2669),
 ('USC00516128', 2612),
 ('USC00514830', 2202),
 ('USC00511918', 1979),
 ('USC00517948', 1372),
 ('USC00518838', 511)]
```

In [17]:

```
# Using the station id from the previous query, calculate the lowest temperature recorded,
# highest temperature recorded, and average temperature of the most active station?
mostActiveStation = "USC00519281"
temps = session.query(func.min(Measurement.tobs), func.max(Measurement.tobs), func.avg(Measurement.tobs)).\
    filter(Measurement.station == mostActiveStation).all()
temps
```

Out[17]:

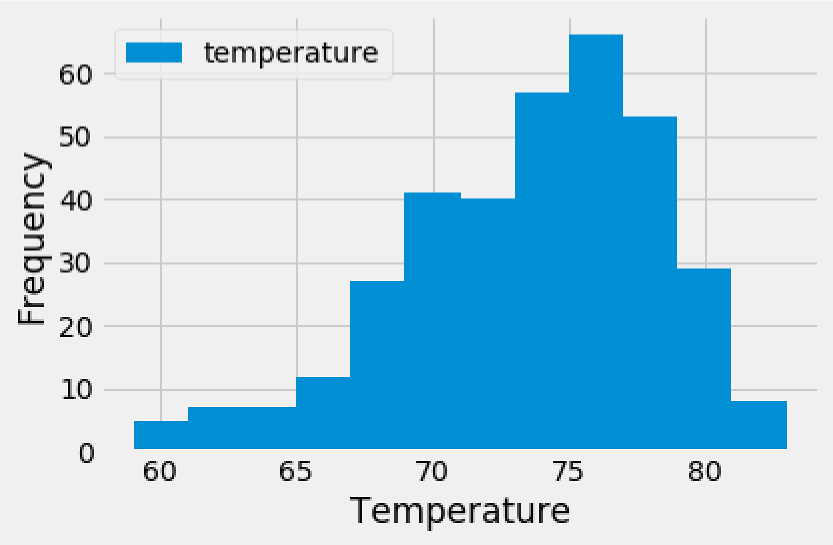
```
[(54.0, 85.0, 71.66378066378067)]
```

In [22]:

```
# Choose 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
tempObservation = session.query(Measurement.date, Measurement.tobs).filter(Measurement.station == mostActiveStation).\
    filter(func.strftime("%Y-%m-%d", Measurement.date) >= dt.date(2016, 8, 23)).all()
tempObservation_df = pd.DataFrame(tempObservation, columns = ["date", "temperature"])

#Now plotting
fig, ax = plt.subplots()
tempObservation_df.plot.hist(bins = 12, ax = ax)
ax.set_xlabel("Temperature")
ax.set_ylabel("Frequency")

plt.tight_layout()
plt.show()
```



## Bonus Challenge Assignment

In [ ]:

```
# 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'))
```

In [ ]:

```
# 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.
```

In [ ]:

```
# 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)
```

In [ ]:

```
# Calculate the total amount of rainfall per weather station for your trip dates using the previous year's matching date
s.
# Sort this in descending order by precipitation amount and list the station, name, latitude, longitude, and elevation
```

In [ ]:

```
# 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")
```

In [ ]:

```
# 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
```

In [ ]:

```
# Load the previous query results into a Pandas DataFrame and add the `trip_dates` range as the `date` index
```

In [ ]:

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