Weather Analysis

01-01-2008 to 31-12-2017

AIRPORT CODE	STATION NAME	STATION CODE
BUF	BUFFALO NIAGARA INTERNATIONAL NY US	USW00014733
DEL	NEW DELHI PALAM IN	IN022023000
CCU	CALCUTTA DUM DUM IN	IN024141500

1. BUF Quarterly Mean Weather Variation over 10 years
2. DEL Quarterly Mean Weather Variation over 10 years
3. CCU Quarterly Mean Weather Variation over 10 years
4. BUF Annual Precipitation Variation over 10 years

Weather Data Source

The data that has been analysed have been cited from http://ncdc.noaa.gov/cdo-web/datatools vide order number 1263442. The data downloaded was in the form of a CSV containing weather records for a period of 10 years for 3 stations: USW00014733 (BUF), IN022023000 (DEL), IN024141500 (CCU).

The example format for the downloaded CSV file have been cited underneath.

```
"STATION","NAME","LATITUDE","LONGITUDE","ELEVATION","DATE","AWND","PRCP",
"SNOW","SNWD","TAVG","TMAX","TMIN","WT01","WT02","WT03","WT04","WT05","WT
06","WT07","WT08","WT09","WT11","WT13","WT14","WT15","WT16","WT17","WT18"
,"WT21","WT22"
"USW00014733","BUFFALO NIAGARA INTERNATIONAL NY
US","42.9408","-78.7358","218.2","2008-01-01","5.1","8.9","58.0","25.0",,
"1.7","-5.0","    1","    1",,,,,,,,,,,,,"    1",,"    1",,
"USW00014733","BUFFALO NIAGARA INTERNATIONAL NY
US","42.9408","-78.7358","218.2","2008-01-02","5.6","0.0","5.0","76.0",,"
-5.0","-10.6","    1",,,,,,,,,,"    1",,,,,"    1",,
```

Weather Analysis

The quarterly mean temperature of an area is calculated over a 3 months period of time and is the mean of the highest recorded temperature and the lowest recorded temperature for the given period of time. In the metric system temperature is expressed in °C (degree celsius).

The annual precipitation of an area is calculated over a time interval of 1 year and is the total sum of the daily recorded count of precipitation received. In the metric system precipitation is expressed in mm (millimetres).

Daily records of weather data for a particular station can be used to calculate the quarterly mean temperature and annual precipitation of an area.

Source Code

Python program to create a JSON file of quarterly mean temperatures of multiple stations from aforesaid data source.

```
#!/usr/bin/env python3

# defining values

filePath = './1263442.csv'
stations = {
    'USW00014733': 'BUFFALO NIAGARA INTERNATIONAL NY US',
    'IN022023000': 'NEW DELHI PALAM IN',
    'IN024141500': 'CALCUTTA DUM DUM IN'
}
outFilePath = './wa_quarterly_temp.json'

# parse csv

def csvLineToList(csvLine):
    dataList = csvLine.strip('\n').split(',')
    # return list of data items stripping the quotes
    return list(map(lambda data: data.strip('"').strip(' '), dataList))

def listToDict(values, keys):
    # return a dictionary looping the key and value lists
    return {keys[index].lower(): values[index] for index in range(len(values))}

csvFile = open(filePath)

labels = csvLineToList(csvFile.readline())

csvFile.readline()

# create mean periods and respective months per quarter year

meanPeriods = ['{}Q{}'.format(year, quarter) for year in range(2008, 2018) for
quarter in range(1, 5)]

months = {}
for meanPeriod in meanPeriods:
    (year, quarter) = meanPeriod.split('Q')
    quarter = int(quarter)
    months[meanPeriod] = ['%s-%02d' % (year, month) for month in range((quarter
- 1) * 3 + 1, quarter * 3 + 1)]

# record for each station
quarterlyCityTemperatures = {}

for stationCode in stations:
    quarterlyCityTemperatures[stationCode] = {}
    weatherRecords = []

    for line in iter(lambda: csvFile.readline(), ''):
        weatherData = listToDict(csvLineToList(line), labels)
        if weatherData['station'] != stationCode:
            break
        weatherRecords.append(weatherData)

    for meanPeriod in meanPeriods:
        currentMonths = months[meanPeriod]
        quarterlyRecords = []
        for month in currentMonths:
            quarterlyRecords = quarterlyRecords + list(filter(lambda record:
record['date'].startswith(month), weatherRecords))

        tempMax = -9999.99
        tempMin = 1000.00
        for record in quarterlyRecords:
            try:
                if float(record['tmax']) > tempMax:
                    tempMax = float(record['tmax'])
            except ValueError:
                pass
            try:
                if float(record['tavg']) > tempMax:
                    tempMax = float(record['tavg'])
            except ValueError:
                pass
            try:
                if float(record['tmin']) < tempMin:
                    tempMin = float(record['tmin'])
            except ValueError:
                pass
            try:
                if float(record['tavg']) < tempMin:
                    tempMin = float(record['tavg'])
            except ValueError:
                pass

        tempMean = (tempMax + tempMin) / 2
        quarterlyCityTemperatures[stationCode][meanPeriod] = tempMean

import json

quarterlyCityWiseMeanTemperatures = [{city: quarterlyCityTemperatures[city]} for
city in quarterlyCityTemperatures]
print(json.dumps(quarterlyCityWiseMeanTemperatures))

with open(outFilePath, 'w') as outFile:
    print(json.dumps(quarterlyCityWiseMeanTemperatures), end = '',  file =
outFile)
```

