climate-change-impact-prediction

April 1, 2024

[33]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

```
/kaggle/input/global-surface-temperatures/sh_temps.csv
/kaggle/input/global-surface-temperatures/nh_temps.csv
/kaggle/input/global-surface-temperatures/zonann_temps.csv
/kaggle/input/global-surface-temperatures/global_temps.csv
```

[34]:
```
import warnings
warnings.filterwarnings('ignore')
```

1 About Data

The data comes from the NASA GISS Surface Temperature Analysis (GISTEMP v4). This datasets are tables of global and hemispheric monthly means and zonal annual means. They combine land-surface, air and sea-surface water temperature anomalies (Land-Ocean Temperature Index, L-OTI).

The values in the tables are deviations from the corresponding 1951-1980 means. In this notebook, I will focus on the data of global temperatures.

1.1 Loading Libraries & Data

[35]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[36]:
```
df = pd.read_csv('/kaggle/input/global-surface-temperatures/global_temps.csv')
df.head()
```

[36]:
```
   Year   Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov  \
0  1880 -0.19 -0.25 -0.09 -0.17 -0.10 -0.21 -0.18 -0.11 -0.15 -0.24 -0.22
1  1881 -0.20 -0.15  0.03  0.05  0.05 -0.19  0.00 -0.04 -0.16 -0.22 -0.19
2  1882  0.16  0.13  0.04 -0.16 -0.14 -0.22 -0.17 -0.08 -0.15 -0.24 -0.17
3  1883 -0.30 -0.37 -0.13 -0.19 -0.18 -0.08 -0.08 -0.14 -0.23 -0.12 -0.24
4  1884 -0.13 -0.09 -0.37 -0.40 -0.34 -0.35 -0.31 -0.28 -0.28 -0.25 -0.34

    Dec   J-D   D-N   DJF   MAM   JJA   SON
0 -0.18 -0.17   NaN   NaN -0.12 -0.17 -0.20
1 -0.08 -0.09 -0.10 -0.18  0.04 -0.08 -0.19
2 -0.36 -0.11 -0.09  0.07 -0.09 -0.16 -0.19
3 -0.11 -0.18 -0.20 -0.34 -0.17 -0.10 -0.20
4 -0.31 -0.29 -0.27 -0.11 -0.37 -0.32 -0.29
```

1.2 Initial Data Observation

[37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 19 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Year    144 non-null    int64
 1   Jan     144 non-null    float64
 2   Feb     144 non-null    float64
 3   Mar     144 non-null    float64
 4   Apr     144 non-null    float64
 5   May     144 non-null    float64
 6   Jun     143 non-null    float64
 7   Jul     143 non-null    float64
 8   Aug     143 non-null    float64
 9   Sep     143 non-null    float64
 10  Oct     143 non-null    float64
```

```
 11  Nov     143 non-null    float64
 12  Dec     143 non-null    float64
 13  J-D     143 non-null    float64
 14  D-N     142 non-null    float64
 15  DJF     143 non-null    float64
 16  MAM     144 non-null    float64
 17  JJA     143 non-null    float64
 18  SON     143 non-null    float64
dtypes: float64(18), int64(1)
memory usage: 21.5 KB
```

[38]:
```
# Check for duplicated rows
dupes = df.duplicated()
dupes.sum()
```

[38]:
```
0
```

[39]:
```
# Summary Stats
df.describe()
```

[39]:
```
              Year         Jan         Feb         Mar         Apr  \
count   144.000000  144.000000  144.000000  144.000000  144.000000
mean   1951.500000    0.063333    0.070903    0.088889    0.063681
std      41.713307    0.423598    0.428513    0.433790    0.396609
min    1880.000000   -0.810000   -0.630000   -0.630000   -0.580000
25%    1915.750000   -0.240000   -0.240000   -0.222500   -0.250000
50%    1951.500000   -0.015000   -0.040000    0.015000   -0.025000
75%    1987.250000    0.310000    0.382500    0.322500    0.282500
max    2023.000000    1.180000    1.370000    1.360000    1.130000

              May         Jun         Jul         Aug         Sep         Oct  \
count  144.000000  143.000000  143.000000  143.000000  143.000000  143.000000
mean     0.052917    0.033147    0.055874    0.054406    0.058182    0.084196
std      0.377894    0.367363    0.347531    0.363304    0.360199    0.369290
min     -0.550000   -0.520000   -0.510000   -0.550000   -0.580000   -0.580000
25%     -0.240000   -0.250000   -0.190000   -0.220000   -0.190000   -0.200000
50%     -0.040000   -0.050000   -0.030000   -0.050000   -0.060000    0.010000
75%      0.272500    0.240000    0.235000    0.235000    0.240000    0.245000
max      1.020000    0.930000    0.940000    1.020000    0.990000    1.090000

              Nov         Dec         J-D         D-N         DJF         MAM  \
count  143.000000  143.000000  143.000000  142.000000  143.000000  144.000000
mean     0.077762    0.051818    0.060210    0.060775    0.063566    0.068542
std      0.376197    0.393168    0.369845    0.370719    0.404956    0.398376
min     -0.550000   -0.820000   -0.480000   -0.490000   -0.670000   -0.580000
25%     -0.175000   -0.220000   -0.200000   -0.210000   -0.225000   -0.252500
50%      0.020000   -0.040000   -0.060000   -0.055000   -0.020000   -0.025000
```

```
75%      0.230000    0.305000    0.265000    0.277500    0.315000    0.310000
max      1.110000    1.160000    1.020000    1.040000    1.240000    1.140000

              JJA         SON
count  143.000000  143.000000
mean     0.047692    0.072867
std      0.355535    0.363067
min     -0.500000   -0.520000
25%     -0.215000   -0.190000
50%     -0.050000   -0.010000
75%      0.235000    0.240000
max      0.940000    1.000000
```

1.3 Check and Handle Missing values

[40]:
```python
df.isnull().sum()
```

[40]:
```
Year     0
Jan      0
Feb      0
Mar      0
Apr      0
May      0
Jun      1
Jul      1
Aug      1
Sep      1
Oct      1
Nov      1
Dec      1
J-D      1
D-N      2
DJF      1
MAM      0
JJA      1
SON      1
dtype: int64
```

[41]:
```python
# Handle missing data points with Linear Interpolation
for column in df.columns:
    if df[column].isnull().any():
        df[column] = df[column].interpolate(method='linear')
print(df.isnull().sum())
```

