Final Project CMSC6950 Fall 2025

Mona Hejazi

201996350

Introduction

Project Aim

This project investigates a one-year weather dataset for Tehran, with a primary focus on identifying extreme values in temperature and precipitation. Through statistical analysis, the study aims to uncover insights into Tehran's climatic trends and extreme weather events over the observed period.

Background

Weather data serves as a crucial resource for understanding climatic patterns, forecasting, and decision-making in various sectors. By analyzing meteorological parameters such as temperature, precipitation, and wind patterns, we can identify anomalies, explore correlations, and assess their potential impacts. This project explores the statistical distribution of key weather variables for Tehran, delving into both summary statistics and outlier detection.

Dataset Overview:

The dataset contains daily weather records for Tehran from **October 1, 2023**, to **September 30, 2024**. It comprises 366 entries, representing each day of the year, and includes 11 meteorological variables:

- `date`: Date of the weather record
- `tavg`: Average temperature (°C)
- `tmin`: Minimum temperature (°C)
- `tmax`: Maximum temperature (°C)
- `prcp`: Precipitation (mm)
- `snow`: Snowfall (cm) — with limited data
- `wdir`: Wind direction (degrees)
- `wspd`: Wind speed (m/s)
- `wpgt`: Wind gust (m/s) — with missing data
- `pres`: Atmospheric pressure (hPa)
- `tsun`: Sunshine duration (hours) — missing data

Objectives

The project’s primary objectives include:

1) Analyzing daily averages, minimum and maximum temperatures, and precipitation trends across the year. 1) Identifying and visualizing extreme values in temperature and precipitation using statistical methods. 1) Exploring correlations between key variables to understand weather dynamics in Tehran.

Hypothesis

1) The average temperature, wind speed, and precipitation do not follow a normal distribution. 1) There is no linear correlation between average temperature and precipitation. 1) A linear correlation exists between wind speed and precipitation. 1) Removing outliers enhances the reliability of statistical analysis.

Source :

https://meteostat.net/en/place/ir/tehran?s=40754&t=2023-10-01/2024-09-30

Data Processing and Analysis of Tehran's Weather Data

Overview of Data Inspection and Handling Missing Values

The project commenced with an in-depth inspection of the dataset to understand its structure and identify potential issues such as missing data. Several columns, including snow, wpgt, and tsun, exhibited substantial amounts of missing values. Given their limited data availability, these columns were excluded from further analysis to maintain the integrity and relevance of the findings.

Summary Statistics

Key variables—`tavg` (average temperature), `tmin`(minimum temperature), `tmax` (maximum temperature), and `prcp` (precipitation)—were analyzed to compute summary statistics. These included metrics such as the mean, standard deviation, minimum, maximum, and interquartile range (25th, 50th, and 75th percentiles). The summary statistics served as a foundational step to understand data distributions, detect anomalies, and gain insights into the central tendencies and variability within the dataset.

Analyzing Extreme Temperature Values in Tehran

Workflow Overview

To examine Tehran's weather trends over one year, I developed a Python-based analytical pipeline. The workflow was initiated by importing and preprocessing the dataset from an Excel file. The date column was converted to a datetime format and set as the index to enable time-series operations. This setup facilitated a structured approach to analyzing temporal trends in weather data.

Identifying and Highlighting Extreme Values

A custom function was implemented to extract extreme temperature values, pinpointing:

1) The highest maximum temperature along with its corresponding date. 1) The lowest minimum temperature and its occurrence date. 1) These findings were crucial for understanding significant deviations in Tehran's weather patterns.

Visualization of Temperature Trends

To communicate insights effectively, a comprehensive visualization was created:

Daily Trends: The plot featured daily minimum, maximum, and average temperatures, with shaded areas representing the range between minimum and maximum values for enhanced clarity.

Extreme Highlights: Markers and annotations were used to emphasize extreme temperature points, ensuring they stood out within the broader trends.

Mean Temperature: A horizontal line denoting the annual mean temperature was added to provide contextual reference against daily values. The plot was further customized with:

1) Clear axis labels and a descriptive title. 1) An informative caption to contextualize the visualization within the project's objectives. 1) Enhanced aesthetics, ensuring alignment with professional documentation standards.

Output and Documentation

This systematic approach not only highlights extreme weather events but also underscores broader temperature patterns in Tehran, offering a valuable perspective for stakeholders interested in climate trends.

Double-click (or enter) to edit

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

def load_and_prepare_data(file_path):
    """
    Load the Excel file and preprocess the data.
    - Converts the 'date' column to datetime.
    - Sets 'date' as the index.

    Args:
        file_path (str): Path to the Excel file.

    Returns:
        pd.DataFrame: Preprocessed DataFrame.
    """
    data = pd.read_excel(file_path)
    data['date'] = pd.to_datetime(data['date'])
```

```
    data.set_index('date', inplace=True)
    return data

def find_extreme_values(data):
    """
    Find the extreme temperatures and their corresponding dates.

    Args:
        data (pd.DataFrame): DataFrame with temperature data.

    Returns:
        tuple: (max_temp, max_temp_date, min_temp, min_temp_date)
    """
    max_temp = data['tmax'].max()
    max_temp_date = data['tmax'].idxmax()
    min_temp = data['tmin'].min()
    min_temp_date = data['tmin'].idxmin()
    return max_temp, max_temp_date, min_temp, min_temp_date

def plot_temperature_trends(data, max_temp, max_temp_date, min_temp, min_temp_date, mean_tem
    """
    Plot temperature trends and highlight extreme values along with mean temperature.