Python program to create a JSON file of annual precipitation of a single station from aforesaid data source.

```
#!/usr/bin/env python3

filePath = './old_1261071.csv'
outFilePath = './wa_annual_prcp_buf.json'

def csvLineToList(csvLine):
    dataList = csvLine.strip('\n').split(',')
    # return list of data items stripping the quotes
    return list(map(lambda data: data.strip('"').strip(' '), dataList))

def listToDict(values, keys):
    return {keys[index].lower(): values[index] for index in range(len(values))}

csvFile = open(filePath)

labels = csvLineToList(csvFile.readline())
```

```python
pptData = []
for line in iter(lambda: csvFile.readline(), ''):
    dataDict = listToDict(csvLineToList(line), labels)
    try:
        pptData.append({ 'station': dataDict['station'], 'date':
dataDict['date'], 'prcp': dataDict['prcp'] })
    except:
        pass

stationCode = pptData[0]['station']

pptData = filter(lambda record: record['station'] == stationCode, pptData)
pptData = {stationCode: list(map(lambda record: { 'date': record['date'],
'prcp': record['prcp'] }, pptData))}

yearlyPptData = {}

for year in range(2008, 2018):
    yearlyData = list(filter(lambda record:
record['date'].startswith(str(year)), pptData[stationCode]))
    prcpSum = 0
    for data in yearlyData:
        try:
            prcpSum = prcpSum + float(data['prcp'])
        except ValueError:
            pass
    yearlyPptData[str(year)] = prcpSum

import json

print(json.dumps({stationCode: yearlyPptData}))

with open(outFilePath, 'w') as outFile:
    print(json.dumps({stationCode: yearlyPptData}), end = '',  file = outFile)
```