```
Year     0
Jan      0
Feb      0
```

```
Mar     0
Apr     0
May     0
Jun     0
Jul     0
Aug     0
Sep     0
Oct     0
Nov     0
Dec     0
J-D     0
D-N     1
DJF     1
MAM     0
JJA     0
SON     0
dtype: int64
```

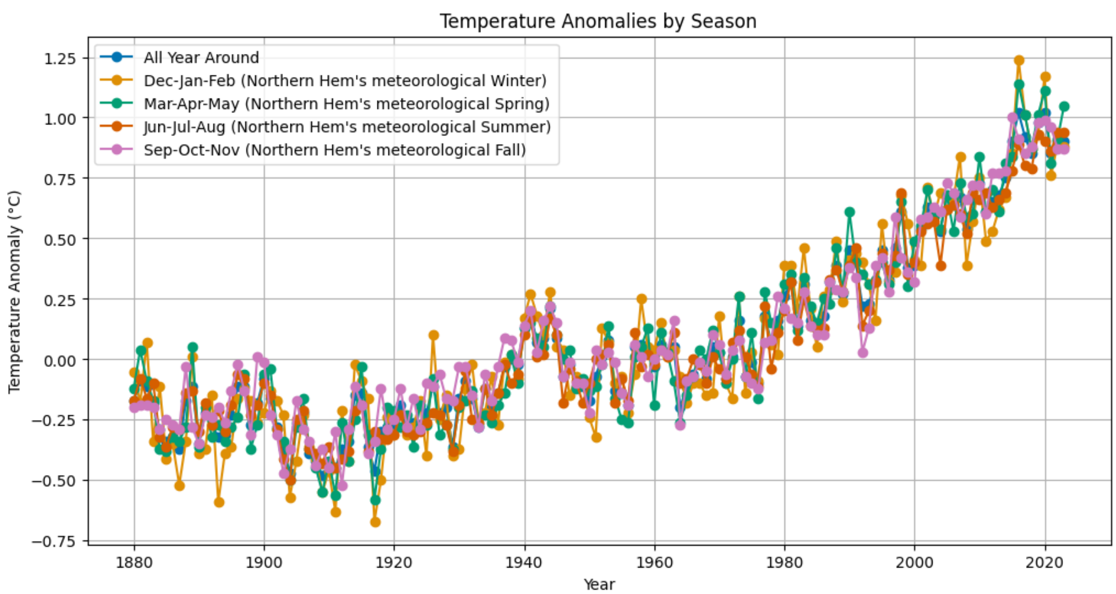
[42]:

```python
# Handle missing data points for first rows
# Filling NaN values in the first row with the mean of the 2nd and 3rd rows

if df.iloc[0].isnull().any():
    mean_val = df.iloc[1:3].mean()
    df.iloc[0] = df.iloc[0].fillna(mean_val)

print(df.isnull().sum())
```

```
Year    0
Jan     0
Feb     0
Mar     0
Apr     0
May     0
Jun     0
Jul     0
Aug     0
Sep     0
Oct     0
Nov     0
Dec     0
J-D     0
D-N     0
DJF     0
MAM     0
JJA     0
SON     0
dtype: int64
```

2 Visual Exploration

[43]:
```python
# Mapping of columns to season names for the legend
season_info = {
    'J-D': 'All Year Around',
    'DJF': 'Dec-Jan-Feb (Northern Hem\'s meteorological Winter)',
    'MAM': 'Mar-Apr-May (Northern Hem\'s meteorological Spring)',
    'JJA': 'Jun-Jul-Aug (Northern Hem\'s meteorological Summer)',
    'SON': 'Sep-Oct-Nov (Northern Hem\'s meteorological Fall)'
}

# Set the seaborn color palette so that each season gets a distinct color
sns.set_palette("colorblind")

# Plot each season on the same graph
plt.figure(figsize=(12,6))
for col, title in season_info.items():
    plt.plot(df['Year'], df[col], marker='o', label=title)

plt.title('Temperature Anomalies by Season')
plt.xlabel('Year')
plt.ylabel('Temperature Anomaly (°C)')
plt.grid(True)
plt.legend()
plt.show()
```



2.1 Finding & Visualizing Outliers

2.1.1 IQR Method

[44]:
```
Q1 = df['J-D'].quantile(0.25)
Q3 = df['J-D'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = df[(df['J-D'] < lower_bound) | (df['J-D'] > upper_bound)]
print("Outliers:\n", outliers)
```

```
Outliers:
      Year   Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov  \
136  2016  1.17  1.37  1.36  1.10  0.95  0.80  0.84  1.02  0.91  0.89  0.92
140  2020  1.18  1.25  1.17  1.13  1.02  0.92  0.90  0.88  0.99  0.89  1.11

      Dec   J-D   D-N   DJF   MAM   JJA   SON
136  0.87  1.02  1.04  1.24  1.14  0.89  0.91
140  0.81  1.02  1.04  1.17  1.11  0.90  0.99
```

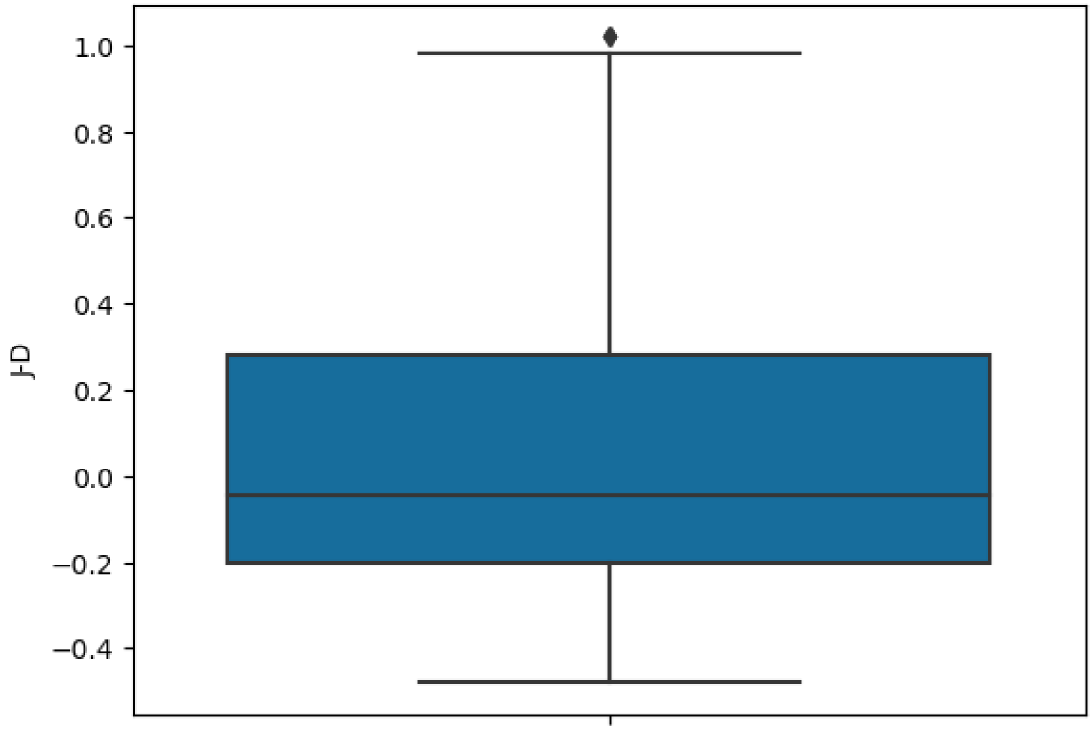
2.1.2 Z-Score Method

[45]:
```
from scipy.stats import zscore

z_score = zscore(df['SON'])
print(z_score)
threshold = 3 # 3 SDs away from the mean

