In [2]:

```
import dtale
import pandas as pd

data1 = pd.read_csv('temperature.csv',header=0)
data2 = pd.read_csv('population.csv',header=0)

temp = pd.DataFrame(data1)
pop = pd.DataFrame(data2)
```

In [ ]:

```
# Just for convenience - Population dataset
dtale.show(pop)
```

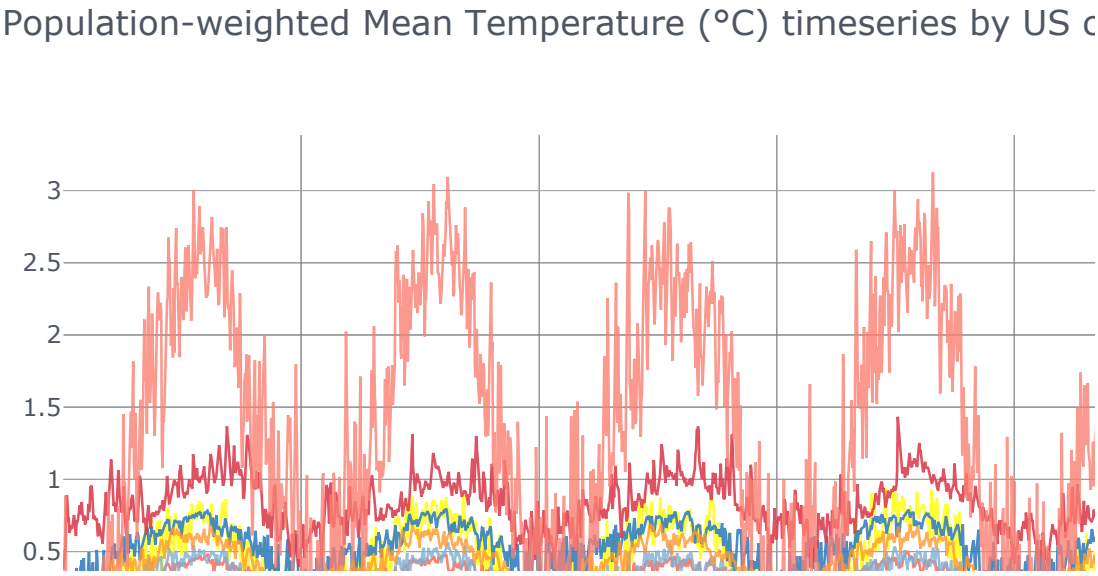
In [ ]:

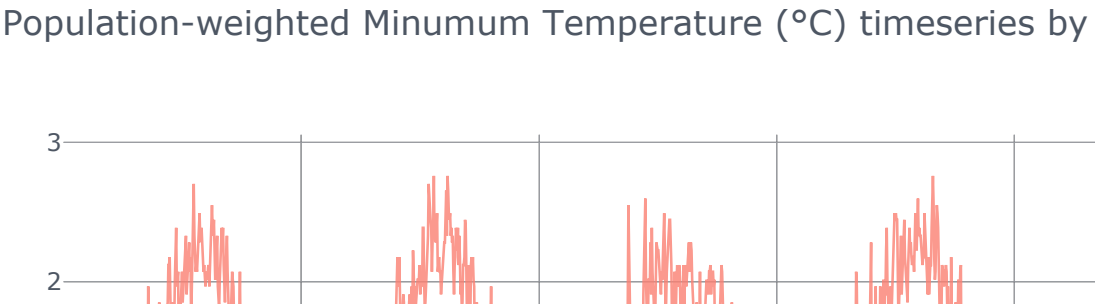
```
# Just for convenience - Temperature dataset
dtale.show(temp)
```

