DATA 601: HW3

Fall 2019

Due: Wed. Oct. 2, 2019 (by 23:55)

Learning Objectives

- Work with realworld datasets that can be represented using tabular data structures.
- Gain experience wrangling and organizing data using `pandas`.
- Produce visualizations summarizing information from tabular data.

This is an individual homework assignment.

Please complete this homework assignment within the Jupypter notebook environment, making use of Markdown and Code cells to properly format your answers. Please provide solutions where asked.

Your completed Jupyter notebook is to be submitted via the HW2 dropbox on D2L.

Warm up

- Please review class slides on `pandas`.
- Please also review the Calgary Rainfall Jupyter notebook. In this homework, we will use the Calgary Rainfall dataset. Please download the dataset if you already haven't done so. You may use the data provided in class or download it directly from Open Calgary (https://data.calgary.ca/Environment/Historical-Rainfall/d9kv-swk3).

Task 1 (5 points)

Cleanup and organization

- Use `pandas` to read in the data set. Do not discard the datetime information in the columns. Convert the 'TIMESTAMP' column to a `datetime` object (You can use `pandas.to_datetime()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html) to accomplish this).
- You may notice that 'YEAR' column is now redundant. Additionally, for this homework, we won't make use of the 'ID' column. Please discard it (you can use `pandas.DataFrame.drop` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop.html) for this). You can also discard any rows where the channel is not active.
- Display the head (`pandas.DataFrame.head` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.head.html)), tail (`pandas.DataFrame.tail` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.tail.html)) and description (`pandas.DataFrame.describe` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.describe.html)) of the resulting dataframe.

Please provide your solution by inserting appropriate code and markdown cells below this cell.

Cleaning Up The Rainfall Dataset

In [1]:
```
# Load up the data

import pandas as pd
import numpy as np

dataset = pd.read_csv("~/601f19/tabular data/Historical_Rainfall.csv")
dataset.head()
```

Out[1]:

	CHANNEL	YEAR	TIMESTAMP	RAINFALL	RG_ACTIVE	ID
0	42	2019	2019/09/14 05:05:00 AM	0.2	Y	2019-09-14T05:05:00-42
1	42	2019	2019/09/14 01:45:00 AM	0.4	Y	2019-09-14T01:45:00-42
2	42	2019	2019/09/14 01:40:00 AM	0.4	Y	2019-09-14T01:40:00-42
3	48	2019	2019/09/13 10:45:00 AM	0.2	Y	2019-09-13T10:45:00-48
4	21	2019	2019/09/13 10:40:00 AM	0.2	Y	2019-09-13T10:40:00-21

In [2]:
```
# Start cleaning it

cleaned = dataset.drop(columns=["YEAR","ID","TIMESTAMP"])
cleaned = cleaned[cleaned["RG_ACTIVE"] == "Y"]

# Passing in the format is much faster as opposed to letting pandas
# automatically infer the format
cleaned["timestamp"] = pd.to_datetime(dataset["TIMESTAMP"], format='%Y/%
m/%d  %I:%M:%S %p')
```

The Head, Tail, and Description of the Rainfall Dataset

In [3]:
```
cleaned.head()
```

Out[3]:

	CHANNEL	RAINFALL	RG_ACTIVE	timestamp
0	42	0.2	Y	2019-09-14 05:05:00
1	42	0.4	Y	2019-09-14 01:45:00
2	42	0.4	Y	2019-09-14 01:40:00
3	48	0.2	Y	2019-09-13 10:45:00
4	21	0.2	Y	2019-09-13 10:40:00

In [4]:
```
cleaned.tail()
```

Out[4]:

	CHANNEL	RAINFALL	RG_ACTIVE	timestamp
889775	8	0.2	Y	1988-05-20 18:45:00
889776	21	0.2	Y	1988-05-20 18:45:00
889777	8	0.2	Y	1988-05-20 18:25:00
889778	15	0.2	Y	1988-05-20 18:15:00
889779	17	0.2	Y	1988-05-20 12:55:00

In [5]:
```
cleaned.describe()
```

Out[5]:

	CHANNEL	RAINFALL
count	889780.000000	889780.000000
mean	17.493284	0.317491
std	11.116896	0.418900
min	1.000000	0.100000
25%	8.000000	0.200000
50%	16.000000	0.200000
75%	26.000000	0.200000
max	99.000000	43.200000

Task 2 (10 points)

Restructure and determine rainfall daily totals per channel

- We are interested in the daily rainfall totals per channel. Restructure and aggregate your table so that entries now contain *daily totals per channel*.