ol = df[abs(z_score) > threshold]
print(ol)
```

```
0     -0.759493
1     -0.732213
2     -0.732213
3     -0.759493
4     -1.005017
         …
139    2.459592
140    2.486872
141    2.405031
142    2.159508
143    2.159508
Name: SON, Length: 144, dtype: float64
Empty DataFrame
```

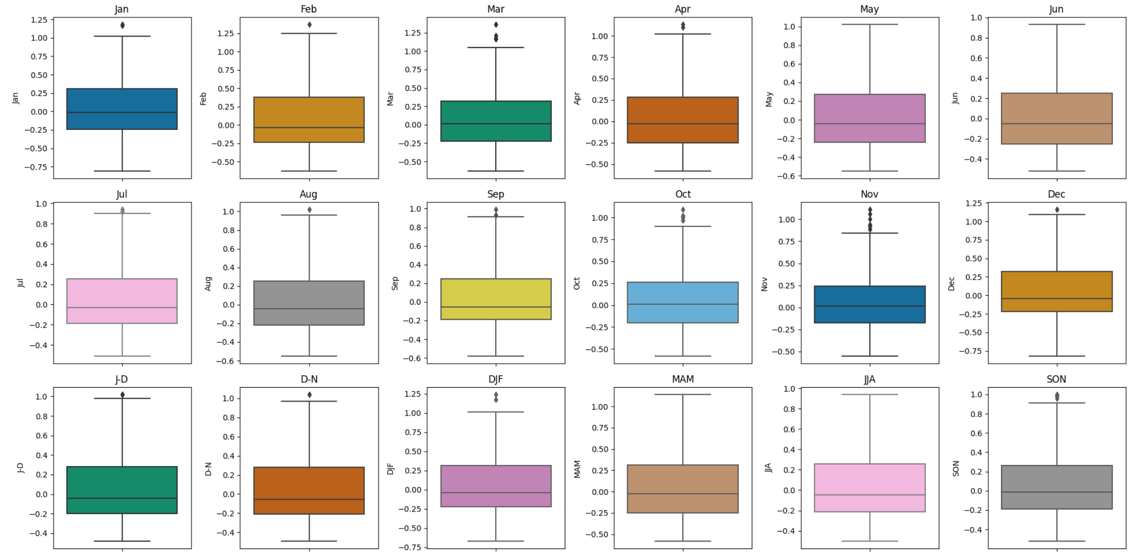
```
Columns: [Year, Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec, J-D,
D-N, DJF, MAM, JJA, SON]
Index: []
```

[46]:
```
sns.boxplot(y=df['J-D'], color=sns.color_palette()[0])

plt.show()
```



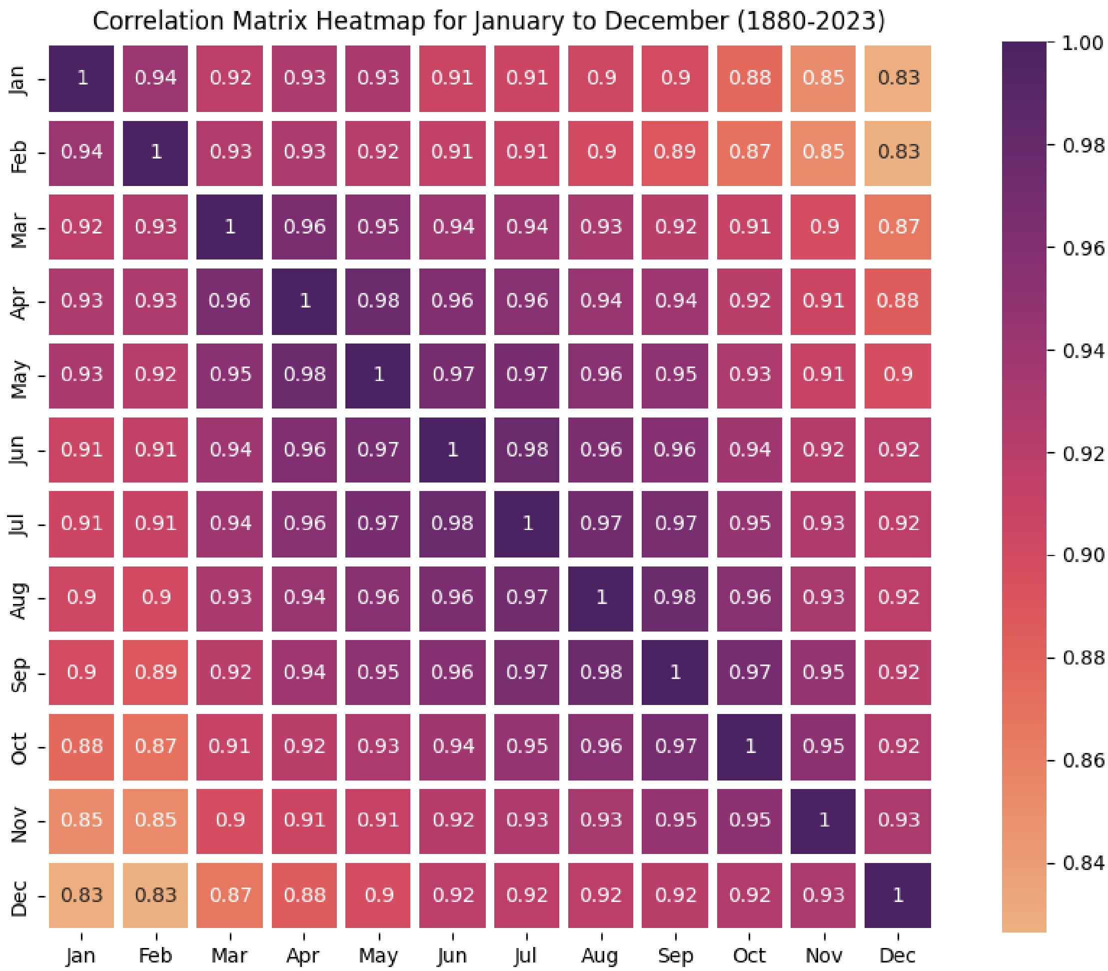
[47]:
```
# Set the color palette to 'colorblind'
palette = sns.color_palette("colorblind")

plt.figure(figsize=(20,10))

# Loop through each column (excluding 'Year')
for i, column in enumerate(df.columns.drop('Year')):
    # Create a subplot for each column
    plt.subplot(3, 6, i+1)
    # Use modulo to cycle through the colorblind-friendly palette
    color = palette[i % len(palette)]
    sns.boxplot(y=df[column], color=color)
    plt.title(column)
    plt.tight_layout()
```

