Michelle Helfman Term Project Milestone 4

Moving Starter Kit

The Moving Starter Kit contains basic demographic, economic, education, and additional location-based information to be used as a starting point to finding a new city to live or confirm the current location is the best place to be. The addition of weather is another piece of information to help with the decision.

Note - There are no outliers or bad data. To obtain weather data, a city and state is required so parts of the Moving Starter Kit file is used to drive the process.

In [1]:
```
# Import Functions

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import ssl
import re
import sys
import os
import json

from datetime import datetime
from requests.exceptions import HTTPError
from scipy.constants import convert_temperature

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Read in the Moving Starter Kit, load Weather API key
# and delete output file.

# Moving Starter Kit - Use the metro_short, anchor_city, and state_code columns
MSK_df = pd.read_excel('Moving Starter Kit Flat File.xlsx', usecols='B, D, F')

with open('VC_Weather_APIkey.json') as df:
    key = json.load(df)
    vc_weather_api = key['weather_api']

# Delete the existing output file.
file = 'MSK Milestone 4.xlsx'
location = "C:/DSC540_Data/"
path = os.path.join(location, file)

# Remove the file
try:
    os.remove(path)

except:
    print('No Prior File Deleted')
```

In [3]:

```python
# Error handling for the API requests.
# HTTP errors are more granular

def HTTP_errors(response_code, function):

    if response_code == 400:
        err_msg = '400 - Bad Request, Unable To Retrieve ' + function + ' Information.'
    elif response_code == 401:
        err_msg = ('401 - Unauthorized Access, Unable To Retrieve ' + function +
                   ' Information.')
    elif response_code == 403:
        err_msg = '403 - Forbidden Access, Unable To Retrieve ' + function + ' Information.'
    elif response_code == 404:
        err_msg = '404 - Not Found, Unable To Retrieve ' + function + ' Information.'
    elif response_code == 500:
        err_msg = ('500 - Internal Server Error, Unable To Retrieve ' + function +
                   ' Information.')
    elif response_code == 502:
        err_msg = '502 - Bad Gateway, Unable To Retrieve ' + function + ' Information.'
    elif response_code == 503:
        err_msg = ('503 - Service Unavailable, Unable To Retrieve ' + function +
                   ' Information.')
    else:
        err_msg = 'Other ' + function + ' API Error.'

    return err_msg
```

1. Parse the Weather Data

Separate the weather information by forcast date

2. Convert the Temperatures

Convert the temperatures from Kelvin to Fahrenheit

3. Format Data into a Readable Format

Round integers, create percentages, remove extraneous brackets, create a discription of cloud cloud cover based on percentage, and create new columns for forecast prose from the weather information

4. Transpose the Weather DataFrame

Flip the information into separate columns instead of rows.

5. Rename Headers

Rename the columns holding the parsed and formatted weather information

In [4]:

```
# format the weather information
# Note - The metro_short column is for merging the dataframes back together
def format_weather_info(weather_info, metro_short):

    new_headers = {0: 'metro_short', 1: 'longitude', 2: 'latitude',
                3: 'forecast_today', 4: 'forecast_tom'}
    f_err_code = 0
    forecast_today = ' '
    forecast_tom = ' '
    longitude = weather_info["longitude"]
    latitude = weather_info["latitude"]
    days = weather_info['days']

# Parse weather data by day
    for d in days:
        w_day = d["datetime"]
        if w_day == '2023-05-21':
            tod_tom = 'Today'
        else:
            tod_tom = 'Tomorrow'
        tempmax = d["tempmax"]
        tempmin = d["tempmin"]
        avgtemp = d["temp"]
        precipprob = d["precipprob"]
        cloudcover = int(d["cloudcover"])

# Convert min, max, and average temperatures from Kelvin to Fahrenheit
        tempmax1 = convert_temperature(np.array([tempmax]), 'Kelvin', 'Fahrenheit')
        tempmin1 = convert_temperature(np.array([tempmin]), 'Kelvin', 'Fahrenheit')
        avgtemp1 = convert_temperature(np.array([avgtemp]), 'Kelvin', 'Fahrenheit')

# Create percipitation percentage
        precipprob1 = str(round(precipprob)) + '%'

# Round the temperature to a whole number
        tempmax1 = str(tempmax1.round())
        tempmax1a = str(tempmax1).lstrip('[').rstrip('.]')
        tempmin1 = str(tempmin1.round())
        tempmin1a = str(tempmin1).lstrip('[').rstrip('.]')
        avgtemp1 = str(avgtemp1.round())
        avgtemp1a = str(avgtemp1).lstrip('[').rstrip('.]')

# translate the cloud cover to prose based on the percentage.
        if cloudcover == 100:
            cloudy = 'Completely Cloudy'
        elif cloudcover > 80:
            cloudy = 'Mostly Cloudy'
        elif cloudcover > 60:
            cloudy = 'Partly Cloudy'
        elif cloudcover > 40:
            cloudy = 'Partly Sunny'
        elif cloudcover > 20:
            cloudy = 'Mostly Sunny'
        else:
            cloudy = 'Clear'

# Create forecast for the city and state
        forecast = ('The weather forcast for ' + tod_tom + ' has a high temperature of ' +
                    tempmax1a + 'F, a low temperature of ' + tempmin1a +
                    'F, with an average of ' + avgtemp1a + 'F.  There is a ' + precipprob1 +
                    ' chance of rain under ' + cloudy + ' skies.')
        if tod_tom == 'Today':
            forecast_today = forecast
        else:
            forecast_tom = forecast

# Create the temporary dataframe from the weather info
# Include the metro_short column for matching with the MSK_df
```

```
# Place the information in a list, load the temporary dataframe
# and flip the information into separate columns and rename the headers
        temp_list = [metro_short, longitude, latitude, forecast_today, forecast_tom]
        temp_df = pd.DataFrame(temp_list)
        temp_df = temp_df.transpose()
        temp_df.rename(columns = new_headers, inplace = True)

    return temp_df, f_err_code
```