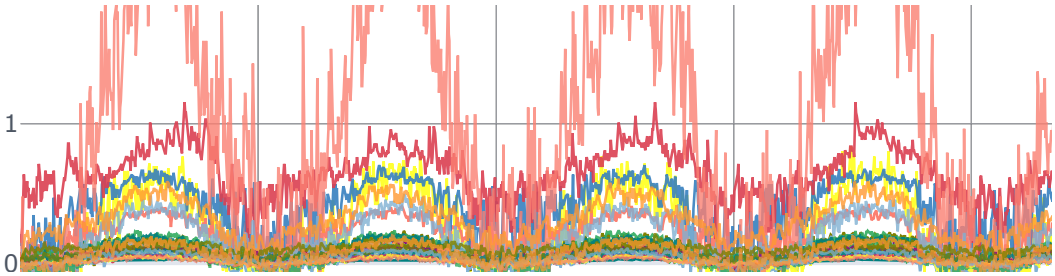
In [3]:

```
############################
########## PART 1 ##########
############################

import cufflinks as cf
from IPython.display import display,HTML
# making all charts public and setting a global theme
cf.set_config_file(sharing='public',theme='white',offline=True)

# Manually cleaning up few city names in the temperature dataset
# In order to correctly match cities when merging
temp['name'] = temp['name'].str.split('/').str[0]
temp.loc[temp['name']=='NYC','name'] = 'New York'
temp.loc[temp['name']=='Wash DC','name'] = 'Washington'
temp.loc[temp['name']=='Chicago O\'Hare','name'] = 'Chicago'
temp.loc[temp['name']=='St Louis','name'] = 'St. Louis'

# Population weight: (city_population / total_population)
pop['ratio'] = pop['population']/(pop['population'].sum())

# Merging population and temperature dataframes
merged = pop.merge(temp, left_on='City', right_on='name', how='outer').dropna()

# Multiplying the corresponding population weight to each city's temperature
merged['temp_mean_c'] = merged['temp_mean_c']*merged['ratio']
merged['temp_min_c'] = merged['temp_min_c']*merged['ratio']
merged['temp_max_c'] = merged['temp_max_c']*merged['ratio']

# Further cleaning the merged dataframe
merged = merged.drop(columns=['name', 'ratio', 'continent', 'country_code', 'cou
merged = merged.drop(columns=['State', 'population', 'Lon', 'Lat', 'station_code

# Change location_date column to datetime type
merged['location_date'] =  pd.to_datetime(merged['location_date'])

# Creating a datetime dataframe in order to merge and identify which dates were
df = pd.DataFrame({'dates':pd.date_range('2015-01-01','2021-04-20')})

merged_min = merged[['City','location_date','temp_min_c']]
merged_mean = merged[['City','location_date','temp_mean_c']]
merged_max = merged[['City','location_date','temp_max_c']]

########## Mininum temperature ##########
merged_min = merged_min.set_index('location_date')
merged_min = merged_min.pivot_table(values='temp_min_c', index=merged_min.index,
merged_min = merged_min.merge(df, left_on='location_date', right_on='dates', how
merged_min = merged_min.set_index('dates').sort_values(by='dates', ascending=Tru
missing_min = merged_min # to be used for later
# for a missing data, render a reasonably predictive data by getting the average
merged_min = merged_min.where(merged_min.notnull(), other=(merged_min.fillna(met

##########  Maximum temperature ##########
merged_max = merged_max.set_index('location_date')
merged_max = merged_max.pivot_table(values='temp_max_c', index=merged_max.index,
merged_max = merged_max.merge(df, left_on='location_date', right_on='dates', how
merged_max = merged_max.set_index('dates').sort_values(by='dates', ascending=Tru
missing_max = merged_max # to be used for later
# for a missing data, render a reasonably predictive data by getting the average
merged_max = merged_max.where(merged_max.notnull(), other=(merged_max.fillna(met
```