```
plt.show()
```

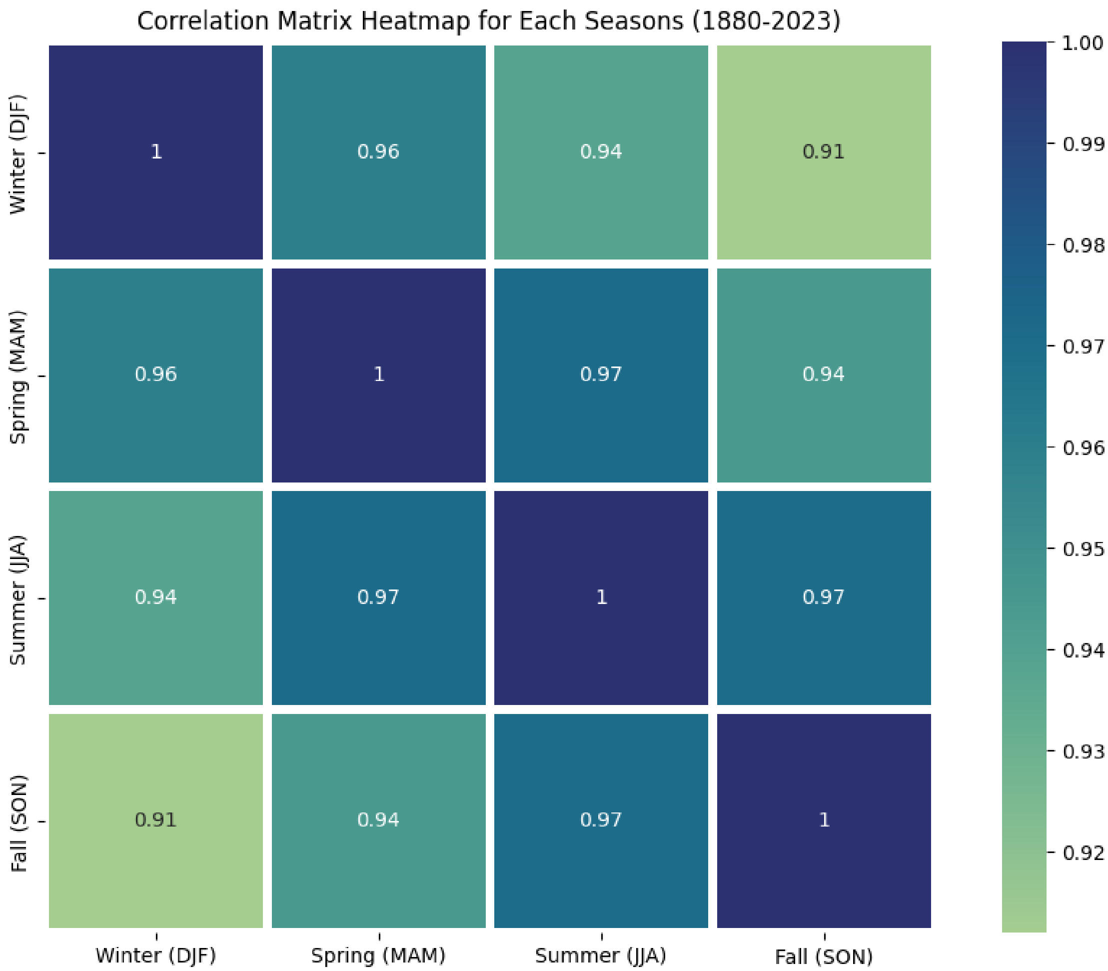


2.2 Correlation Heatmap

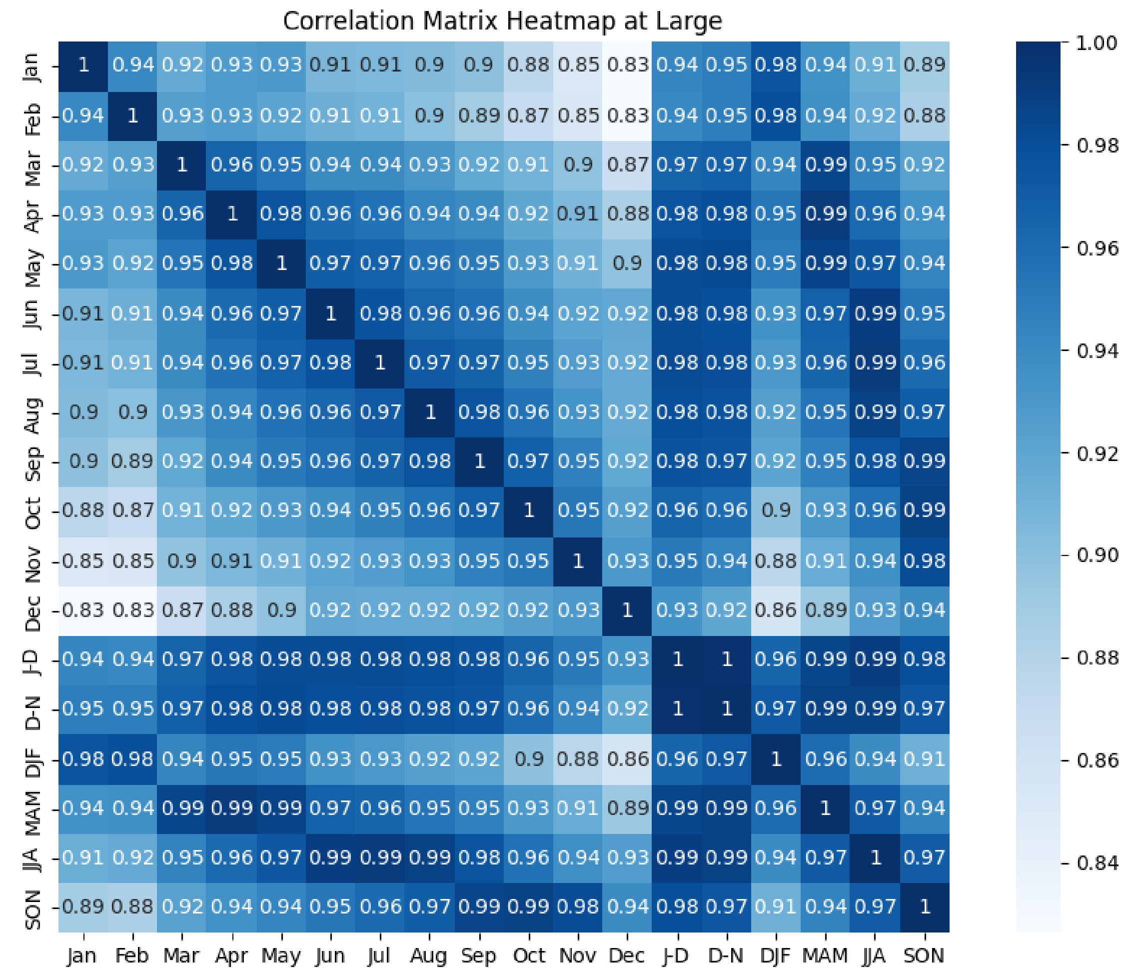
[48]:
```
months_columns = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
df_months = df[months_columns]

corr_matrix = df_months.corr()

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix,
            annot = True,
            cmap='flare',
            linewidth=3,
            square=True)
plt.title('Correlation Matrix Heatmap for January to December (1880-2023)')
plt.show()
```




```
[49]: season_columns = ['DJF', 'MAM', 'JJA', 'SON'] # winter, spring, summer, fall
df_seasons = df[season_columns]

corr_matrix_seasons = df_seasons.corr()
season_labels = ['Winter (DJF)', 'Spring (MAM)', 'Summer (JJA)', 'Fall (SON)']