Swift program to implement plotting of graphs natively inside UIViews.

```swift
//: A UIKit based Playground for presenting user interface

//
//  GraphView.swift
//
//  Created by SwG Ghosh on 25/04/17.
//  Copyright © 2017-18. All rights reserved.
//

//  Last modified on 29/03/18.


import UIKit
import CoreGraphics

@IBDesignable
class GraphView: UIView {

    // IB defines the colors to be used for various elements in the line graph
    @IBInspectable var graphAxisColor: UIColor = .black
    @IBInspectable var graphPointColor: UIColor = .blue
    @IBInspectable var graphLineColor: UIColor = .blue
    @IBInspectable var graphLegendPointColor: UIColor = .gray
    @IBInspectable var graphLegendLabelColor: UIColor = .gray
    @IBInspectable var graphLegendLineColor: UIColor = .gray

    // sample y values whose graph is to be plotted
    var currents: [Float] = [Float]()
    // lowest and highest values in the actual graph used for calibrating the
scale in the graph
    var lvalue = 30
    var hvalue = 50

    // sample x values in the form of
    var labels = [String]()

    // Only override draw() if you perform custom drawing.
    // An empty implementation adversely affects performance during animation.
    override func draw(_ rect: CGRect) {
        // Drawing code

        // defines an offset to place the axes of the graph with respect to rect
        let axisOffset: CGFloat = 20.0
        // defines the point of origin of graph in terms of rect position
        let origin: CGPoint = CGPoint(x: axisOffset, y: rect.height -
axisOffset)
        // defines an offset for allowing a small border and to ensure plotted
points do not get out of screen
        let borderOffset: CGFloat = 10.0
        // defines the radius of each point that is to be drawn
        let pointRadius: CGFloat = 5.0

        // draws the x and y axis of the graph with the specified offset
        drawAxes(axisOffset: axisOffset)

        // draws the scale for the graph to facilitate ease of understanding
values
        drawScale(originPoint: origin, borderOffset: borderOffset, pointRadius:
pointRadius, yLowest: lvalue, yHighest: hvalue, numberOfValues: currents.count -
1)

        // plots a graph based on the integral values provide and highest and
lowest values
        plotGraph(originPoint: origin, borderOffset: borderOffset, pointRadius:
pointRadius, yValues: currents, yLowest: lvalue, yHighest: hvalue)
    }

    func drawAxes(axisOffset: CGFloat) {
        // refers to current rect
        let rect = self.bounds

        // draw the axes for the graph

        // draws the x axis line for the graph
        let xAxisLine = UIBezierPath()
        xAxisLine.move(to: CGPoint(x: axisOffset, y: rect.height - axisOffset))
        xAxisLine.addLine(to: CGPoint(x: rect.width, y: rect.height -
axisOffset))
        xAxisLine.close()

        // draws the y axis line for the graph
        let yAxisLine = UIBezierPath()
        yAxisLine.move(to: CGPoint(x: axisOffset, y: 0.0))
        yAxisLine.addLine(to: CGPoint(x: axisOffset, y: rect.height -
axisOffset))
        yAxisLine.close()

        // uses the color specified in IB to stroke the axes lines
        graphAxisColor.setStroke()
        xAxisLine.stroke()
        yAxisLine.stroke()
    }

    func drawScale(originPoint: CGPoint, borderOffset: CGFloat, pointRadius:
CGFloat, yLowest lvalue: Int, yHighest hvalue: Int, numberOfValues: Int) {
        // refers to the current rect
        let rect = self.bounds

        // defines a constant float value to help in drawing points on screen
with rects
        let originOffsetForPoint: CGFloat = pointRadius / 2;

        // sets the colors for the line graph that is to be plotted based on
colours specified in IB
        graphLegendPointColor.setFill()
        // sets the color for the legend line that is specified in IB
        graphLegendLineColor.setStroke()

        // defines the x, y positions on rect of the origin of the graph
        let xOrigin: CGFloat = originPoint.x
        let yOrigin: CGFloat = originPoint.y

        // defines the difference in x, y from highest postion of graph to
lowest position on graph along x, y respectively as per points on the rect
        let xDiff: CGFloat = (rect.width - xOrigin - borderOffset)
        let yDiff: CGFloat = (yOrigin - 0.0 - borderOffset)

        // defines the difference in highest and lowest of the actual graph
values
        let valDiff = CGFloat(hvalue - lvalue)
        // defines a scaling calibration value useful for plotting of y points
on graph, differenceInActualValuesOfGraph/differenceInYLimitOfRect
        let yDiffScale = valDiff / yDiff

        // defines a scaling calibration value useful for plotting of x points
on graph, provided that x incremennts by one for each consecutive value of y
        let xDiffScale = (xDiff / CGFloat(numberOfValues))

        // loop to draw the legend points and legend text labels along y axis
        var i = 0
        while(i <= 10) {
            // actual rect point for the legend graph point that is to be
plotted on the graph
            let point = CGPoint(x: xOrigin, y: yOrigin - (CGFloat(hvalue -
lvalue) * CGFloat(i) / 10.0) / yDiffScale)
            // defining a rect with an origin offset at that point
            let pointRect = CGRect(x: point.x - originOffsetForPoint, y: point.y
- originOffsetForPoint, width: pointRadius, height: pointRadius)
            // drawing a oval/circle in that rect to represent a point on the
actual graph
            let pointPath = UIBezierPath(ovalIn: pointRect)
            // fill the oval with color
            pointPath.fill()

            // draw labels for legend texts
            let label = UILabel(frame: CGRect(x: xOrigin - xOrigin, y: point.y -
(yDiff / 11.0) + (yDiff / 11.0 / 4.0) , width: originPoint.x - 2.0, height:
yDiff / 11.0))
            // make font in label to adjust to fit width
            label.adjustsFontSizeToFitWidth = true
            // set legend text color as specified in IB
            label.textColor = graphLegendLabelColor
            label.backgroundColor = .clear