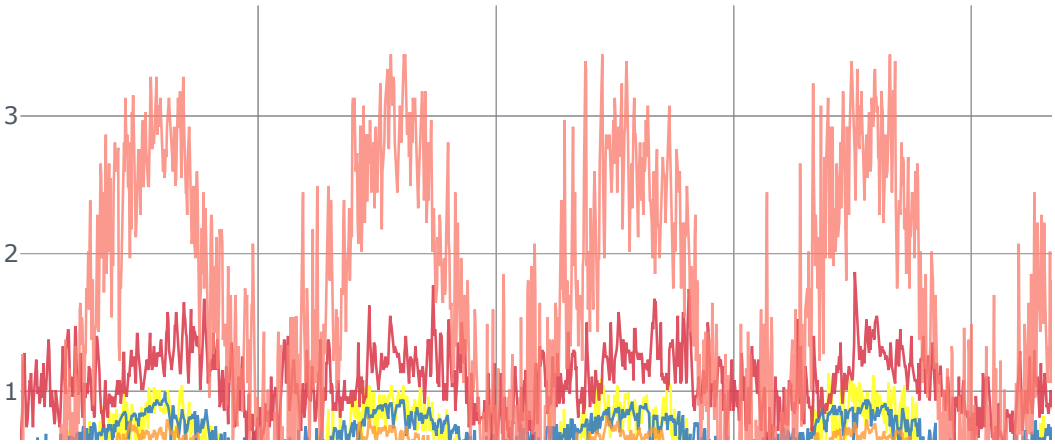
```
60
61  ##########  Mean temperature ##########
62  merged_mean = merged_mean.set_index('location_date')
63  merged_mean = merged_mean.pivot_table(values='temp_mean_c', index=merged_mean.ir
64  merged_mean = merged_mean.merge(df, left_on='location_date', right_on='dates', h
65  merged_mean = merged_mean.set_index('dates').sort_values(by='dates', ascending=T
66  missing_mean = merged_mean # to be used for later
67  # for a missing data, render a reasonably predictive data by getting the average
68  merged_mean = merged_mean.where(merged_mean.notnull(), other=(merged_mean.fillna
69
70  # Plotting
71  ########## You can click the label on the legend to unsee the chosen line on the
72  merged_mean.iplot(kind='line', title='Population-weighted Mean Temperature (°C)
73  merged_min.iplot(kind='line', title='Population-weighted Minumum Temperature (°C
74  merged_max.iplot(kind='line', title='Population-weighted Maximum Temperature (°C
75
76
```