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix_seasons,
            annot = True,
            cmap='crest',
            linewidth=3,
            xticklabels=season_labels,
            yticklabels=season_labels,
            square=True)
plt.title('Correlation Matrix Heatmap for Each Seasons (1880-2023)')
plt.show()
```



[50]:
```
new_df = df.drop(columns=['Year'])
corr_matrix2 = new_df.corr()

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix2,
            annot = True,
            cmap='Blues',
            square=True)
plt.title('Correlation Matrix Heatmap at Large')
plt.show()
```



2.2.1 Observation from Heatmaps

1. High Correlation Among Months: There is a high degree of correlation amonth individual motnhs, but this is expected in time series data especially for climatic or environmental data, where adjacent months often have similar conditions or patterns.

2. Seasonal Correlations: The seasonal indicators (DJF, MAM, JJA, SON) also show high correlation with the individual months that comprise each season. This is also expected, as these are aggregate measure of the months they represent.

3. Multicollinearity Consideration: In the context of linear models, multicollinearity can be problematic as it inflates the variance of the coefficient estimates and makes the model sensitive to changes in model specification. High correlations among variables indicates multicollinearity. If I will use there variables in a predictive model, I will need to address this issue. (Dimensional Reduction, Variable Selection, Regularization)

Correlation does NOT imply causation!

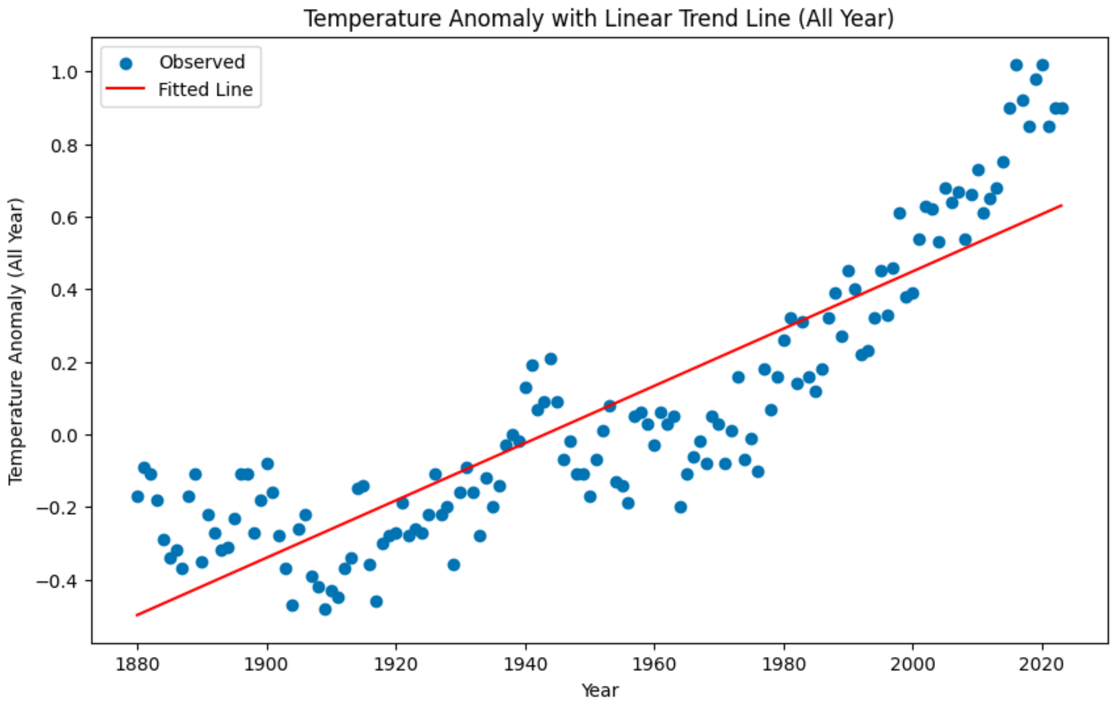
3 Trend Analysis

3.1 Rate of Change

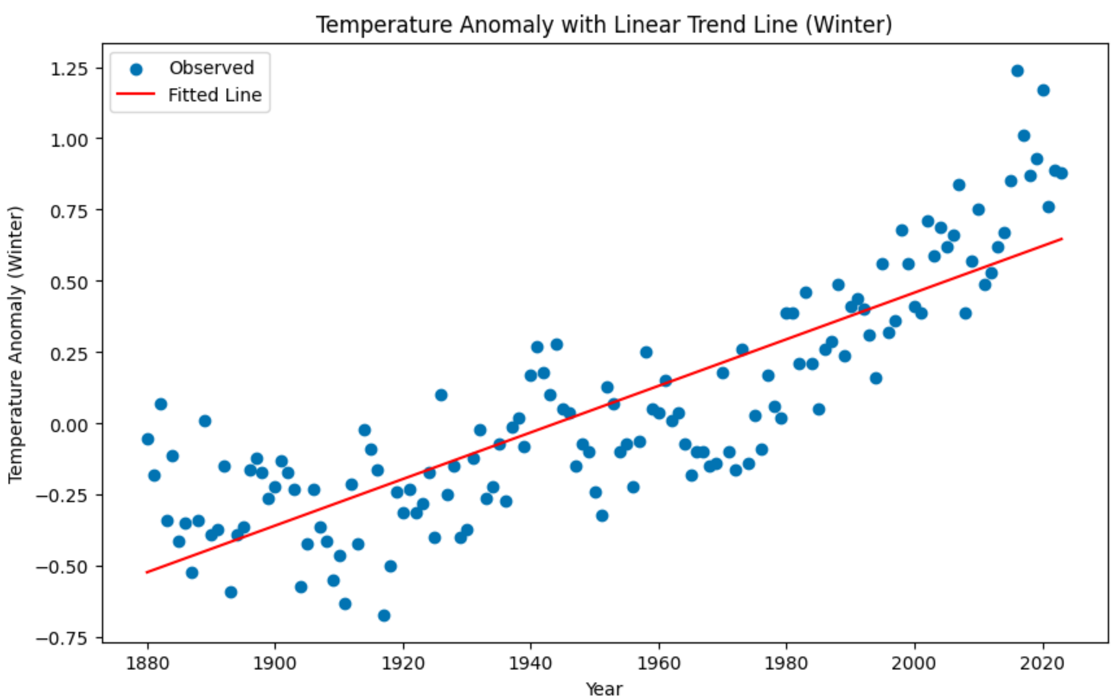
From visual inspection of time series plots, it shows a strong trend of increase in global surface temperature. In this section, I will run the trend analysis using various statistical methods to analyze further into the phenomenon.

[19]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

# Mapping of columns to season names
season_names = {
    'J-D': 'All Year',
    'DJF': 'Winter',
    'MAM': 'Spring',
    'JJA': 'Summer',
    'SON': 'Fall'
}

X = df['Year'].values.reshape(-1, 1)

for col, season in season_names.items():
    y = df[col]

    model = LinearRegression().fit(X, y)
    print(f"\nTrend Analysis for {col}:")
    print(f"Slope: {model.coef_[0]}")
    print(f"Intercept: {model.intercept_}")

    y_pred = model.predict(X)

    plt.figure(figsize=(10, 6))
    plt.scatter(df['Year'], y, label='Observed')
    plt.plot(df['Year'], y_pred, color='red', label='Fitted Line')
    plt.xlabel('Year')
    plt.ylabel(f'Temperature Anomaly ({season})')
    plt.title(f'Temperature Anomaly with Linear Trend Line ({season})')
    plt.legend()
    plt.show()
```