            label.textAlignment = .center
            // assign appropriate valua as text to the legend label
            label.text = "\(((hvalue - lvalue) * i / 10) + lvalue)"

            // add the label to the current view
            self.addSubview(label)

            if (i != 0) {
                // draw a line for graph legend
                let line = UIBezierPath()
                // point upto which line is to be drawn
                let toPoint = CGPoint(x: rect.width, y: point.y)
                // line to start from current defined point
                line.move(to: point)
                // line to end at
                line.addLine(to: toPoint)
                line.close()

                // line is stroked with color set
                line.stroke()
            }

            // increment loop counter
            i = i + 1
        }

        // loop to draw the legend points and legend text labels along x axis
        i = 0
        while(i <= numberOfValues) {
            // actual rect point for the legend graph point that is to be
plotted on the graph
            let point = CGPoint(x: xOrigin + (CGFloat(i) * xDiffScale), y:
yOrigin)
            // defining a rect with an origin offset at that point
            let pointRect = CGRect(x: point.x - originOffsetForPoint, y: point.y
- originOffsetForPoint, width: pointRadius, height: pointRadius)
            // drawing a oval/circle in that rect to represent a point on the
actual graph
            let pointPath = UIBezierPath(ovalIn: pointRect)
            // fill the oval with color
            pointPath.fill()

            // draw labels for legend texts
            let label = UILabel(frame: CGRect(x: point.x - (xDiffScale / 2.0),
y: originPoint.y , width: xDiffScale, height: rect.height - originPoint.y))
            // set font in label
            label.font = .systemFont(ofSize: 8.0)
            // set legend text color as specified in IB
            label.textColor = graphLegendLabelColor
            label.backgroundColor = .clear
            label.textAlignment = .center
            // assign appropriate value as text to the legend label
            label.text = "\(labels[i])"

            // add the label to the current view
            self.addSubview(label)

            if(i != 0) {
                // draw a line for graph legend
                let line = UIBezierPath()
                // point upto which line is to be drawn
                let toPoint = CGPoint(x: point.x, y: 0.0)
                // line to start from current defined point
                line.move(to: point)
                // line to end at
                line.addLine(to: toPoint)
                line.close()
                // line is stroked with color set
                line.stroke()
            }

            // increment loop counter
            i = i + 1
        }
    }

    func plotGraph(originPoint: CGPoint, borderOffset: CGFloat, pointRadius: 
CGFloat, yValues values: [Float], yLowest lvalue: Int, yHighest hvalue: Int) {
        // refers to the current rect
        let rect = self.bounds

        // defines a constant float value to help in drawing points on screen 
with rects
        let originOffsetForPoint: CGFloat = pointRadius / 2;

        // sets the colors for the line graph that is to be plotted based on 
colours specified in IB
        graphPointColor.setFill()
        graphLineColor.setStroke()

        // defines the x, y positions on rect of the origin of the graph
        let xOrigin: CGFloat = originPoint.x
        let yOrigin: CGFloat = originPoint.y

        // defines the difference in x, y from highest postion of graph to 
lowest position on graph along x, y respectively as per points on the rect
        let xDiff: CGFloat = (rect.width - xOrigin - borderOffset)
        let yDiff: CGFloat = (yOrigin - 0.0 - borderOffset)

        // defines the difference in highest and lowest of the actual graph 
values
        let valDiff = CGFloat(hvalue - lvalue)
        // defines a scaling calibration value useful for plotting of y points 
on graph, differenceInActualValuesOfGraph/differenceInYLimitOfRect
        let yDiffScale = valDiff / yDiff

        // defines a scaling calibration value useful for plotting of x points 
on graph, provided that x incremennts by one for each consecutive value of y
        let xDiffScale = (xDiff / CGFloat(values.count - 1))

        // counter to run a loop to plot points and lines on the graph for each 
of the values
        var i = 0
        while(i < values.count) {
            // actual rect point for the graph point that is to be plotted on 
the graph
            let point = CGPoint(x: xOrigin + (CGFloat(i) * xDiffScale), y: 
yOrigin - ((CGFloat(values[i]) - CGFloat(lvalue)) / yDiffScale))
            // defining a rect with an origin offset at that point
            let pointRect = CGRect(x: point.x - originOffsetForPoint, y: point.y 
- originOffsetForPoint, width: pointRadius, height: pointRadius)
            // drawing a oval/circle in that rect to represent a point on the 
actual graph
            let pointPath = UIBezierPath(ovalIn: pointRect)
            // fill the oval with color
            pointPath.fill()

            // draw a line between this point and the next point on the graph 
that is to plotted on next iteration of the loop
            if(i + 1 != values.count) {
                // bezier path to draw the line
                let line = UIBezierPath()
                // point to
                let nextPoint = CGPoint(x: xOrigin + (CGFloat(i + 1) *
xDiffScale), y: yOrigin - ((CGFloat(values[i + 1]) - CGFloat(lvalue)) /
yDiffScale))

                // line to start from current defined point
                line.move(to: point)
                // line to be drawn upto the next defined point
                line.addLine(to: nextPoint)
                line.close()

                // stroke the line with color
                line.stroke()
            }

            // increment loop counter
            i = i + 1
        }

    }

}
```