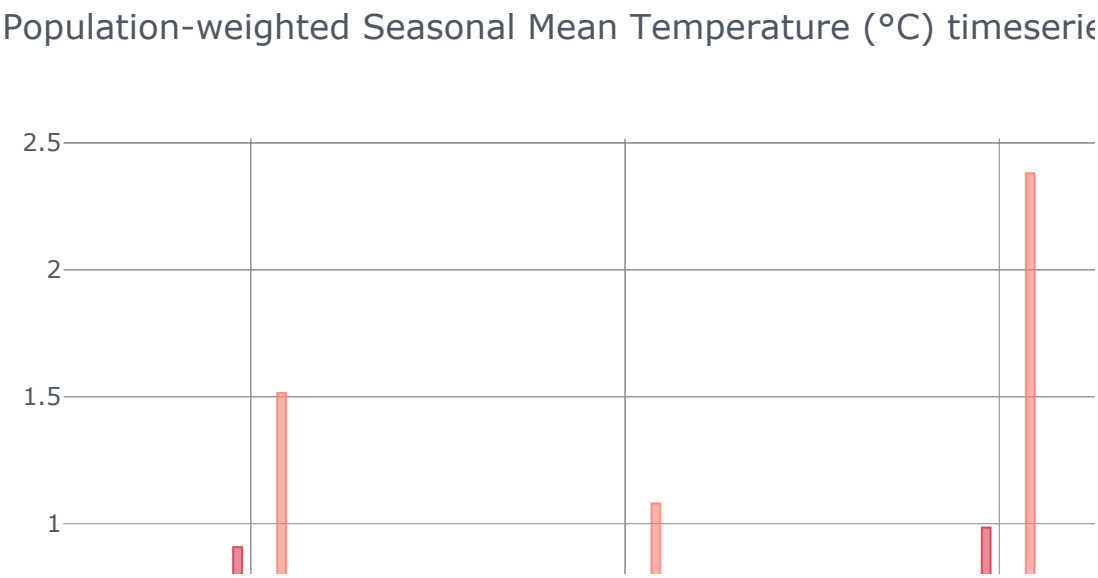


Population-weighted Maximum Temperature (°C) timeseries by

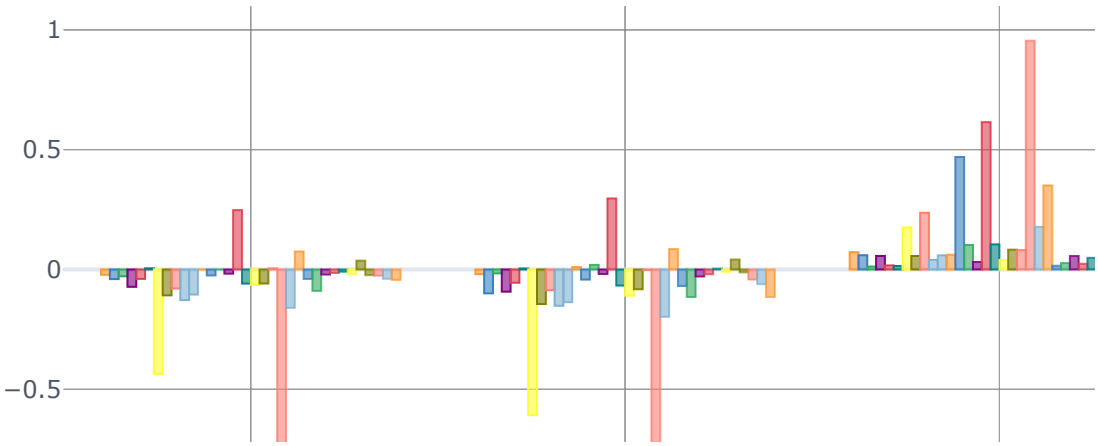


In [4]:

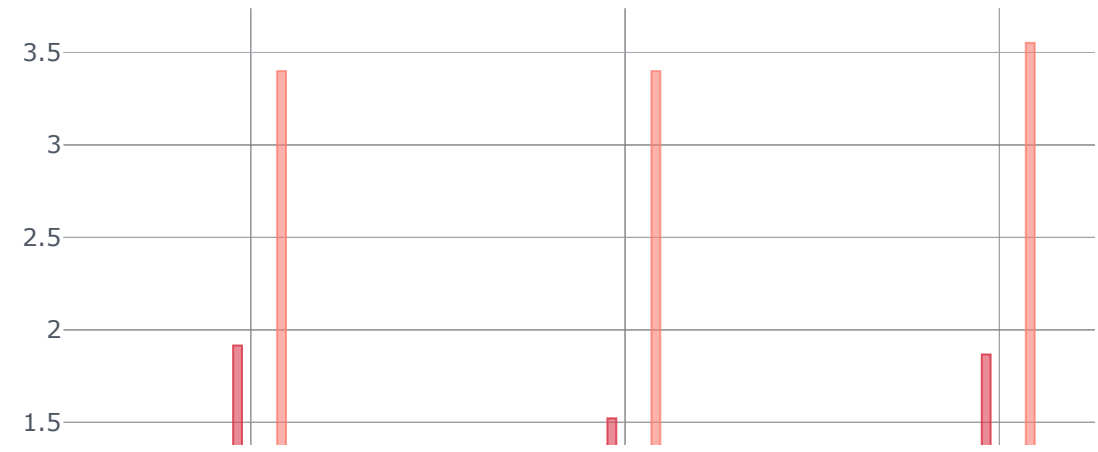
```
############################
########## PART 2 ##########
############################
######## (1) Season ########

s_mean = merged_mean.reset_index(level='dates')
s_min = merged_min.reset_index(level='dates')
s_max = merged_max.reset_index(level='dates')


def get_season(row):
    if row['dates'].month >= 3 and row['dates'].month <= 5:
        return 'Spring'
    elif row['dates'].month >= 6 and row['dates'].month <= 8:
        return 'Summer'
    elif row['dates'].month >= 9 and row['dates'].month <= 11:
        return 'Fall'
    else:
        return 'Winter'

# Comparing mean temperature for each season
s_mean['Season'] = s_mean.apply(get_season, axis=1)
seasmean = s_mean.groupby(s_mean['Season']).mean()

# Comparing min temperature for each season
s_min['Season'] = s_min.apply(get_season, axis=1)
seasmin = s_min.groupby(s_min['Season']).min()
seasmin = seasmin.drop(columns='dates')

# Comparing max temperature for each season
s_max['Season'] = s_max.apply(get_season, axis=1)
seasmax = s_max.groupby(s_max['Season']).max()
seasmax = seasmax.drop(columns='dates')

# Plot
seasmean.iplot(kind='bar', title='Population-weighted Seasonal Mean Temperature
seasmin.iplot(kind='bar', title='Population-weighted Seasonal Min Temperature (°
seasmax.iplot(kind='bar', title='Population-weighted Seasonal Max Temperature (°
```