The precise details of how you accomplish this are up to you. You can for example build a hierarchical index for the rows with the year, month and day. You can also have a hierarchical index on the columns based on the channels. Please make use of `pandas` grouping and aggregation facilities to accomplish this.

Please provide your solution by inserting appropriate code and markdown cells below this cell.

In [6]:
```
# Create some new columns

cleaned['year'] = cleaned["timestamp"].apply( lambda x: x.year )
cleaned['month'] = cleaned["timestamp"].apply( lambda x: x.month )
cleaned['day'] = cleaned["timestamp"].apply( lambda x: x.day )
```

In [7]:
```
# Use group-by and aggregate to consolidate

daily_rainfall_per_channel = cleaned.groupby( by=["year",'month','day',
'CHANNEL'] ).agg(np.sum)
```

Task 3 (15 points)

Visualization

Produce visualizations that show:

- Rainiest day of the year for the years 1989 through 2018. For each day, show the date and the total rainfall. Note that you will need to aggregate over the channels. Use the maximum over the channels for this, i.e. the channel that recorded the most rainfall.
- Average number of rainy days per month. Average over the years 1989 through 2018.
- Rainfall *monthly* statistics such as mean, median, min and max. Aggregate over the years 1989 through 2018 (years for which the data is complete). You will need to aggregate over the channels as well.

The details of what visualizations to use are not spelled out. Please choose a visualization that is appropriate for each of the above tasks and *clearly* shows the requested information. Please also ensure that you provide appropriate labels/legends/colorbars so that your visualizations are readable and self-contained.

Please provide your solution by inserting appropriate code and markdown cells below this cell.

In [8]:
```
# First off, we need to consolidate the channels into daily totals.
#  Our first attempt will do this the easy way, by aggregating across th
e max.

temp = daily_rainfall_per_channel.reset_index()
mask = (temp['year'] >= 1989) & (temp['year'] <= 2018)

daily_rainfall = temp[mask].groupby( by=['year','month','day'] \
                              ).agg( np.max ).drop( columns=["CHANNEL"] )
```

