Notebook

October 15, 2024

[1]:
```
import pandas as pd
import numpy as np
```

[5]:
```
data  = pd.read_csv("88758_1948_1_1_2022-3.csv")
```

[6]:
```
data.describe()
```

[6]:

	COOPID	YEAR	MONTH	DAY	PRECIPITATION	\
count	27443.0	27443.000000	27443.000000	27443.000000	27443.000000	
mean	88758.0	1984.991583	6.514813	15.753453	-4.343804	
std	0.0	21.645405	3.450984	8.810275	20.777564	
min	88758.0	1948.000000	1.000000	1.000000	-99.990000	
25%	88758.0	1966.000000	4.000000	8.000000	0.000000	
50%	88758.0	1985.000000	7.000000	16.000000	0.000000	
75%	88758.0	2004.000000	10.000000	23.000000	0.040000	
max	88758.0	2022.000000	12.000000	31.000000	8.860000	

	MAX TEMP	MIN TEMP	MEAN TEMP
count	27443.000000	27443.000000	27443.000000
mean	71.565795	49.409908	60.476362
std	39.017565	35.421789	36.996575
min	-99.900000	-99.900000	-99.900000
25%	70.000000	43.000000	56.500000
50%	81.000000	58.000000	69.000000
75%	89.000000	70.000000	79.500000
max	105.000000	81.000000	91.000000

[18]:
```
data.columns
```

[18]:
```
Index(['COOPID', 'YEAR', 'MONTH', 'DAY', 'PRECIPITATION', 'MAX TEMP',
       'MIN TEMP', 'MEAN TEMP'],
      dtype='object')
```

[23]:
```
data.dtypes #checking if the YEAR, MONTH, DAY are of same datatype
```

[23]:
```
COOPID             int64
YEAR               int64
MONTH              int64
```

```
DAY                int64
PRECIPITATION    float64
MAX TEMP         float64
MIN TEMP         float64
MEAN TEMP        float64
dtype: object
```

[24]:
```
# since we have different columns for YEAR, MONTH, DAY we have to transform␣
 ↪them and make a different variable called day_of_year
(data[['YEAR', 'MONTH', 'DAY']].isnull().sum())
```

[24]:
```
YEAR     0
MONTH    0
DAY      0
dtype: int64
```

4) If you suspected that a few of the temperature readings were inaccurate, describe how you would select the possible a candidates and identify which ones those are in the dataset. How would you determine whether these are inaccuracies or just strange weather patterns for those days?

[29]:
```
# Now, checking if the data has any missing values and abnormal temperatures
missing_values = data.isnull().sum()
# abnormal temperature
anomalous_temp = data[(data['MAX TEMP'] > 130) | (data['MIN TEMP'] < -50)]

missing_values, anomalous_temp
```

[29]:
```
(COOPID           0
 YEAR             0
 MONTH            0
 DAY              0
 PRECIPITATION    0
 MAX TEMP         0
 MIN TEMP         0
 MEAN TEMP        0
 date_str         0
 dtype: int64,
        COOPID  YEAR  MONTH  DAY  PRECIPITATION  MAX TEMP  MIN TEMP  MEAN TEMP
\
 31      88758  1948      2    1         -99.99     -99.9     -99.9      -99.9
 32      88758  1948      2    2         -99.99     -99.9     -99.9      -99.9
 33      88758  1948      2    3         -99.99     -99.9     -99.9      -99.9
 34      88758  1948      2    4         -99.99     -99.9     -99.9      -99.9
 35      88758  1948      2    5         -99.99     -99.9     -99.9      -99.9
 …       …    …    …  …             …        …       …         …
 21653   88758  2007      2   29         -99.99     -99.9     -99.9      -99.9
 22385   88758  2009      2   29         -99.99     -99.9     -99.9      -99.9
```

```
22751   88758  2010      2   29         -99.99     -99.9     -99.9      -99.9
23117   88758  2011      2   29         -99.99     -99.9     -99.9      -99.9
23849   88758  2013      2   29         -99.99     -99.9     -99.9      -99.9

        date_str
31      1948-2-1
32      1948-2-2
33      1948-2-3
34      1948-2-4
35      1948-2-5
…          …
21653  2007-2-29
22385  2009-2-29
22751  2010-2-29
23117  2011-2-29
23849  2013-2-29

[1231 rows x 9 columns])
```

4) As we can see, there are quite a few anamolies in the temp readings. So what we did was: we replaced the values of these anamolies with NaN and then used ffill method to fill these anamoly values.