Swift program to parse JSON from file for annual precipitation (single station) and quarterly temperature (multiple stations).

```
import Foundation
import PlaygroundSupport

// fetch json(s) of data values
var prcpJson = try String(contentsOf:
playgroundSharedDataDirectory.appendingPathComponent("wa_annual_prcp_buf.json"))
var prcpJsonData = prcpJson.data(using: .utf8)

var tempJson = try String(contentsOf:
playgroundSharedDataDirectory.appendingPathComponent("wa_quarterly_temp.json"))
var tempJsonData = tempJson.data(using: .utf8)

// serialising the json(s) into an object
var parsedPrcpJson = try JSONSerialization.jsonObject(with: prcpJsonData!,
options: JSONSerialization.ReadingOptions()) as! [String: [String: Float]]
var parsedTempJson = try JSONSerialization.jsonObject(with: tempJsonData!,
options: JSONSerialization.ReadingOptions()) as! [[String: [String: Float]]]

// Precipitation
var pptXValues = [String]()
var pptYValues = [Float]()
for (_, dataDict) in parsedPrcpJson {
    for record in dataDict.sorted(by: <) {
        pptXValues.append(record.key)
        pptYValues.append(record.value)
    }
}

// Temperature
var allTempXValues = [[String]]()
var allTempYValues = [[Float]]()

for item in parsedTempJson {
    for (_, dataDict) in item {
        var xValues = [String]()
```