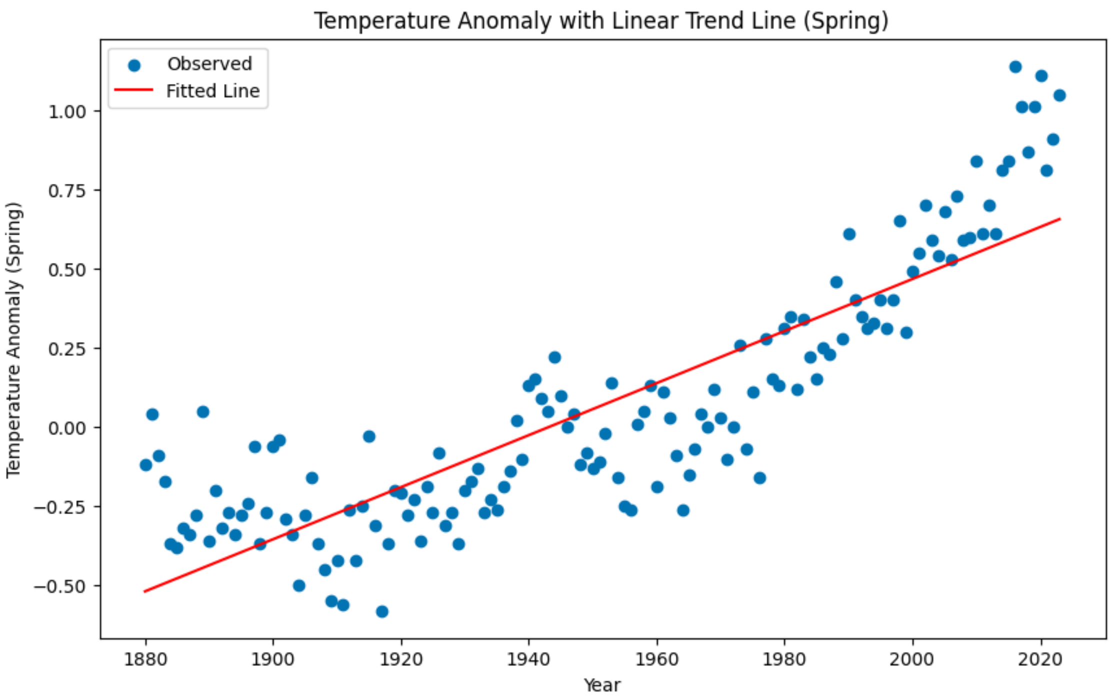
```
Trend Analysis for J-D:
Slope: 0.007891266779197815
Intercept: -15.333765452937868
```



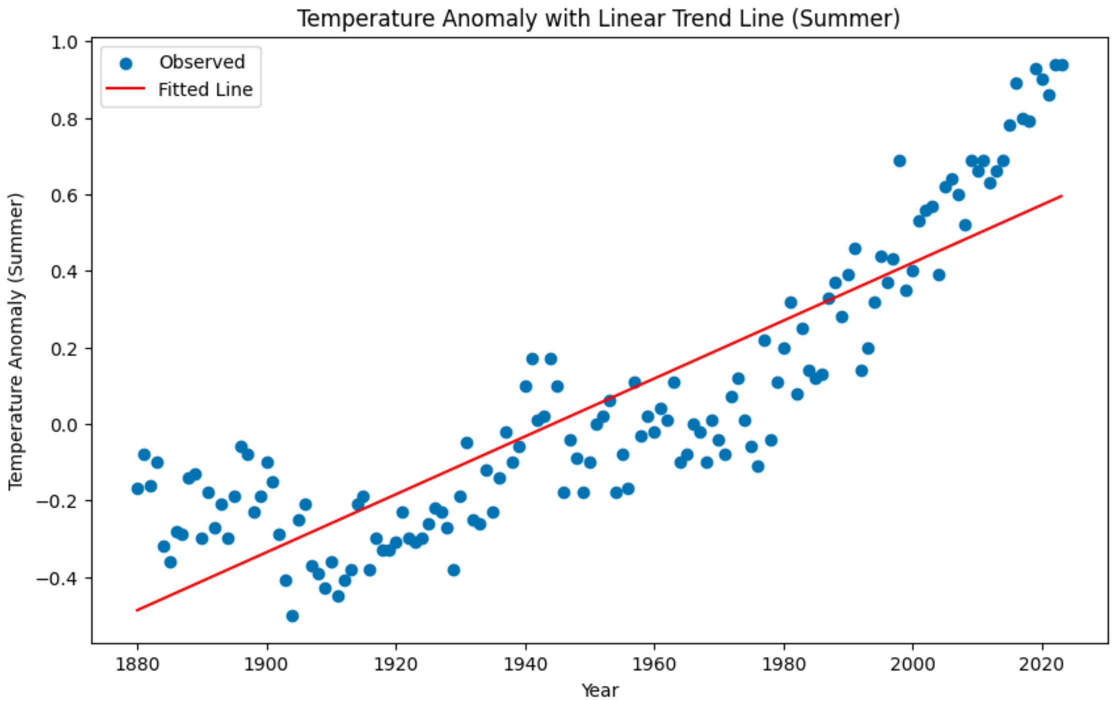

```
Trend Analysis for DJF:
Slope: 0.008169911984567157
Intercept: -15.880840182327251
```