[37]:
```
data.columns = data.columns.str.strip()
```

[38]:
```
invalid_values = [-99.9, -99.99]
```

[45]:
```
# cleaning the temperature-related columns using .loc to avoid chained␣
 ↪assignment warnings
data.loc[:, 'MAX TEMP'] = data['MAX TEMP'].replace(invalid_values, np.nan)
data.loc[:, 'MIN TEMP'] = data['MIN TEMP'].replace(invalid_values, np.nan)
data.loc[:, 'MEAN TEMP'] = data['MEAN TEMP'].replace(invalid_values, np.nan)

# filling NaN values using forward fill directly
climate_data_filled = data.ffill()

# calculating the yearly average mean temperature with filled NaN values
yearly_avg_temp_filled = climate_data_filled.groupby('YEAR')['MEAN TEMP'].mean()
```

1) Finding Day of the year has the highest variance in high temperature between 1948 and 2022:

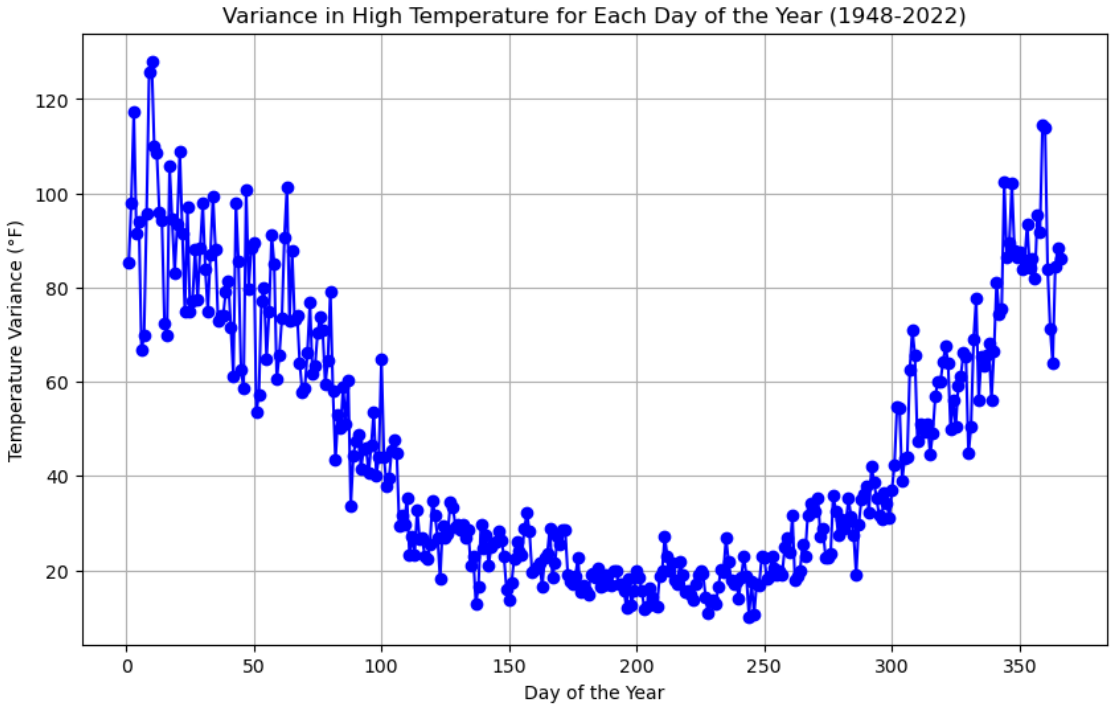
[57]:
```
#now, we are going to use this cleaned data for further analysis
#creating a column for 'Day of Year' as we have three different features DAY,␣
 ↪MONTH, YEAR
climate_data_filled.loc[:, 'date'] = pd.to_datetime(cleaned_data[['YEAR',␣
 ↪'MONTH', 'DAY']])
climate_data_filled.loc[:, 'day_of_year'] = climate_data_filled['date'].dt.
 ↪dayofyear
```

```
# grouping the data by 'day_of_year' and calculate the variance in 'MAX TEMP'␣
 ↪for each day
variance_per_day = climate_data_filled.groupby('day_of_year')['MAX TEMP'].var()

# finding the day of the year with the highest variance
highest_variance_day = variance_per_day.idxmax()
highest_variance_value = variance_per_day.max()

sample_year = data['YEAR'].min()  # taking the earliest year (1948 in this case)
#highest_variance_date = pd.Timestamp(year=sample_year, month=1, day=1) + pd.
 ↪Timedelta(days=highest_variance_day - 1)

highest_variance_day, highest_variance_value
#highest_variance_date
```