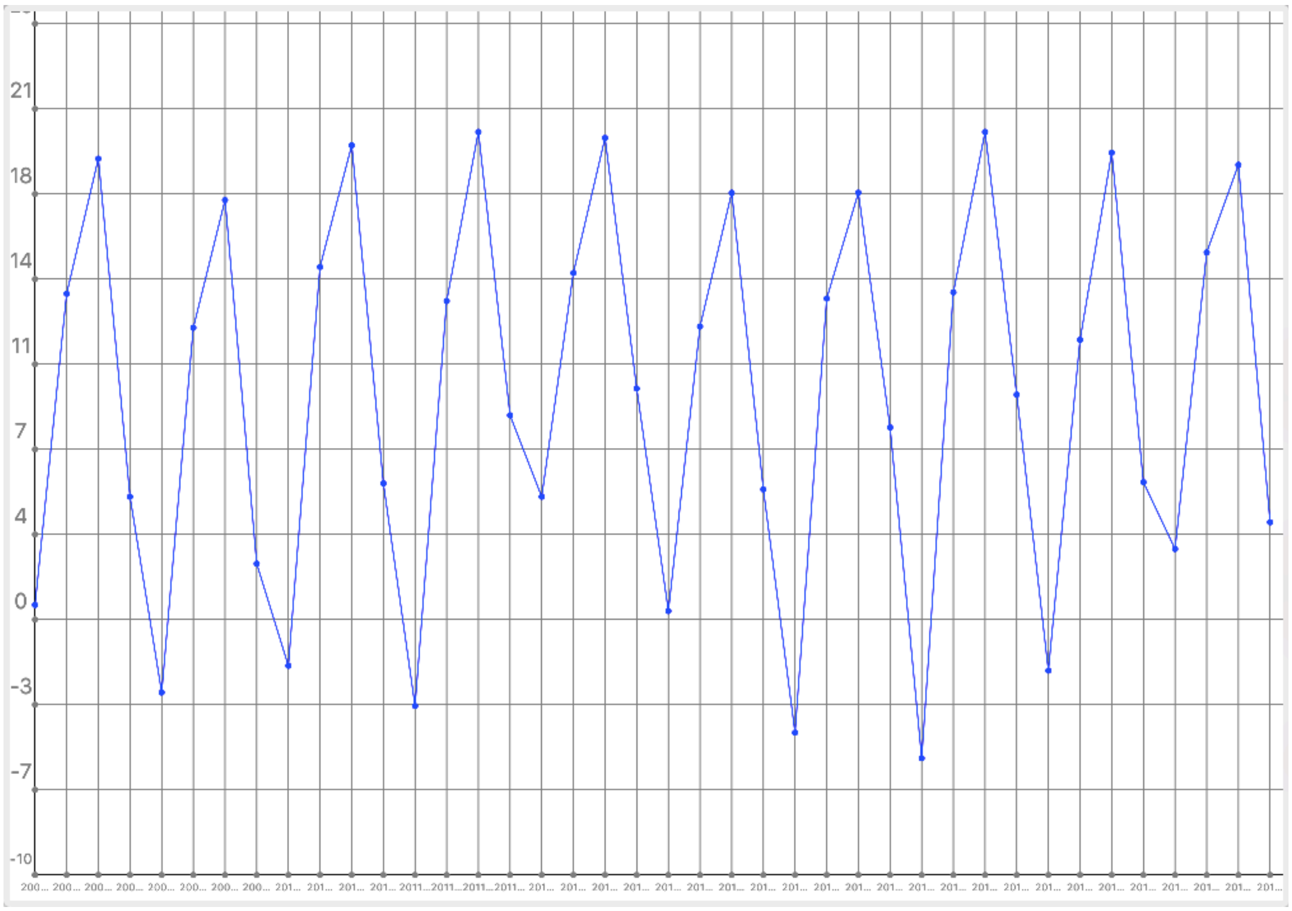
```swift
        var yValues = [Float]()

        for record in dataDict.sorted(by: <) {
            xValues.append(record.key)
            yValues.append(record.value)
        }

        allTempXValues.append(xValues)
        allTempYValues.append(yValues)
    }
}
```

Swift program to generate the required graphs showing variations as aforementioned.

```swift
// BUF Quarterly Mean Temperature
let graph1 = GraphView(frame: CGRect(x: 0, y: 0, width: 1000, height: 700))
graph1.currents = allTempYValues[0]
graph1.labels = allTempXValues[0]
graph1.lvalue = -10
graph1.hvalue = 50
graph1.backgroundColor = .white
graph1.setNeedsDisplay()
graph1

// DEL Quarterly Mean Temperature
let graph2 = GraphView(frame: CGRect(x: 0, y: 0, width: 1000, height: 700))
graph2.currents = allTempYValues[1]
graph2.labels = allTempXValues[1]
graph2.lvalue = 0
graph2.hvalue = 50
graph2.backgroundColor = .white
graph2.setNeedsDisplay()
graph2

// CCU Quarterly Mean Temperature
let graph3 = GraphView(frame: CGRect(x: 0, y: 0, width: 1000, height: 700))
graph3.currents = allTempYValues[2]
graph3.labels = allTempXValues[2]
graph3.lvalue = 0
graph3.hvalue = 50
graph3.backgroundColor = .white
graph3.setNeedsDisplay()
graph3

// BUF Annual Precipitation
let graph4 = GraphView(frame: CGRect(x: 0, y: 0, width: 1000, height: 700))
graph4.currents = pptYValues
graph4.labels = pptXValues
graph4.lvalue = 30
graph4.hvalue = 50
graph4.backgroundColor = .white
graph4.setNeedsDisplay()
graph4
```