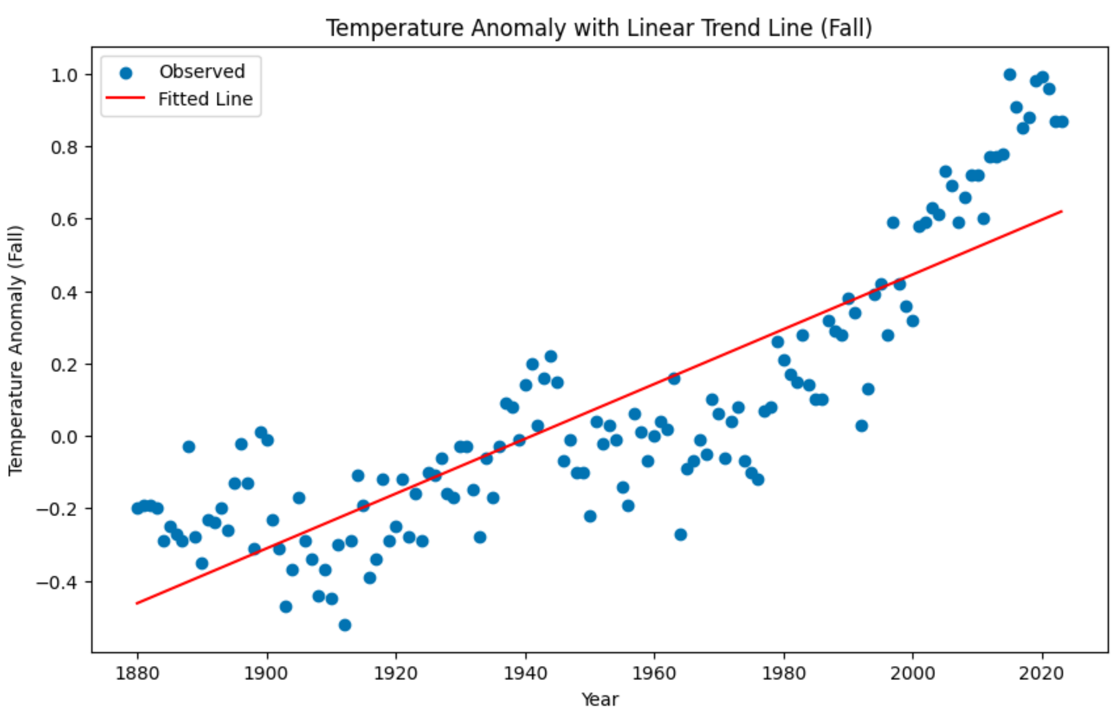
```
Trend Analysis for MAM:
Slope: 0.008217888433405677
Intercept: -15.968667611124511
```




```
Trend Analysis for JJA:
Slope: 0.007568041154248054
Intercept: -14.715143423626188
```

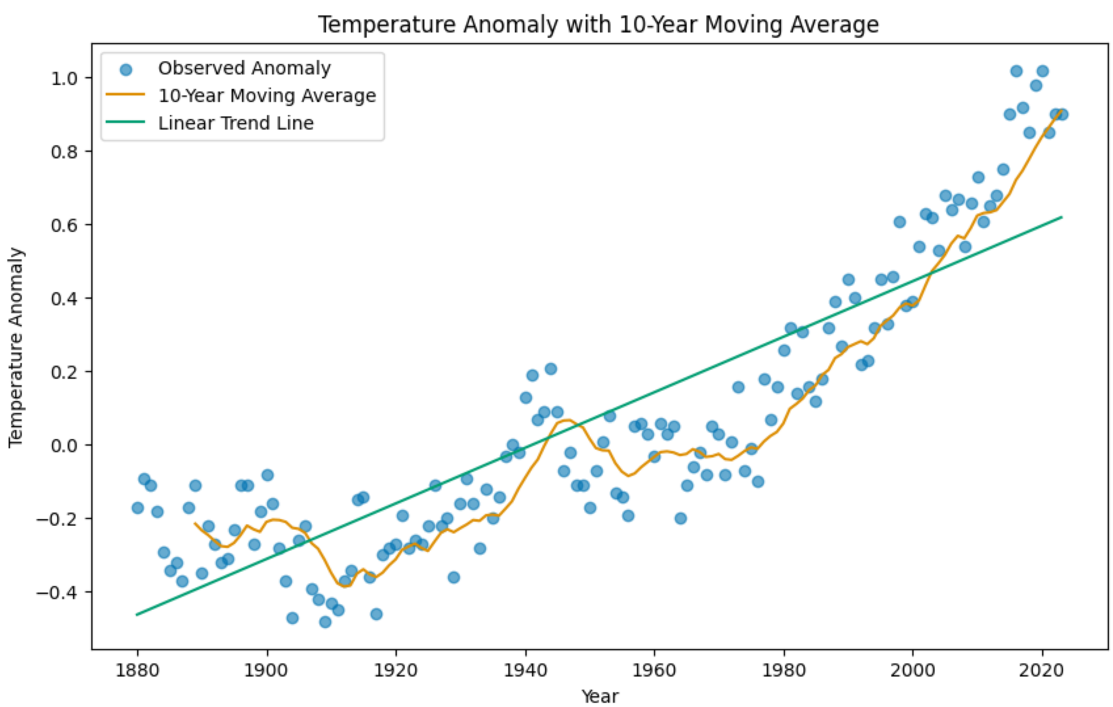



```
Trend Analysis for SON:
Slope: 0.007563278675347641
Intercept: -14.681335557163145
```



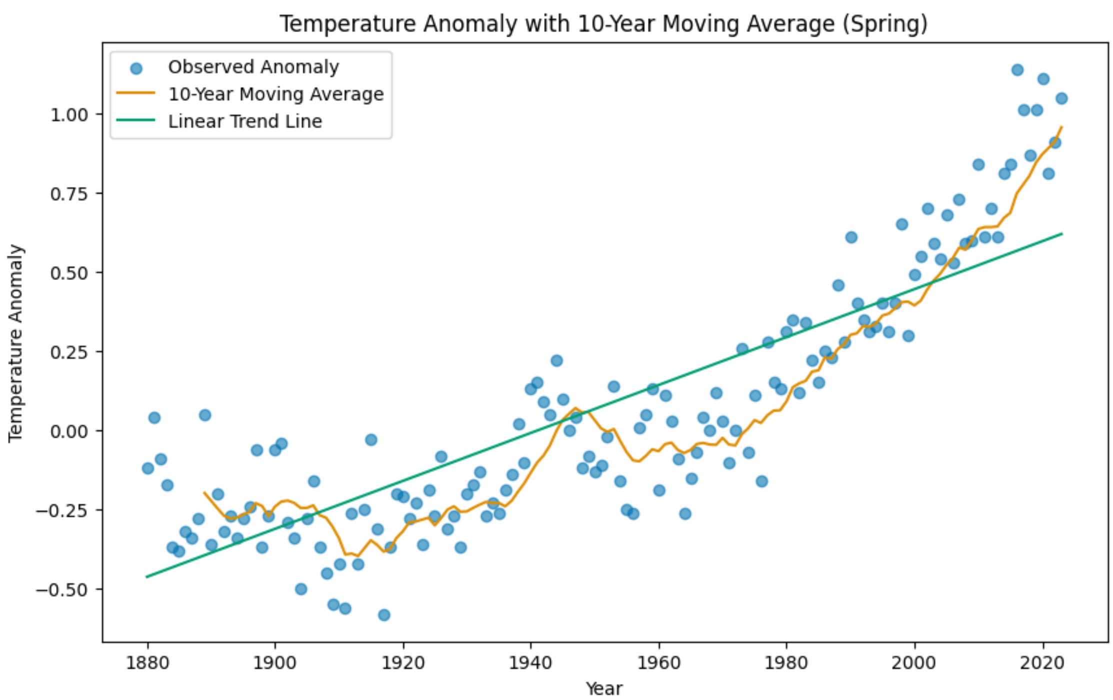
3.2 Moving Average Method

```
[20]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

palette = sns.color_palette("colorblind", 3)
window_size = 10 # try different size of window
df['Moving_Avg'] = df['J-D'].rolling(window=window_size).mean()

plt.figure(figsize=(10,6))
plt.scatter(df['Year'], df['J-D'], alpha=0.6, label='Observed Anomaly')
plt.plot(df['Year'], df['Moving_Avg'], color=palette[1],␣
 ↪label=f'{window_size}-Year Moving Average')
plt.plot(df['Year'], y_pred, color=palette[2], label='Linear Trend Line')
plt.xlabel('Year')
plt.ylabel('Temperature Anomaly')
plt.title(f'Temperature Anomaly with {window_size}-Year Moving Average')
plt.legend()
plt.show()
```