[57]: (10.0, 128.07433489827855)

Here, we have got the output as the tenth day of the year with highest variance in high temperature is 10. We verified this output by plotting the variance per day

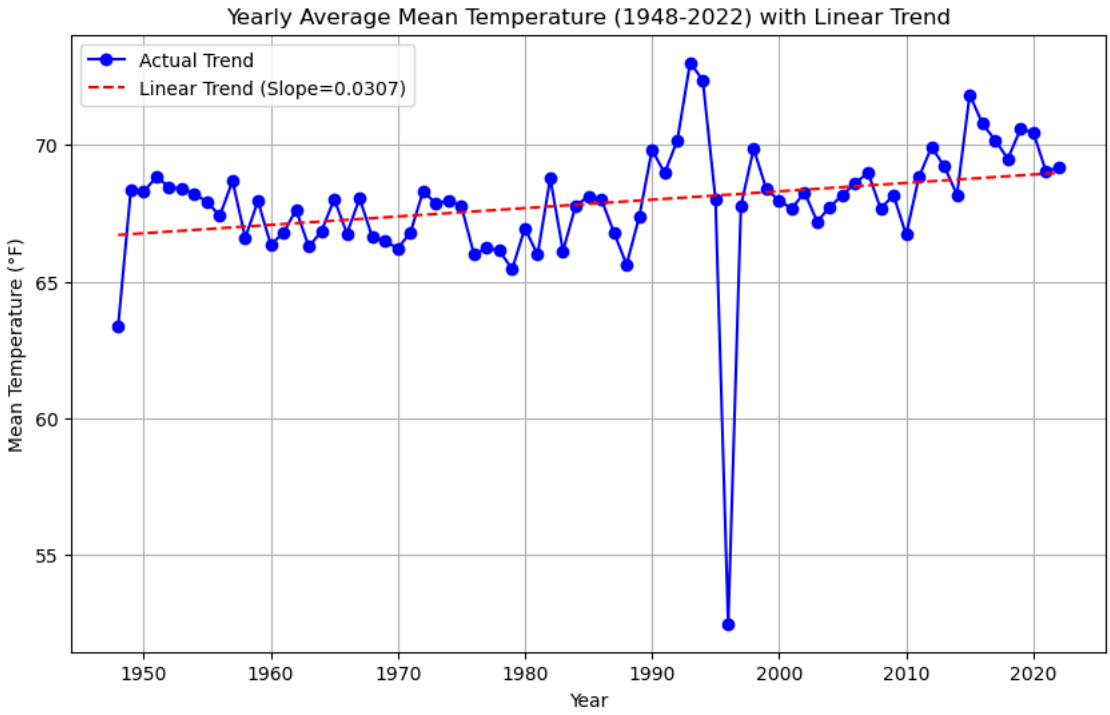
[58]:
```
# plotting the variance per day of the year:
plt.figure(figsize=(10, 6))
plt.plot(variance_per_day.index, variance_per_day.values, marker='o',␣
 ↪linestyle='-', color='b')
plt.title('Variance in High Temperature for Each Day of the Year (1948-2022)')
plt.xlabel('Day of the Year')
plt.ylabel('Temperature Variance (°F)')
plt.grid(True)
plt.show()
```