    Args:
        data (pd.DataFrame): DataFrame with temperature data.
        max_temp (float): Maximum temperature value.
        max_temp_date (pd.Timestamp): Date of maximum temperature.
        min_temp (float): Minimum temperature value.
        min_temp_date (pd.Timestamp): Date of minimum temperature.
        mean_temp (float): Mean temperature value.
        save_path (str, optional): Path to save the plot as a PDF.
    """
    plt.figure(figsize=(16, 10))
    plt.rcParams.update({
        'font.size': 14,
        'axes.titlesize': 20,
        'axes.labelsize': 16,
        'legend.fontsize': 14,
        'xtick.labelsize': 12,
        'ytick.labelsize': 12
    })

    # Plotting temperatures
    plt.plot(data.index, data['tmin'], label='Min Temperature (°C)', color='blue', linestyle
    plt.plot(data.index, data['tmax'], label='Max Temperature (°C)', color='red', linestyle=
    plt.plot(data.index, data['tavg'], label='Average Temperature (°C)', color='orange', lin
    plt.fill_between(data.index, data['tmin'], data['tmax'], color='lightgray', alpha=0.3)

    # Highlight extreme values
    plt.scatter(max_temp_date, max_temp, color='red', label=f'Highest Max Temp ({max_temp}°C
    plt.scatter(min_temp_date, min_temp, color='blue', label=f'Lowest Min Temp ({min_temp}°C
```

```
# Annotating extreme values outside the plot with a box and arrow
plt.annotate(
    f'Max Temp: {max_temp}°C',
    (max_temp_date, max_temp),
    textcoords="offset points",
    xytext=(120, 30),  # Position outside the plot
    ha='center',
    color='red',
    fontsize=12,
    fontweight='bold',
    bbox=dict(facecolor='white', edgecolor='red', boxstyle='round,pad=1'),
    arrowprops=dict(arrowstyle="->", color='red')
)
plt.annotate(
    f'Min Temp: {min_temp}°C',
    (min_temp_date, min_temp),
    textcoords="offset points",
    xytext=(-50, -100),  # Position outside the plot
    ha='center',
    color='blue',
    fontsize=12,
    fontweight='bold',
    bbox=dict(facecolor='white', edgecolor='blue', boxstyle='round,pad=1'),
    arrowprops=dict(arrowstyle="->", color='blue')
)

# Adding mean temperature line
plt.axhline(y=mean_temp, color='green', linestyle='-', linewidth=2, label=f'Mean Temp ({

# Adding labels, title, and legend
plt.title('Temperature Trends in Tehran Over Time with Extremes Highlighted', fontweight
plt.xlabel('Month', fontweight='bold')
plt.ylabel('Temperature (°C)', fontweight='bold')
plt.legend(loc='upper left')
plt.grid()

# Change x-axis to month names
plt.xticks(ticks=data.index[::30], labels=data.index.month_name()[::30], rotation=45)

# Adding a caption
caption = (
    "Figure 1 : This plot illustrates the trends in minimum, maximum, and average temper
    "Highlighted are the extreme temperatures: the highest maximum temperature and the l
    "along with their corresponding dates. Shaded areas between the minimum and maximum 
    "represent the range of temperature variation on each day. The green horizontal line
)
plt.figtext(
    0.5, -0.1, caption,
    wrap=True, horizontalalignment='center', fontsize=14, color='black'
)
```

```
    plt.tight_layout()

# Main script
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'
data = load_and_prepare_data(file_path)
max_temp, max_temp_date, min_temp, min_temp_date = find_extreme_values(data)
mean_temp = data['tavg'].mean()  # Calculate the mean temperature
save_path = r'E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temp_extreme_Plot.pdf'

# Call the function to plot and save the figure
plot_temperature_trends(data, max_temp, max_temp_date, min_temp, min_temp_date, mean_temp, s

# Save the plot as a PDF if a save_path is provided
if save_path:
        plt.savefig(save_path, format='pdf', bbox_inches='tight')  # Ensure caption is inclu
        print(f"Plot saved as {save_path}")