[21]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

palette = sns.color_palette("colorblind", 3)
window_size = 10 # try different size of window
df['Moving_Avg_S'] = df['MAM'].rolling(window=window_size).mean()

plt.figure(figsize=(10,6))
plt.scatter(df['Year'], df['MAM'], alpha=0.6, label='Observed Anomaly')
plt.plot(df['Year'], df['Moving_Avg_S'], color=palette[1],␣
 ↪label=f'{window_size}-Year Moving Average')
plt.plot(df['Year'], y_pred, color=palette[2], label='Linear Trend Line')
plt.xlabel('Year')
plt.ylabel('Temperature Anomaly')
plt.title(f'Temperature Anomaly with {window_size}-Year Moving Average␣
 ↪(Spring)')
plt.legend()
plt.show()
```



4 Statistical Analysis

4.1 ANOVA

One-Way ANOVA is suitable when:

1. Group Comparison: there are multiple groups to compare
2. One Factor: groups that can be categorized in one factor
3. Independence of observations: under assumptions that the observations in each group are independent of each other
4. Normality: under assumptions that residuals are normally distributed and variances of the groups are equal.

Normality Test prior to ANOVA

4.2 Change Point Analysis

To identify points in time where the statistical properties of a time series change

[ ]:
```
!pip install ruptures
```

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,
status=None)) after connection broken by
'NewConnectionError('<pip._vendor.urllib3.connection.HTTPSConnection object at
0x7e5b36401e10>: Failed to establish a new connection: [Errno -3] Temporary
failure in name resolution')': /simple/ruptures/
```

4.2.1 Change Point Detection for All-Year

[51]:
```
# Reset the dataset
df = pd.read_csv('/kaggle/input/global-surface-temperatures/global_temps.csv')
for column in df.columns:
    if df[column].isnull().any():
        df[column] = df[column].interpolate(method='linear')
if df.iloc[0].isnull().any():
    mean_val = df.iloc[1:3].mean()
    df.iloc[0] = df.iloc[0].fillna(mean_val)
# Check dataset
df.head()
```

[51]:
```
   Year   Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov  \
0  1880 -0.19 -0.25 -0.09 -0.17 -0.10 -0.21 -0.18 -0.11 -0.15 -0.24 -0.22
1  1881 -0.20 -0.15  0.03  0.05  0.05 -0.19  0.00 -0.04 -0.16 -0.22 -0.19
2  1882  0.16  0.13  0.04 -0.16 -0.14 -0.22 -0.17 -0.08 -0.15 -0.24 -0.17
3  1883 -0.30 -0.37 -0.13 -0.19 -0.18 -0.08 -0.08 -0.14 -0.23 -0.12 -0.24
4  1884 -0.13 -0.09 -0.37 -0.40 -0.34 -0.35 -0.31 -0.28 -0.28 -0.25 -0.34
```

```
    Dec   J-D    D-N    DJF   MAM   JJA   SON
0 -0.18 -0.17 -0.095 -0.055 -0.12 -0.17 -0.20
1 -0.08 -0.09 -0.100 -0.180  0.04 -0.08 -0.19
2 -0.36 -0.11 -0.090  0.070 -0.09 -0.16 -0.19
3 -0.11 -0.18 -0.200 -0.340 -0.17 -0.10 -0.20
4 -0.31 -0.29 -0.270 -0.110 -0.37 -0.32 -0.29
```

[ ]:
```
import ruptures as rpt

points = df["J-D"].values
print(points)

# Search, binary segmentation
algo = rpt.Binseg(model="rank").fit(points)
result = algo.predict(n_bkps=5) # number of breakpoints to detect
print(result)

# Plotting
plt.figure(figsize=(10,6))
plt.plot(df['Year'], points, label='J-D')
palette = sns.color_palette("colorblind")

# Addinv vertical lines for change points
for i, cp in enumerate(result[:-1]):
    year = df['Year'].iloc[cp]
    color = palette[i+1]
    plt.axvline(x=year, color=color, linestyle='--', label=f'Change Points␣
 ↪{i+1}')
    plt.text(year, max(points), f'{year}', color=color,␣
 ↪verticalalignment='top', horizontalalignment='right')
    print(f"Change Point {i+1}: Year {year}, Position {cp}")

plt.xlabel('Year')
plt.title('Change Point Detection for All-Year')
plt.legend()
plt.show()
```

The model to choose from:

l1, l2, rbf, linear, normal, ar, rank, mahalanobis

Interpretation:

Change Point 1 - Year 1930: This point could indicate the beginning of a significant warming phase. It could be due to the early impacts of industrialization.

Change Point 2 - Year 1940: A decade later, this point suggests another shift in the temperature trend. The period around 1940 marks the beginning of short period of cooling phase, followed by the rapid increase. It could be attributed to factors like natural climate variability.

Change Point 3 - Year 1945: The mid-1940s are significant for historical reasons, including the end of World War II, which could have influenced atmospheric conditions through industrial activities during the War time.

Change Point 4 - Year 1980: The early 1980s marks the beginning of the Global Warming, and the warming trend was observed towards the end of the 20th century and continuing well into the 21st century.

Change Poing 5 - Year 2000: The critical change point came the Year 2000. The early 21st century has seen some of the warmest years on record across the world, suggesting an acceleration of climate change impacts.

4.2.2 Change Point Detection for Each Season

[53]:
```
import ruptures as rpt

season_names = {'DJF': 'Winter', 'MAM': 'Spring', 'JJA': 'Summer', 'SON':␣
↪'Fall'}
for abb, season in season_names.items():
    points = df[abb].values
    print(points)

    # Search, binary segmentation
    algo = rpt.Binseg(model="rank").fit(points)
    result = algo.predict(n_bkps=5) # number of breakpoints to detect
    print(result)

    # Plotting
    plt.figure(figsize=(10,6))
    plt.plot(df['Year'], points, label=f'{season}')
    palette = sns.color_palette("colorblind")

    # Addinv vertical lines for change points
    for i, cp in enumerate(result[:-1]):
        year = df['Year'].iloc[cp]
        color = palette[i+1]
        plt.axvline(x=year, color=color, linestyle='--', label=f'Change Points␣
↪{i+1}')
        y_pos = max(points)
        plt.text(year, y_pos, f'{year}', color=color, verticalalignment='top',␣
↪horizontalalignment='right')
        print(f"Change Point {i+1}: Year {year}, Position {cp}, Y-Position␣
↪{y_pos}")

    plt.xlabel('Year')
    plt.title(f'Change Point Detection for {season}')
    plt.legend()
```

```
plt.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[53], line 1
----> 1 import ruptures as rpt
      3 season_names = {'DJF': 'Winter', 'MAM': 'Spring', 'JJA': 'Summer', 'SON':
 ↪ 'Fall'}
      4 for abb, season in season_names.items():

ModuleNotFoundError: No module named 'ruptures'
```