In [5]:

```
# Get weather forecasts for today and tomorrow and check for errors
def get_weather_info(weather_loc):

    w_err_code = 0
    units = 'base'
    try:
# Retrieve weather forecast information for today and tomorrow
        weather_data = requests.get(f'https://weather.visualcrossing.com/'
                                    f'VisualCrossingWebServices/rest/services/'
                                    f'timeline/{weather_loc}/2023-05-21/2023-05-22?'
                                    f'&key={vc_weather_api}&unitGroup={units}&include=days'
                                    f'&elements=datetime,tempmax,tempmin,temp,precipprob,'
                                    f'cloudcover')

        weather_data.raise_for_status()
        w_err_code = 0

# If there's an exception on the API request send error to error
# handling code and print a message.  If no error set error code to 0
    except HTTPError as http_err:
        w_response = weather_data.status_code
        error_message = HTTP_errors(w_response, 'Weather')
        w_err_code = -99
        print('HTTP Errors = ', error_message)

    if w_err_code == 0:
        weather_info = json.loads(weather_data.text)
        if len(weather_info) > 0:
            w_err_code = 0
        else:
            w_err_code = -99

    else:
        w_err_code = -99

    return weather_info, w_err_code
```

In [6]:
```
# Retrieve the weather information using the Moving Starter
# Kit one row at a time.
weather_df = pd.DataFrame(columns=['metro_short', 'longitude','latitude',
                                   'forecast_today','forecast_tom'])

for index, row in MSK_df.iterrows():
# Save the MSK variables and create the location parameter
    metro_short = row['metro_short']
    anchor_city = row['anchor_city']
    state_cd = row['state_code']
    country = 'US'
    weather_loc = anchor_city + ',' + state_cd + ',' + country

# Get weather info from visual crossing
    weather_info, w_err_code = get_weather_info(weather_loc)

# If no error format the weather info
    if w_err_code == 0:
# Format weather information
        temp_df, f_err_code = format_weather_info(weather_info, metro_short)
    else:
        print("No Weather Information, Check Your City and State.  ",
              anchor_city + ' ' + state_cd)
    weather_df = weather_df.append(temp_df)
```

6. Merge the Weather and Moving Starter Kit information

Merge the weather and MSK dataframes on the metro_short column

7. Add a Timestamp column

Add a timestamp column to the end of the weather info dataframe

In [7]:
```
# Merge the weather and MSK dataframes, add a timestamp column,
# sort the results, and write to an excel file

# Add the anchor city and state to the weather dataframe
weather_info_df = weather_df.merge(MSK_df)

# Add a timestamp
weather_info_df['create_date'] = datetime.now()

# Sort the information
weather_info_df.sort_values('metro_short')

# Write out the Weather Information
weather_info_df.to_excel("C:/DSC540_Data/MSK Milestone 4.xlsx",
                         sheet_name='Weather Information')

print('The End')
```

```
The End
```