plt.show()
```

Plot saved as E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temp_extreme_Plot.p

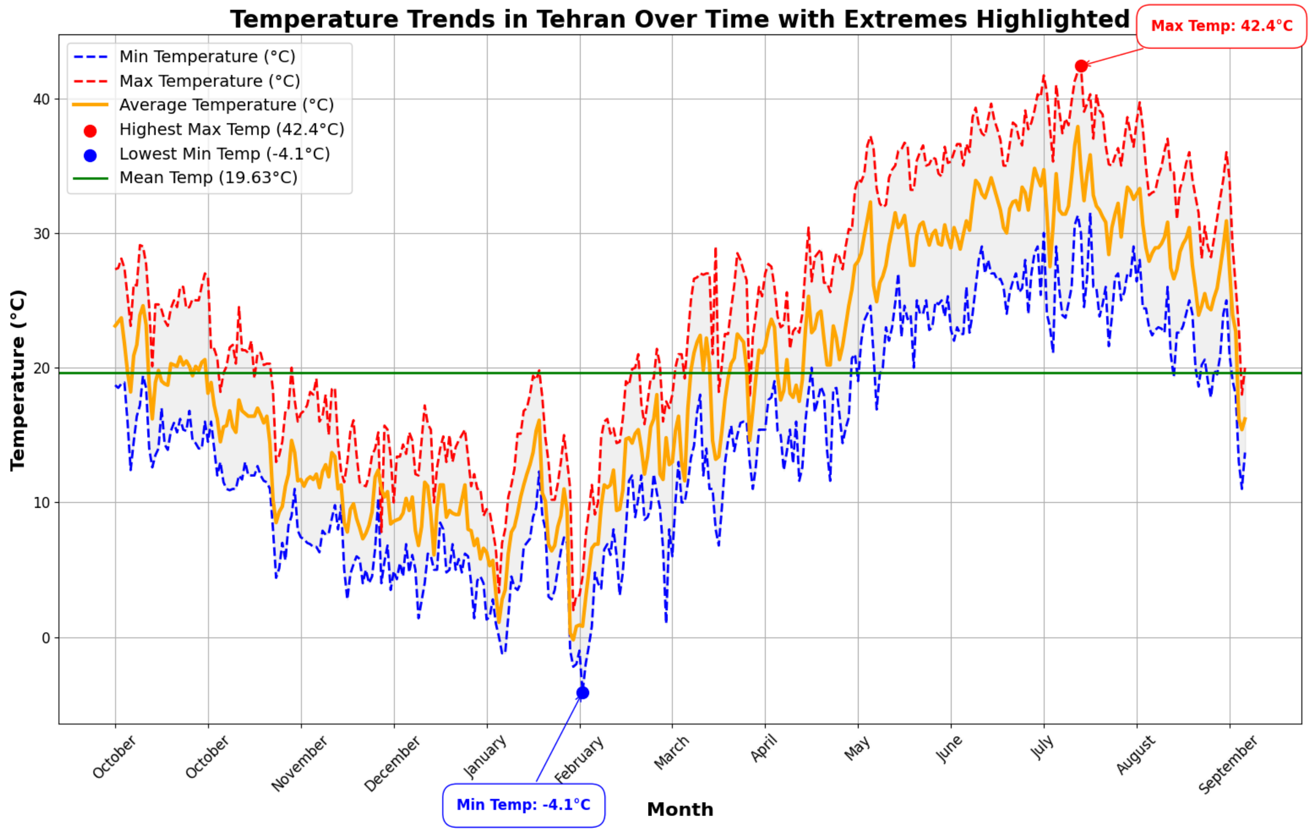


Figure 1 : This plot illustrates the trends in minimum, maximum, and average temperatures in Tehran over time. Highlighted are the extreme temperatures: the highest maximum temperature and the lowest minimum temperature, along with their corresponding dates. Shaded areas between the minimum and maximum temperatures represent the range of temperature variation on each day. The green horizontal line represents the mean temperature.

Temperature Analysis: Deviations from Historical Mean by ±1.5 Standard Deviations

Objective and Methodology

This analysis focuses on identifying temperature trends that deviate significantly from the historical mean, using a threshold of ±1.5 standard deviations (SD). The aim was to classify daily temperatures into three categories:

- Above the Mean by +1.5 SD (significantly warmer days).
- Below the Mean by -1.5 SD (significantly cooler days).
- Within ±1.5 SD (normal range).

Data Preparation

Data Source: The dataset was imported from an Excel file.

Preprocessing: The date column was converted to a datetime format for proper indexing and time-series analysis.

Statistical Computations: The script computed the historical mean and standard deviation for daily average temperatures (tavg). These values were used to define thresholds for categorizing the data.

Categorization and Monthly Trends

Each day was classified into one of the three categories based on its temperature value:

- Days exceeding the upper threshold (+1.5 SD) were labeled as "Above."
- Days below the lower threshold (-1.5 SD) were labeled as "Below."
- Remaining days were categorized as "Normal."

The script further analyzed these trends by month:

Monthly Counts: The number of days in each category was tallied for every month.

Majority Trends: For each month, the predominant category was identified, indicating whether the month was predominantly warmer, cooler, or within normal temperature ranges.

Visualization and Outputs

Scatter Plot: A scatter plot was generated with color-coded markers to represent the temperature categories:

-Red dots for "Above" temperatures.

-Blue dots for "Below" temperatures.

-Green dots for "Normal" temperatures.

Summary Table: A detailed table summarizing monthly trends was created, showing the count of days in each category and the majority trend for each month.

Key Findings

Based on the analysis and results presented in Table 2:

Warmest Month: `July` recorded the highest number of days with temperatures significantly above the threshold, marking it as the warmest month of the year.

Coldest Month: `February` had the highest number of days with temperatures significantly below the threshold, making it the coldest month of the year.

Conclusion

This analysis provides a detailed understanding of temperature deviations from the historical mean, highlighting seasonal patterns and significant anomalies. The insights gained can support climate monitoring, urban planning, and resource management strategies in Tehran.

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

def plot_temperature_and_check_majority(data, temp_column='tavg', save_path=None):
    """
    Plot days where the temperature is above or below the historical mean by ±1.5 standard d
    determine if the majority of days are above or below the mean ±1.5 SD, and analyze trenc
    Also saves the results in a nice table format in a PDF.

    Args:
        data (pd.DataFrame): DataFrame containing temperature data.
        temp_column (str): Column to calculate the mean temperature (default 'tavg').
        save_path (str): Optional path to save the figure as a PDF.

    Returns:
        str: A summary of the majority trend and monthly breakdown.
    """
    # Ensure the temperature column exists
    if temp_column not in data.columns:
        raise ValueError(f"The specified temperature column '{temp_column}' does not exist i

    # Calculate the historical mean and standard deviation
    historical_mean = data[temp_column].mean()
    historical_std = data[temp_column].std()

    # Define thresholds for ±1.5 SD
    upper_threshold = historical_mean + 1.5 * historical_std
    lower_threshold = historical_mean - 1.5 * historical_std
```

```
    # Create masks for categories
    above_significant = data[temp_column] > upper_threshold
    below_significant = data[temp_column] < lower_threshold
    normal = ~above_significant & ~below_significant  # Days within ±1.5 std deviation

    # Add month information
    data['month'] = data.index.month
    data['month_name'] = data.index.month_name()

    # Count days by month
    monthly_counts = data.groupby('month').apply(
        lambda x: pd.Series({
            'days_above': (x[temp_column] > upper_threshold).sum(),
            'days_below': (x[temp_column] < lower_threshold).sum(),
        })
    )
    monthly_counts['total_days'] = monthly_counts['days_above'] + monthly_counts['days_below