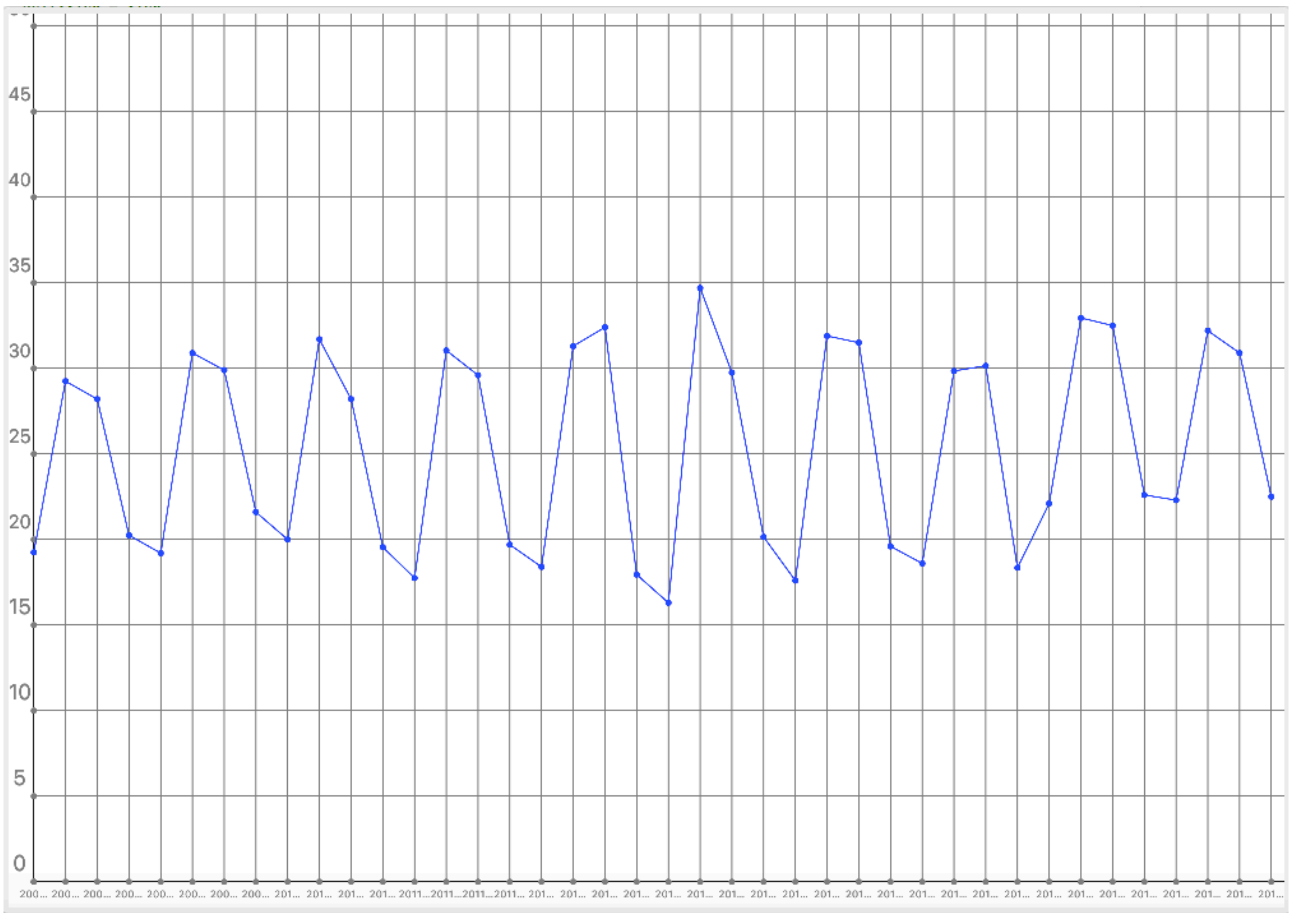
Graphical Results

BUF Quarterly Mean Weather Variation over 10 years



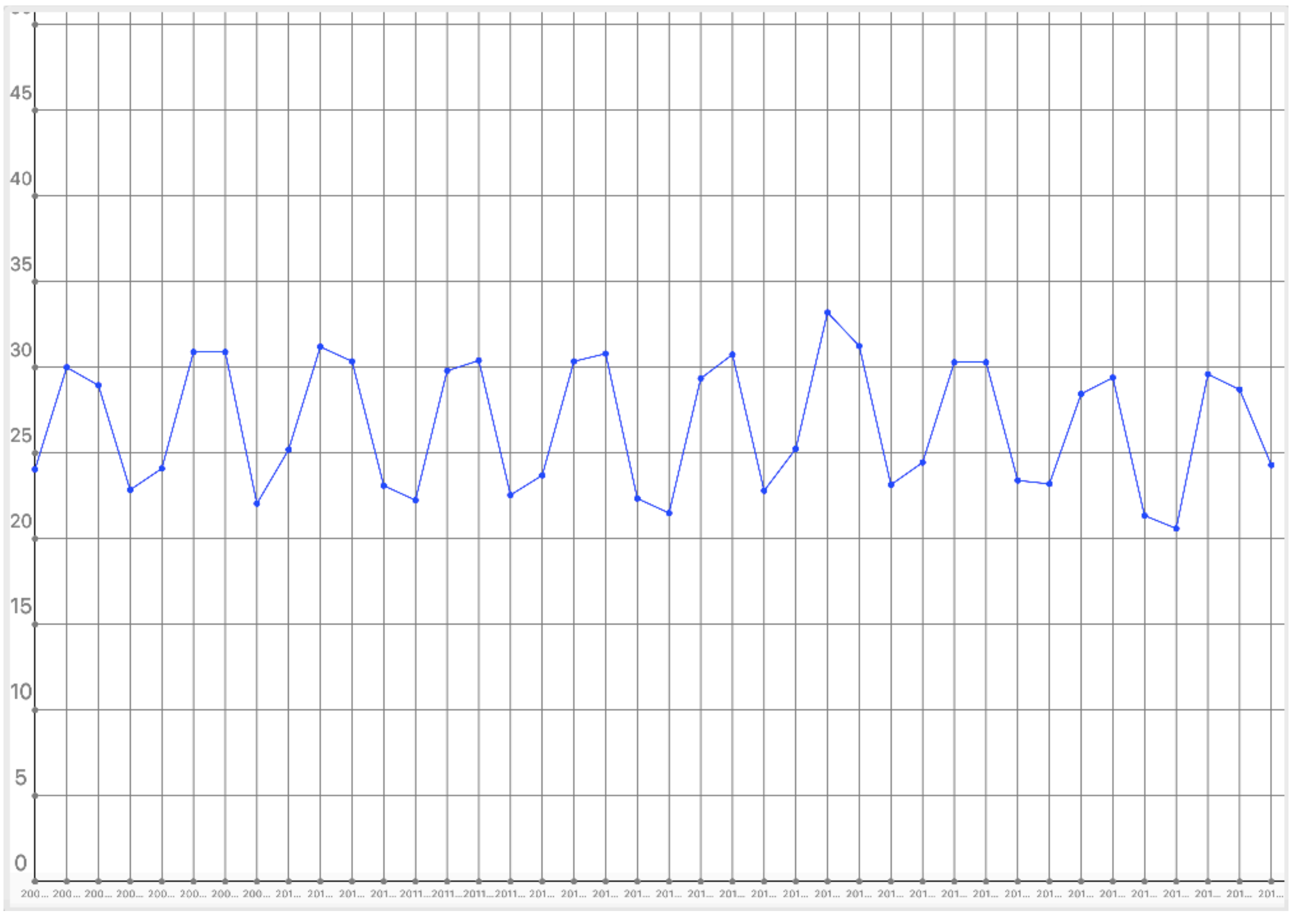
Station: BUFFALO NIAGARA INTERNATIONAL NY US
X-Axis: Year Quarter(s)
Y-Axis: Mean Temperature(s) (in ° C)
Time Period: 01-01-2008 to 31-12-2017
X Labels: [2008Q1, 2008Q2, 2008Q3, 2008Q4, 2009Q1, 2009Q2, 2009Q3, 2009Q4, 2010Q1, 2010Q2, 2010Q3, 2010Q4, 2011Q1, 2011Q2, 2011Q3, 2011Q4, 2012Q1, 2012Q2, 2012Q3, 2012Q4, 2013Q1, 2013Q2, 2013Q3, 2013Q4, 2014Q1, 2014Q2, 2014Q3, 2014Q4, 2015Q1, 2015Q2, 2015Q3, 2015Q4, 2016Q1, 2016Q2, 2016Q3, 2016Q4, 2017Q1, 2017Q2, 2017Q3, 2017Q4]