Population-weighted Seasonal Min Temperature (°C) timeseries



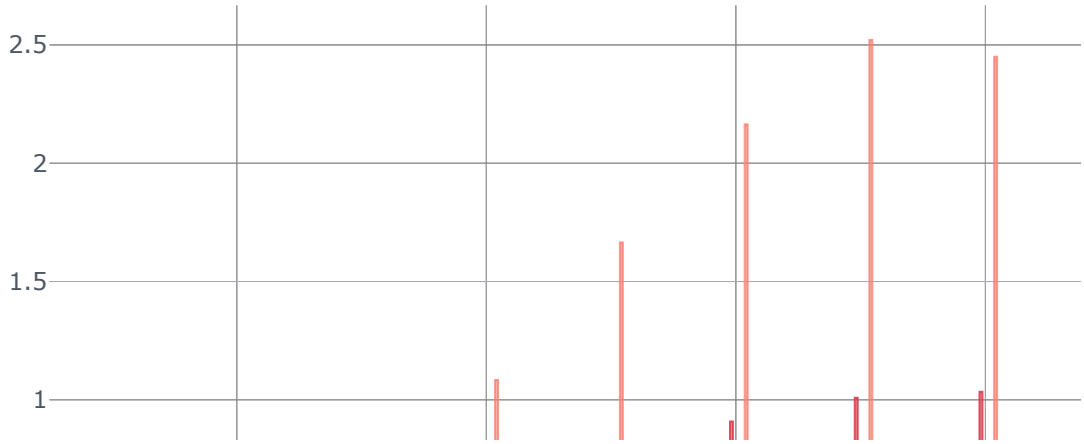
Population-weighted Seasonal Max Temperature (°C) timeseries