    monthly_counts['trend'] = monthly_counts.apply(
        lambda x: 'above' if x['days_above'] > x['days_below'] else 'below', axis=1
    )

    # Print monthly trends with month names
    monthly_counts.index = monthly_counts.index.map(lambda x: pd.to_datetime(f'2024-{x:02d}-
    print(monthly_counts)

    # Determine majority category overall
    count_above = above_significant.sum()
    count_below = below_significant.sum()
    count_normal = normal.sum()
    majority = "above" if count_above > count_below else "below" if count_below > count_abov

    print(f"Days above +1.5 SD: {count_above}")
    print(f"Days below -1.5 SD: {count_below}")
    print(f"Days within ±1.5 SD: {count_normal}")
    print(f"The majority of days are {majority} the ±1.5 SD threshold.")

    # Plot the results
    plt.figure(figsize=(14, 8))
    plt.scatter(data.index[above_significant], data[temp_column][above_significant],
                color='green', label='Above Mean + 1.5 Std', alpha=0.7, s=50)
    plt.scatter(data.index[below_significant], data[temp_column][below_significant],
                color='red', label='Below Mean - 1.5 Std', alpha=0.7, s=50)
    plt.scatter(data.index[normal], data[temp_column][normal],
                color='blue', label='Within ±1.5 Std', alpha=0.5, s=30)
    plt.axhline(y=historical_mean, color='black', linestyle='--', label='Mean Temperature',
    plt.title(r"$\mathbf{Days\ Above/Below\ \pm1.5\ Std\ Dev\ from\ Historical\ Mean\ Temper
    plt.xlabel('Date (Month)', fontsize=14)
    plt.ylabel('Temperature (°C)', fontsize=14)

    # Set x-axis as month names
    plt.xticks(data.index[::30], data['month_name'][::30], rotation=45)  # Adjust ticks for
```

```
    plt.legend(loc='best', fontsize=12)
    plt.grid(True)

  # Add a caption to the plot
    caption = (
        "Figure 2 :This plot illustrates the temperature trends over time, showing days wher
        "above or below the historical mean by ±1.5 standard deviations. The green dots repr
        "temperatures above the mean +1.5 standard deviations, the red dots represent days b
        "standard deviations, and the blue dots represent days within the ±1.5 standard devi
    )
    plt.figtext(0.5, -0.15, caption, wrap=True, horizontalalignment='center', fontsize=14, c

    # Save the plot as a PDF if save_path is provided
    if save_path:
        plt.savefig(save_path, format='pdf')
        print(f"Plot saved to {save_path}")

    # Show the plot
    plt.show()

    # Create and save the results as a table
    fig, ax = plt.subplots(figsize=(5.5, 3.5))  # Create a new figure for the table
    ax.axis('off')  # Hide the axis

    # Convert the DataFrame to a table and style it
    table_data = monthly_counts.reset_index()
    table = plt.table(cellText=table_data.values, colLabels=table_data.columns, loc='center'
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1.2, 1.2)  # Adjust the table size

    # Add a title to the table
    fig.text(0.5, 0.95, 'Table 1: Monthly Trends in Temperature Above/Below ±1.5 Std',
         ha='center', va='top', fontsize=14, fontweight='bold')

   # Tighten layout to reduce excess space
    plt.subplots_adjust(left=0.1, right=0.9, top=0.8, bottom=0.1)


    # Save the table as a PDF
    table_pdf_path = save_path.replace(".pdf", "_Table.pdf")
    plt.savefig(table_pdf_path, format='pdf')
    print(f"Table saved to {table_pdf_path}")



    # Return the majority result and monthly breakdown
    return f"The majority of days are {majority} the ±1.5 SD threshold.\n\nMonthly Trends:\r

# Example usage
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'  # Replace with your file path
```

```
data = pd.read_excel(file_path)
data['date'] = pd.to_datetime(data['date'])  # Convert the date column to datetime format
data.set_index('date', inplace=True)

# Call the function
save_path = r'E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temperature_Above_Below
result = plot_temperature_and_check_majority(data, temp_column='tavg', save_path=save_path)
print(result)
```

```
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\3168857992.py:40: DeprecationWarnin
  monthly_counts = data.groupby('month').apply(
           days_above  days_below  total_days  trend
month
January             0           3           3  below
February            0           9           9  below
March               0           2           2  below
April               0           0           0  below
May                 0           0           0  below
June                0           0           0  below
July                9           0           9  above
August              7           0           7  above
September           0           0           0  below
October             0           0           0  below
November            0           0           0  below
December            0           0           0  below
Days above +1.5 SD: 16
Days below -1.5 SD: 14
Days within ±1.5 SD: 336
The majority of days are above the ±1.5 SD threshold.
Plot saved to E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temperature_Abov
```