2) Checking if the dataset suggest a warming trend between 1948 and 2022

[53]:
```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
# preparing the data for linear regression
X = yearly_avg_temp_filled.index.values.reshape(-1, 1)  # Years as the␣
 ↪independent variable
y = yearly_avg_temp_filled.values  # Mean temperature as the dependent variable

# initializing and fitting the linear regression model
linear_regressor = LinearRegression()
linear_regressor.fit(X, y)

# finding the coefficient and intercept of the fitted line
coef = linear_regressor.coef_[0]
intercept = linear_regressor.intercept_

# predicting the temperature values based on the linear regression model
y_pred = linear_regressor.predict(X)

# plotting the actual trend and the linear regression line
plt.figure(figsize=(10, 6))
plt.plot(yearly_avg_temp_filled.index, yearly_avg_temp_filled.values,␣
 ↪marker='o', linestyle='-', color='b', label='Actual Trend')
```

```
plt.plot(yearly_avg_temp_filled.index, y_pred, color='r', linestyle='--',␣
 ↪label=f'Linear Trend (Slope={slope:.4f})')
plt.title('Yearly Average Mean Temperature (1948-2022) with Linear Trend')
plt.xlabel('Year')
plt.ylabel('Mean Temperature (°F)')
plt.legend()
plt.grid(True)
plt.show()

coef
```



[53]: 0.030672314933156225

2) Yes, the dataset shows a warming trend between 1948 and 2022 as we have a positive slope for mean temperatures as shown in the graph above

3) If I am planning a 30 day vacation in Tallahassee and I want the high temperature to fall between 75 and 80 the most days possible, what day should I plan to arrive? Use the total number of days from 1948 to 2022 that fall in that range and maximize the total.

[66]:
```
# Modify the function to return only the month and day, excluding the year
def best_vacation_day(data):
    # Ensure 'Date' is a datetime column using .loc to avoid chained assignment␣
 ↪issues
```

```
    data = data.copy()  # Work on a copy to avoid issues with the original␣
↪dataframe
    data.loc[:, 'Date'] = pd.to_datetime(data[['YEAR', 'MONTH', 'DAY']],␣
↪errors='coerce')  # Handle invalid dates

    # Filter data for days where the temperature is between 75 and 80 degrees
    vacation_days = data[(data['MAX TEMP'] >= 75) & (data['MAX TEMP'] <= 80)].
↪copy()

    if vacation_days.empty:
        raise ValueError("No valid days found with temperatures between 75 and␣
↪80 degrees.")

    # Add a count column for each valid day
    vacation_days['Count'] = 1

    # Set the 'Date' as the index and resample by day to ensure continuous data
    vacation_days.set_index('Date', inplace=True)
    vacation_days = vacation_days.resample('D').sum().fillna(0)

    # Use a rolling window to find the best start date for a 30-day vacation
    rolling_counts = vacation_days['Count'].rolling(window=30).sum()

    if rolling_counts.isna().all():
        raise ValueError("All rolling window values are NaN. Check data␣
↪continuity or resampling logic.")

    best_start_day = rolling_counts.idxmax()

    # Return only the month and day, excluding the year
    return best_start_day.strftime("%B %d")

# Call the function and print the result
try:
    best_day = best_vacation_day(data)
    print(f"The best day to start a 30-day vacation is: {best_day}")
except Exception as e:
    print(f"Error: {e}")
```

```
The best day to start a 30-day vacation is: January 31
```

[ ]:

This notebook was converted to PDF with convert.ploomber.io