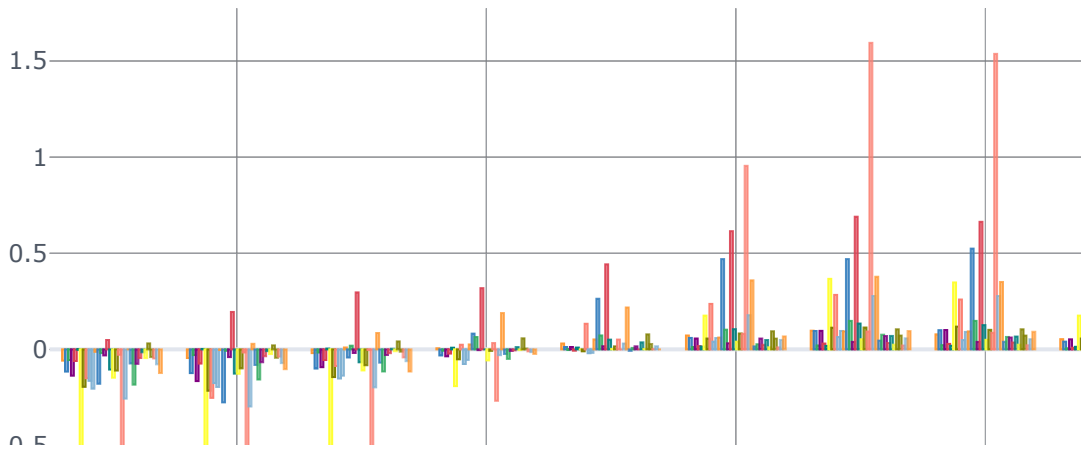
In [5]:

```
############################
########## PART 2 ##########
############################
######## (2) Month #########

m_mean = merged_mean.reset_index(level='dates')
m_min = merged_min.reset_index(level='dates')
m_max = merged_max.reset_index(level='dates')


def get_month(row):
    if row['dates'].month == 1:
        return 1
    elif row['dates'].month == 2:
        return 2
    elif row['dates'].month == 3:
        return 3
    elif row['dates'].month == 4:
        return 4
    elif row['dates'].month == 5:
        return 5
    elif row['dates'].month == 6:
        return 6
    elif row['dates'].month == 7:
        return 7
    elif row['dates'].month == 8:
        return 8
    elif row['dates'].month == 9:
        return 9
    elif row['dates'].month == 10:
        return 10
    elif row['dates'].month == 11:
        return 11
    else:
        return 12

# Comparing mean temperature for each month
m_mean['month'] = m_mean.apply(get_month, axis=1)
monmean = m_mean.groupby(m_mean['month']).mean()
monmean = monmean.sort_index(ascending=True)

# Comparing min temperature for each month
m_min['month'] = m_min.apply(get_month, axis=1)
monmin = m_min.groupby(m_min['month']).min()
monmin = monmin.sort_index(ascending=True)
monmin = monmin.drop(columns='dates')

# Comparing max temperature for each month
m_max['month'] = m_max.apply(get_month, axis=1)
monmax = m_max.groupby(m_max['month']).max()
monmax = monmax.sort_index(ascending=True)
monmax = monmax.drop(columns='dates')

# Plot
monmean.iplot(kind='bar', title='Population-weighted Monthly Mean Temperature (°
monmin.iplot(kind='bar', title='Population-weighted Monthly Min Temperature (°C)
monmax.iplot(kind='bar', title='Population-weighted Monthly Max Temperature (°C)

```

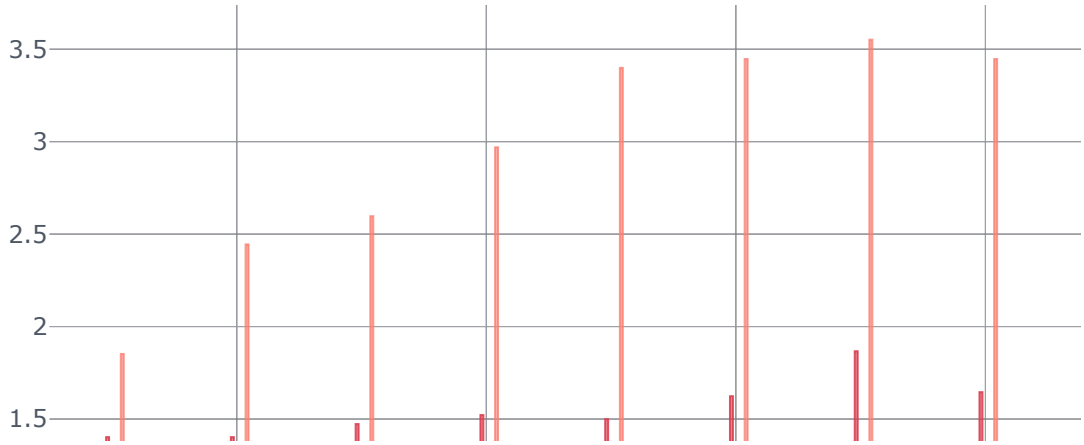
Population-weighted Monthly Mean Temperature (°C) timeseries



Population-weighted Monthly Min Temperature (°C) timeseries b



Population-weighted Monthly Max Temperature (°C) timeseries



In [6]:

```
############################
########## PART 2 ##########
############################
#### (3) Missing Data ######

# Dichotomizing missing/non-missing values
mizz_mean = missing_mean.isna()
# Plotting - Each scatter marker displays the date and the name of the city of t
mizz_mean = mizz_mean.iplot(kind='scatter', title='Missing data (marked true)',
```