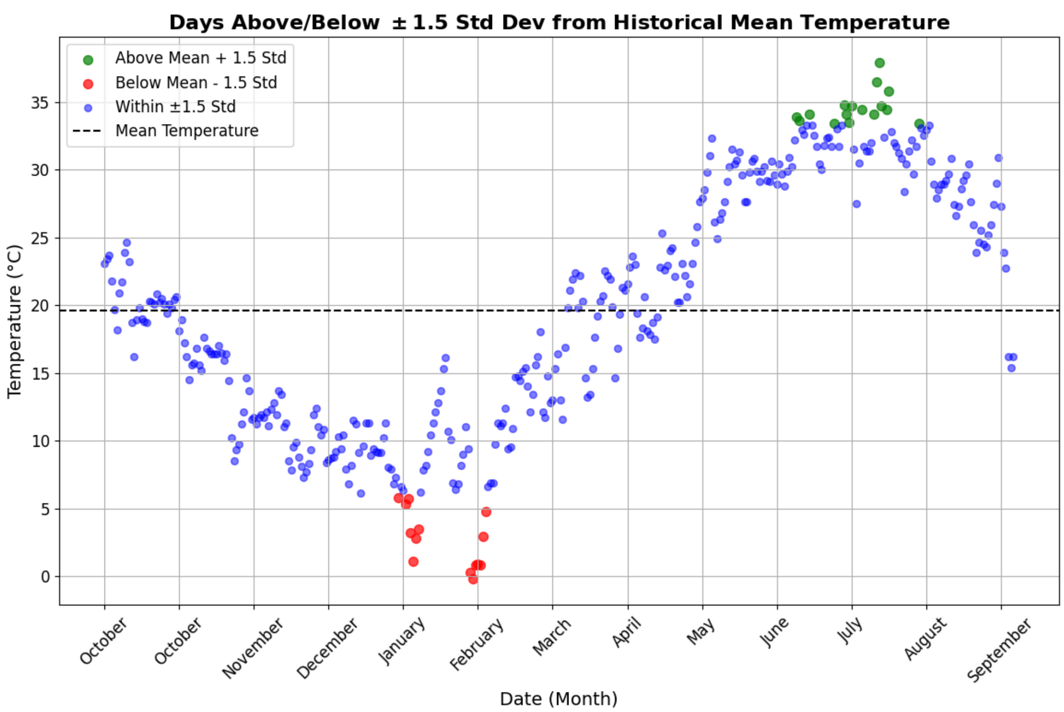


Figure 2 :This plot illustrates the temperature trends over time, showing days where the temperature is above or below the historical mean by ±1.5 standard deviations. The green dots represent days with temperatures above the mean +1.5 standard deviations, the red dots represent days below the mean -1.5 standard deviations, and the blue dots represent days within the ±1.5 standard deviation range.

```
Table saved to E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temperature_Abo
The majority of days are above the ±1.5 SD threshold.

Monthly Trends:
           days_above  days_below  total_days  trend
month
January             0           3           3  below
February            0           9           9  below
March               0           2           2  below
April               0           0           0  below
May                 0           0           0  below
June                0           0           0  below
```

```
July                 9            0            9  above
August               7            0            7  above
September            0            0            0  below
October              0            0            0  below
November             0            0            0  below
December             0            0            0  below
```

Table 1: Monthly Trends in Temperature Above/Below ±1.5 Std

month	days_above	days_below	total_days	trend
January	0	3	3	below
February	0	9	9	below
March	0	2	2	below
April	0	0	0	below
May	0	0	0	below
June	0	0	0	below
July	9	0	9	above
August	7	0	7	above
September	0	0	0	below
October	0	0	0	below
November	0	0	0	below
December	0	0	0	below

Seasonal Temperature Analysis

To divide the dataset into the four seasons (Winter, Spring, Summer, Fall), I will use the following general guidelines for seasonal divisions in Tehran:

- **Winter**: December 21 to March 20
- **Spring**: March 21 to June 20
- **Summer**: June 21 to September 20
- **Fall**: September 21 to December 20

This analysis focuses on average temperature data collected for Tehran from December 2023 to December 2024. To begin, I loaded the dataset from an Excel file and converted the date column into a datetime format for easier manipulation. Seasonal date ranges for Winter, Spring, Summer, and Fall were then defined, and the dataset was segmented accordingly. For each season, I extracted the average temperature data to prepare for visualization.

To better understand the distribution of temperature data across seasons, I generated two boxplots: one before and one after removing outliers. Outliers were identified using two methods: the Z-Score (±1.5 standard deviations) and the Interquartile Range (IQR). In the first plot, outliers identified by the Z-Score method were marked in gold, while those identified by the IQR method were highlighted in orange. Each plot was annotated with the mean and standard deviation for the temperature data of each season.

Following the identification of outliers, I removed them from the dataset and recalculated the statistics. The second boxplot displayed the data after outlier removal. Additionally, I calculated and displayed the percentage of data removed for each season as well as overall.

To enhance the clarity of the visualizations, I added a caption explaining the plots, included a simple legend to differentiate between the types of outliers, and incorporated gridlines for better readability. This analysis provided a clear view of seasonal temperature trends and demonstrated the influence of outliers on the data distribution.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Load the dataset
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'
df = pd.read_excel(file_path)

# Convert the 'date' column to datetime
df['date'] = pd.to_datetime(df['date'])
```

```
# Define seasonal date ranges
winter = (df['date'] >= '2023-12-21') & (df['date'] <= '2024-03-20')
spring = (df['date'] >= '2024-03-21') & (df['date'] <= '2024-06-20')
summer = (df['date'] >= '2024-06-21') & (df['date'] <= '2024-09-20')
fall = (df['date'] >= '2024-09-21') & (df['date'] <= '2024-12-20')

# Segment the dataset by seasons
seasons = {
    'Winter': df[winter]['tavg'],
    'Spring': df[spring]['tavg'],
    'Summer': df[summer]['tavg'],
    'Fall': df[fall]['tavg']
}

# Prepare the data for each season
seasons_data = [seasons['Winter'], seasons['Spring'], seasons['Summer'], seasons['Fall']]