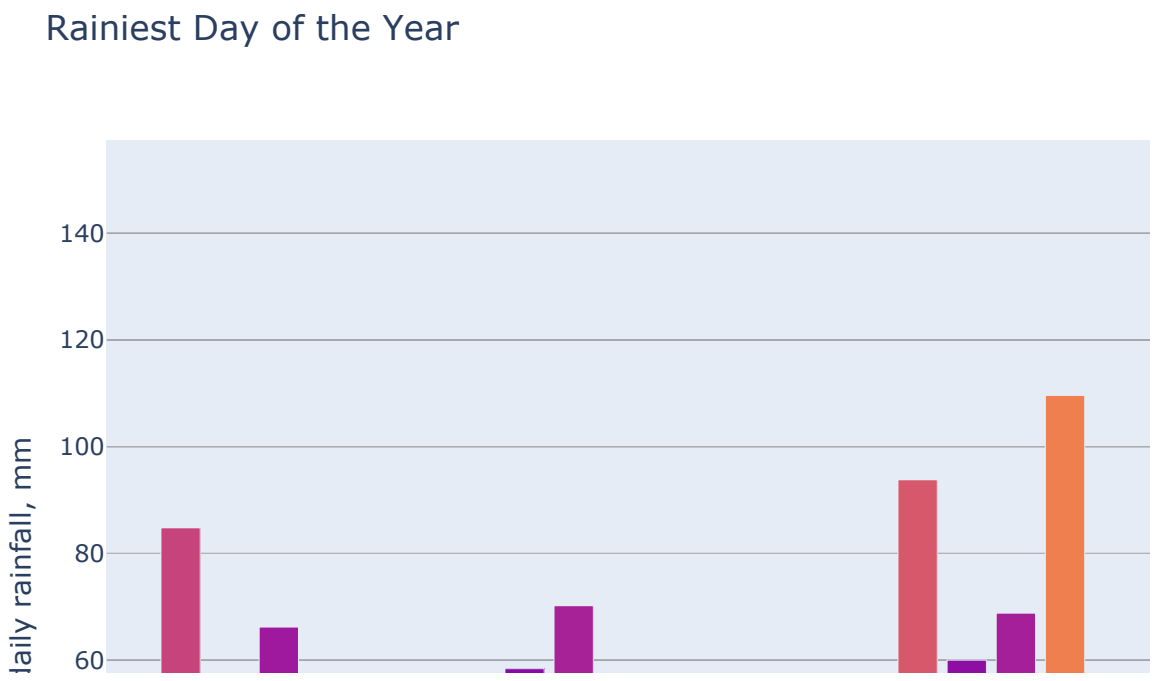
Rainiest Day of the Year

In [9]:
```
# Now to find the rainiest day within each year

daily_rainfall_flat = daily_rainfall.reset_index()
list_of_max_rainfall = list()

for year in set( daily_rainfall_flat['year'] ):

    mask = daily_rainfall_flat['year'] == year
    maximum = daily_rainfall_flat['RAINFALL'][mask].max()

    # in case of multiples:
    list_of_max_rainfall.append( \
                np.random.choice( daily_rainfall_flat[mask & (daily_rain
fall_flat['RAINFALL'] == maximum)].index ))

days_of_max_rainfall = daily_rainfall_flat.loc[ list_of_max_rainfall ]
```

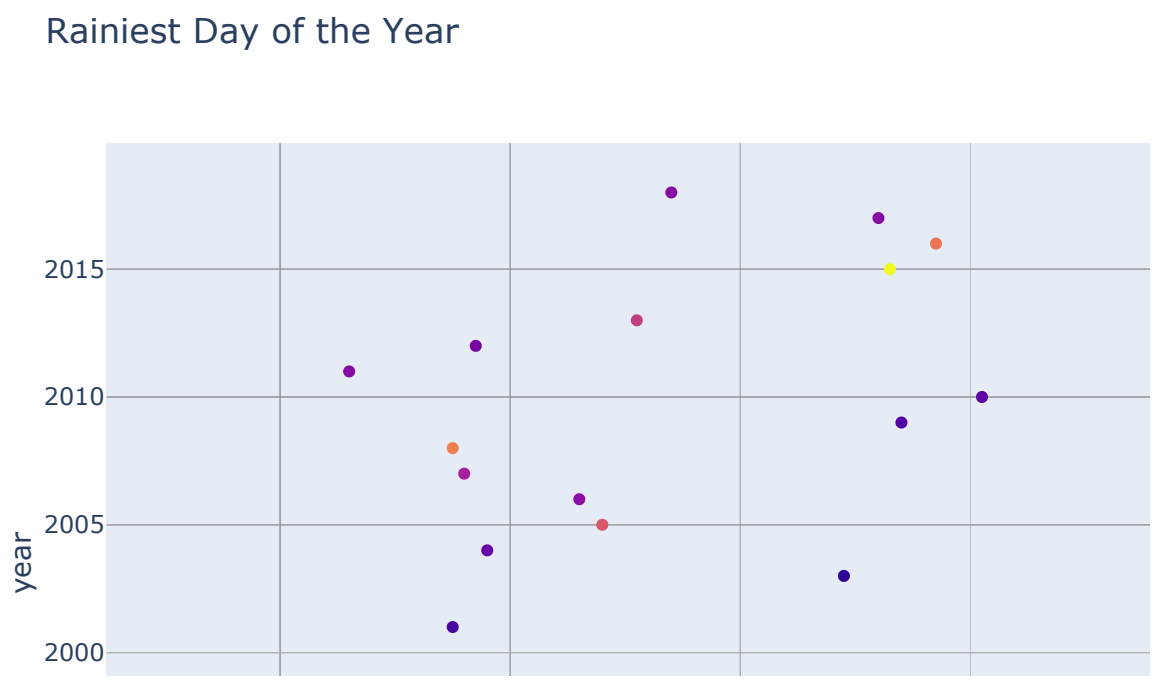
In [10]:

```
# Add some labels and fancy stuff for plotly express
labels = list()
day_of_year = list()

for index, row in days_of_max_rainfall.iterrows():
    labels.append( "{}/{:02d}/{:02d} = {:.1f}mm".format( \
                                int(row['year']), int(row['month']), int
(row['day']), row['RAINFALL']) )
    day_of_year.append( pd.to_datetime( "{}/{}/{}".format( \
                                int(row['year']), int(row['month']), int
(row['day'])) ).dayofyear )

days_of_max_rainfall['labels'] = np.array( labels )
days_of_max_rainfall['day of year'] = np.array( day_of_year )

days_of_max_rainfall.rename(columns={'RAINFALL':'daily rainfall, mm'}, i
nplace=True)
```