DEL Quarterly Mean Weather Variation over 10 years



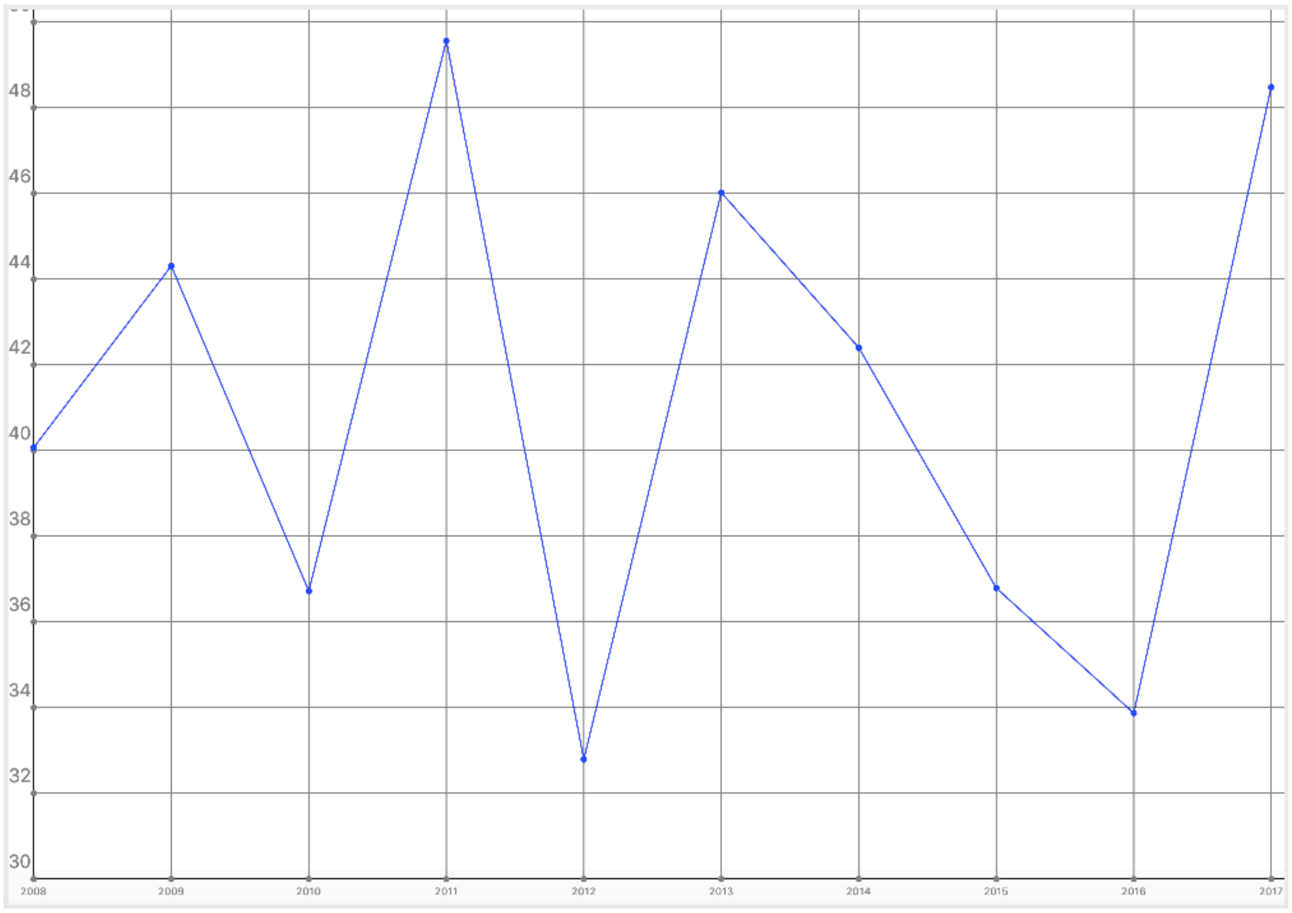
Station: NEW DELHI PALAM IN
X-Axis: Year Quarter(s)
Y-Axis: Mean Temperature(s) (in ° C)
Time Period: 01-01-2008 to 31-12-2017
X Labels: [2008Q1, 2008Q2, 2008Q3, 2008Q4, 2009Q1, 2009Q2, 2009Q3, 2009Q4, 2010Q1, 2010Q2, 2010Q3, 2010Q4, 2011Q1, 2011Q2, 2011Q3, 2011Q4, 2012Q1, 2012Q2, 2012Q3, 2012Q4, 2013Q1, 2013Q2, 2013Q3, 2013Q4, 2014Q1, 2014Q2, 2014Q3, 2014Q4, 2015Q1, 2015Q2, 2015Q3, 2015Q4, 2016Q1, 2016Q2, 2016Q3, 2016Q4, 2017Q1, 2017Q2, 2017Q3, 2017Q4]

CCU Quarterly Mean Weather Variation over 10 years



Station: CALCUTTA DUM DUM IN
X-Axis: Year Quarter(s)
Y-Axis: Mean Temperature(s) (in ° C)
Time Period: 01-01-2008 to 31-12-2017
X Labels: [2008Q1, 2008Q2, 2008Q3, 2008Q4, 2009Q1, 2009Q2, 2009Q3, 2009Q4, 2010Q1, 2010Q2, 2010Q3, 2010Q4, 2011Q1, 2011Q2, 2011Q3, 2011Q4, 2012Q1, 2012Q2, 2012Q3, 2012Q4, 2013Q1, 2013Q2, 2013Q3, 2013Q4, 2014Q1, 2014Q2, 2014Q3, 2014Q4, 2015Q1, 2015Q2, 2015Q3, 2015Q4, 2016Q1, 2016Q2, 2016Q3, 2016Q4, 2017Q1, 2017Q2, 2017Q3, 2017Q4]

BUF Annual Precipitation Variation over 10 years



Station: BUFFALO NIAGARA INTERNATIONAL NY US
X-Axis: Year(s)
Y-Axis: Precipitation Amount(s) (in mm)
Time Period: 01-01-2008 to 31-12-2017
X Labels: [2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017]