# Define colors for each season
box_colors = ['skyblue', 'lightgreen', 'lightcoral', 'lightsalmon']

# Function to calculate Z-Score and IQR outliers
def detect_outliers(data):
    # Z-Score method (±1.5 standard deviations)
    mean = np.mean(data)
    std = np.std(data)
    z_upper = mean + 1.5 * std
    z_lower = mean - 1.5 * std
    z_outliers = data[(data > z_upper) | (data < z_lower)]

    # IQR method
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3 - q1
    iqr_upper = q3 + 1.5 * iqr
    iqr_lower = q1 - 1.5 * iqr
    iqr_outliers = data[(data > iqr_upper) | (data < iqr_lower)]

    return z_outliers, iqr_outliers, mean, std

# Create a figure with two subplots (before and after removing outliers)
fig, axs = plt.subplots(1, 2, figsize=(16, 9))

# Plot Before Removing Outliers
axs[0].set_title('Temperature Distribution by Season (Before Removing Outliers)', fontsize=1
axs[0].set_ylabel('Temperature (°C)', fontsize=14, fontweight='bold', color='darkblue')
axs[0].set_xlabel('Season', fontsize=14, fontweight='bold', color='darkblue')

# Create the boxplot (Before Removing Outliers)
bp = axs[0].boxplot(seasons_data, labels=['Winter', 'Spring', 'Summer', 'Fall'], patch_artis
                    boxprops=dict(facecolor='lightgray', color='black'), whiskerprops=dict(c
                    flierprops=dict(markerfacecolor='red', marker='o', markersize=8, linesty
                    medianprops=dict(color='black', linewidth=2))
```

```
# Assign colors to each boxplot
for patch, color in zip(bp['boxes'], box_colors):
    patch.set_facecolor(color)

# Add scatter plots and annotate outliers (Before Removing)
for i, season_data in enumerate(seasons_data):
    jitter = np.random.normal(0, 0.03, len(season_data))  # Add slight horizontal jitter

    # Scatter individual data points
    axs[0].scatter([i + 1 + jitter_offset for jitter_offset in jitter], season_data,
                   color='black', alpha=0.6, s=15)

    # Detect outliers using both methods
    z_outliers, iqr_outliers, mean, std = detect_outliers(season_data)

    # Plot Z-Score outliers
    axs[0].scatter([i + 1] * len(z_outliers), z_outliers, color='gold', edgecolor='black', s

    # Plot IQR outliers (now explicitly set to orange)
    axs[0].scatter([i + 1] * len(iqr_outliers), iqr_outliers, color='orange', edgecolor='bla

    # Annotate mean and std
    axs[0].text(i + 1, mean + 0.3, f'Mean: {mean:.2f}', ha='center', va='bottom', color='dar
    axs[0].text(i + 1, mean - 0.3, f'Std: {std:.2f}', ha='center', va='top', color='darkred'

# Plot After Removing Outliers
axs[1].set_title('Temperature Distribution by Season (After Removing Outliers)', fontsize=16
axs[1].set_ylabel('Temperature (°C)', fontsize=14, fontweight='bold', color='darkblue')
axs[1].set_xlabel('Season', fontsize=14, fontweight='bold', color='darkblue')

# Remove outliers and plot
seasons_no_outliers = []
total_removed = 0  # Variable to keep track of total removed outliers
season_names = ['Winter', 'Spring', 'Summer', 'Fall']

for season_data, season_name in zip(seasons_data, season_names):
    z_outliers, iqr_outliers, mean, std = detect_outliers(season_data)
    clean_data = season_data[~season_data.isin(z_outliers) & ~season_data.isin(iqr_outliers)
    seasons_no_outliers.append(clean_data)

    # Calculate percentage of removed data
    removed_data = len(season_data) - len(clean_data)
    total_removed += removed_data
    print(f"Removed {removed_data} outliers from {season_name} data (Total {removed_data / l

# Calculate the total percentage of removed data across all seasons
total_data_points = sum(len(season_data) for season_data in seasons_data)
percentage_removed = (total_removed / total_data_points) * 100
print(f"\nTotal percentage of data removed as outliers: {percentage_removed:.2f}%")
```

```
# Create the boxplot (After Removing Outliers)
bp = axs[1].boxplot(seasons_no_outliers, labels=['Winter', 'Spring', 'Summer', 'Fall'], patc
                    boxprops=dict(facecolor='lightgray', color='black'), whiskerprops=dict(c
                    flierprops=dict(markerfacecolor='red', marker='o', markersize=8, linesty
                    medianprops=dict(color='black', linewidth=2))

# Assign colors to each boxplot
for patch, color in zip(bp['boxes'], box_colors):
    patch.set_facecolor(color)

# Add scatter plots and annotate statistics (After Removing Outliers)
for i, season_data in enumerate(seasons_no_outliers):
    jitter = np.random.normal(0, 0.03, len(season_data))
    axs[1].scatter([i + 1 + jitter_offset for jitter_offset in jitter], season_data, color='
    mean = np.mean(season_data)
    std = np.std(season_data)
    axs[1].text(i + 1, mean + 0.3, f'Mean: {mean:.2f}', ha='center', va='bottom', color='dar
    axs[1].text(i + 1, mean - 0.3, f'Std: {std:.2f}', ha='center', va='top', color='darkred'

# Add a simple legend to the right of the first plot
legend_elements = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='gold', markersize=10, label
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='orange', markersize=10, lab
]
axs[0].legend(handles=legend_elements, loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=

# Add gridlines to both plots
for ax in axs:
    ax.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)

# Add a caption to the plots
caption = ("Figure 3: Comparison of temperature distributions across seasons.\n"
           "The left plot shows the original data with Z-Score (gold) and IQR (orange) outli
           "The right plot shows the cleaned data with outliers removed for better represent
plt.figtext(0.5, -0.15, caption, wrap=True, horizontalalignment='center', fontsize=14, color