In [11]:
```
import plotly.express as px

# Try 1: Just a bar chart
px.bar( days_of_max_rainfall, x='year', y='daily rainfall, mm', hover_na
me='year', hover_data=['labels'],\
       color='daily rainfall, mm', title='Rainiest Day of the Year')
```



In [12]:
```
# Try 2: Scatter plot with dates

px.scatter(days_of_max_rainfall, x='day of year', y='year', \
           hover_name='year', hover_data=['labels'], color='daily rainfa
ll, mm', \
           title='Rainiest Day of the Year')
```

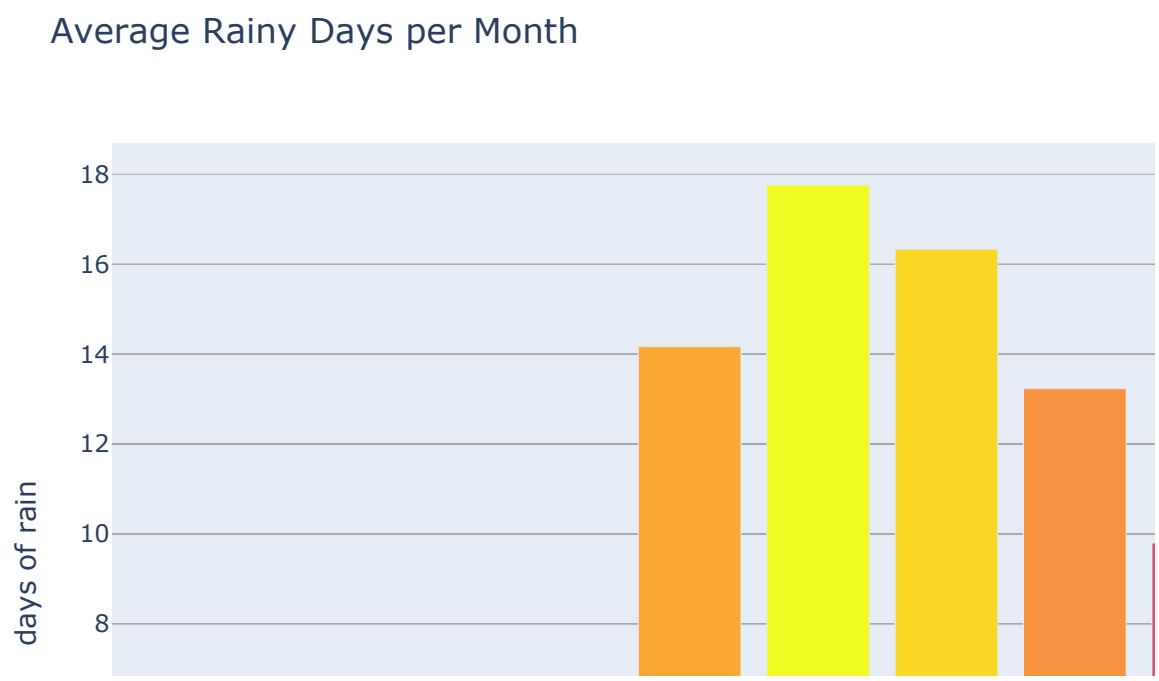


Average Number of Rainy Days per Month

In [13]:

```
month_map = {1:'January', 2:'February', 3:'March', 4:'April', 5:'May', 6
:'June', \
             7:'July', 8:'August', 9:'September', 10:'October', 11:'Nove
mber', 12:'December'}

monthly_rain_totals = daily_rainfall.apply( lambda x: x > 1 ).reset_inde
x().drop( columns=['day'] ).groupby( by=['year','month'] ).agg( np.sum )
monthly_rain_average = monthly_rain_totals.reset_index().drop(columns=[
'year']).groupby( by=['month'] ).agg( np.mean )

needed_months = set( range(1,13) )
for index, row in monthly_rain_average.iterrows():
    needed_months.discard(index)

monthly_rain_average_expanded = monthly_rain_average.append( \
            pd.DataFrame( {'month':list(needed_months), "RAINFALL":[0]*l
en(needed_months)} ).set_index(['month']) )
monthly_rain_average_expanded.sort_index( inplace=True )

monthly_rain_average_labeled = pd.DataFrame( monthly_rain_average_expand
ed, copy=True )
monthly_rain_average_labeled.index = monthly_rain_average_expanded.index
.map( lambda x: month_map[x] )
monthly_rain_average_labeled.rename( columns={'RAINFALL':'days of rain'
}, inplace=True )
```

In [14]:
```
# Try 1: Just a bar chart
px.bar( monthly_rain_average_labeled.reset_index(), x='month', y='days o
f rain', \
       color='days of rain', title='Average Rainy Days per Month' )
```



Rainfall monthly statistics

In [15]:
```
monthly_rain_stats = monthly_rain_average_expanded
temp = daily_rainfall_flat.drop( columns=['year','day'] ).groupby( by=[
'month'] )

monthly_rain_stats['average'] = temp.agg( np.mean )
monthly_rain_stats['median'] = temp.agg( np.median )
monthly_rain_stats['maximum'] = temp.agg( np.max )
monthly_rain_stats['minimum'] = temp.agg( np.min )
monthly_rain_stats['name of month'] = monthly_rain_stats.index.map( lamb
da x: month_map[x] )
```

In [16]:

```
import plotly.graph_objects as go

# Try 1: a line chart
figure = go.Figure()

figure.add_trace( go.Scatter(name='Minimum', x=monthly_rain_stats['name
 of month'], y=monthly_rain_stats['minimum'], \
                              mode='lines', line_color='gray', opacity=0.
1 ) )
figure.add_trace( go.Scatter(name='Maximum', x=monthly_rain_stats['name
 of month'], y=monthly_rain_stats['maximum'], \
                              fill='tonexty', mode='lines', line_color='g
ray', opacity=0.1 ) )
figure.add_trace( go.Scatter(name='Mean', x=monthly_rain_stats['name of
 month'], y=monthly_rain_stats['average'], \
                              line_color='green', opacity=1 ) )
figure.add_trace( go.Scatter(name='Median', x=monthly_rain_stats['name o
f month'], y=monthly_rain_stats['median'], \
                              mode='markers', line_color='gray', opacity=
1 ) )

figure.update_layout(
    title=go.layout.Title(text="Rainfall Monthly Statistics"),
    yaxis=go.layout.YAxis( title=go.layout.yaxis.Title(text="rainfall, m
m") ),
    yaxis_type="log"
    )

figure.show()
```

Rainfall Monthly Statistics