# Adjust layout for better fit
plt.tight_layout()
plt.show()
```

```
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\460221449.py:59: MatplotlibDeprecat
  bp = axs[0].boxplot(seasons_data, labels=['Winter', 'Spring', 'Summer', 'Fall'], pa
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\460221449.py:115: MatplotlibDepreca
  bp = axs[1].boxplot(seasons_no_outliers, labels=['Winter', 'Spring', 'Summer', 'Fal
Removed 17 outliers from Winter data (Total 18.68% removed)
Removed 18 outliers from Spring data (Total 19.57% removed)
Removed 11 outliers from Summer data (Total 11.96% removed)
Removed 0 outliers from Fall data (Total 0.00% removed)

Total percentage of data removed as outliers: 16.14%
```

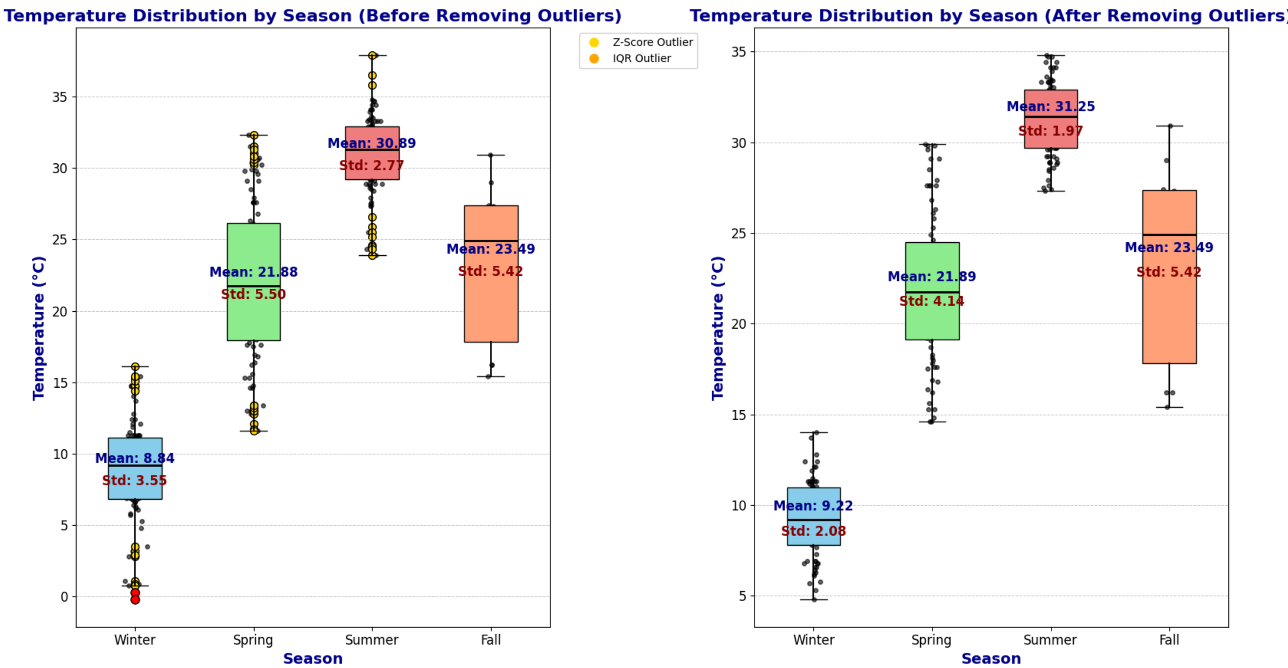


Figure 3: Comparison of temperature distributions across seasons.
The left plot shows the original data with Z-Score (gold) and IQR (orange) outliers highlighted.
The right plot shows the cleaned data with outliers removed for better representation of seasonal trends.

Seasonal Temperature Heatmap for Tehran

In this analysis, I processed a weather dataset to create a seasonal temperature heatmap for Tehran in the year 2024. First, I loaded the dataset from an Excel file and converted the 'date' column to a datetime format for easier manipulation. I then defined the date ranges for each season: winter, spring, summer, and fall. Using these seasonal ranges, I segmented the dataset accordingly. Next, I resampled the data to daily averages for each season and ensured consistency in the dataset length by filling any missing values using forward-fill and backward-fill methods. I combined the seasonal data into a single matrix, where each season was represented as a column. I then visualized this matrix as a heatmap, with color representing temperature, ranging from blue (cold) to red (warm). Labels for the axes and a color bar were added for clarity, and the plot was saved as a PDF. The resulting heatmap provides a clear visual representation of the average daily temperatures across the four seasons in Tehran.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Load the dataset
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'
df = pd.read_excel(file_path)

# Convert the 'date' column to datetime
df['date'] = pd.to_datetime(df['date'])

# Define seasonal date ranges
winter = (df['date'] >= '2023-12-21') & (df['date'] <= '2024-03-20')
spring = (df['date'] >= '2024-03-21') & (df['date'] <= '2024-06-20')
summer = (df['date'] >= '2024-06-21') & (df['date'] <= '2024-09-20')
fall = (df['date'] >= '2024-09-21') & (df['date'] <= '2024-12-20')

# Segment the dataset by seasons
seasons = {
    'Winter': df[winter],
    'Spring': df[spring],
    'Summer': df[summer],
    'Fall': df[fall]
}

# Prepare the data for each season and ensure consistency in length
```

```
resampled_seasons = {}
for season, data in seasons.items():
    resampled_seasons[season] = data.set_index('date').resample('D').mean()['tavg']

# Convert the resampled data to a DataFrame where each season is a column
season_matrix = pd.DataFrame({
    'Winter': resampled_seasons['Winter'],
    'Spring': resampled_seasons['Spring'],
    'Summer': resampled_seasons['Summer'],
    'Fall': resampled_seasons['Fall']
})

# Ensure all seasons have the same length using forward-fill and backward-fill
season_matrix = season_matrix.fillna(method='ffill').fillna(method='bfill')

# Plot the heatmap using Matplotlib
fig, ax = plt.subplots(figsize=(10, 6))

# Create a colormap from blue (cold) to red (warm)
cmap = plt.get_cmap("coolwarm")

# Plot the heatmap without interpolation
cax = ax.imshow(season_matrix.T, cmap=cmap, aspect='auto', interpolation='none')

# Add color bar for temperature scale
cbar = plt.colorbar(cax, ax=ax, label='Temperature (°C)', pad=0.02)
cbar.ax.tick_params(labelsize=10)

# Add labels and title
ax.set_title('Seasonal Temperature Heatmap', fontsize=16, fontweight='bold')
ax.set_xlabel('Day of Year', fontsize=14)
ax.set_ylabel('Season', fontsize=14)

# Set the x-ticks and y-ticks labels
x_ticks = np.linspace(0, len(season_matrix) - 1, 5).astype(int)
x_labels = pd.date_range(start='2024-01-01', periods=len(season_matrix), freq='D').strftime(
ax.set_xticks(x_ticks)
ax.set_xticklabels(x_labels, rotation=45, fontsize=10)

ax.set_yticks(np.arange(4))
ax.set_yticklabels(['Winter', 'Spring', 'Summer', 'Fall'], fontsize=12)

# Add a caption below the plot
caption = "Figure 4 : This heatmap represents the average daily temperatures across seasons 
plt.figtext(0.5, -0.02, caption, wrap=True, horizontalalignment='center', fontsize=14, color

# Ensure layout is tight
plt.tight_layout(rect=[0, 0.05, 1, 1])  # Adjust layout to fit caption

# Save the figure as a PDF
output_path = r'E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Seasonal_Temperature_
```

```
plt.savefig(output_path, format='pdf', bbox_inches='tight')

# Show the plot
plt.show()
```

```
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\3584657650.py:40: FutureWarning: DataF
  season_matrix = season_matrix.fillna(method='ffill').fillna(method='bfill')
```

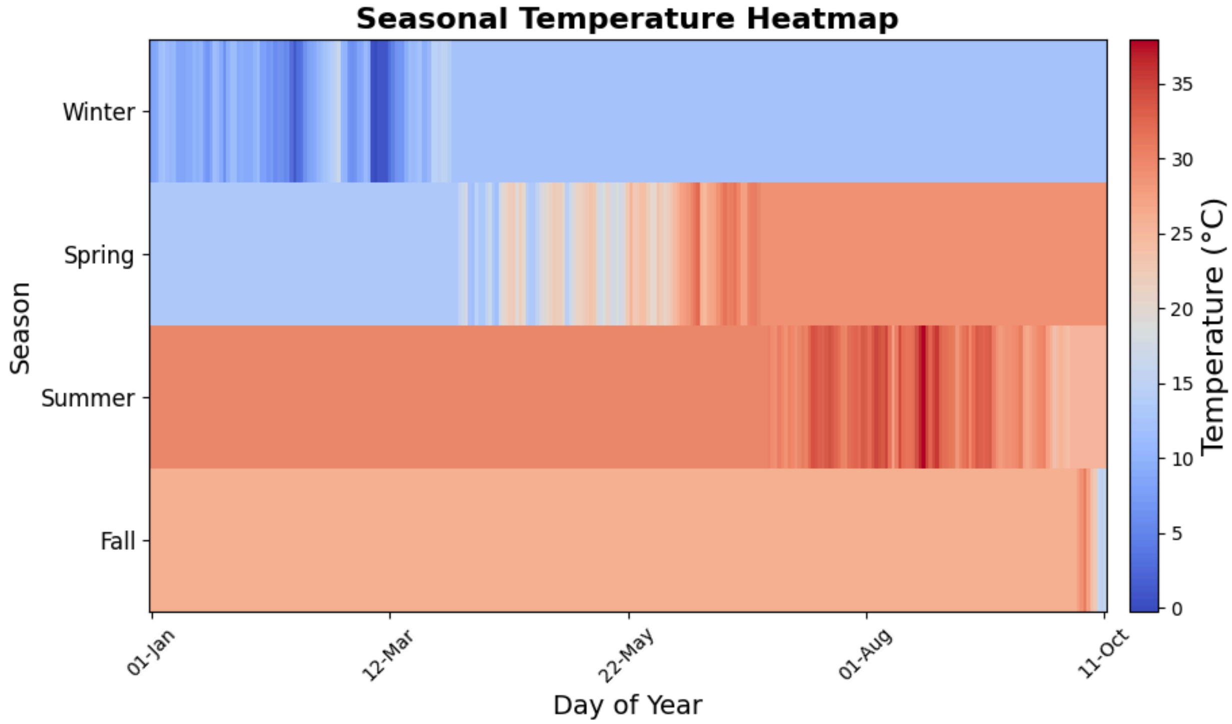


Figure 4 : This heatmap represents the average daily temperatures across seasons in Tehran for the year 2024.

Statistical Analysis of Weather Data

In this analysis, I loaded and preprocessed weather data from an Excel file, focusing on temperature (`tavg`), wind speed (`wspd`), and precipitation (`prcp`). The 'date' column was converted to datetime format, and it was set as the index for easier time-based analysis. I then performed a statistical analysis of the distributions of these variables by plotting histograms and kernel density estimates

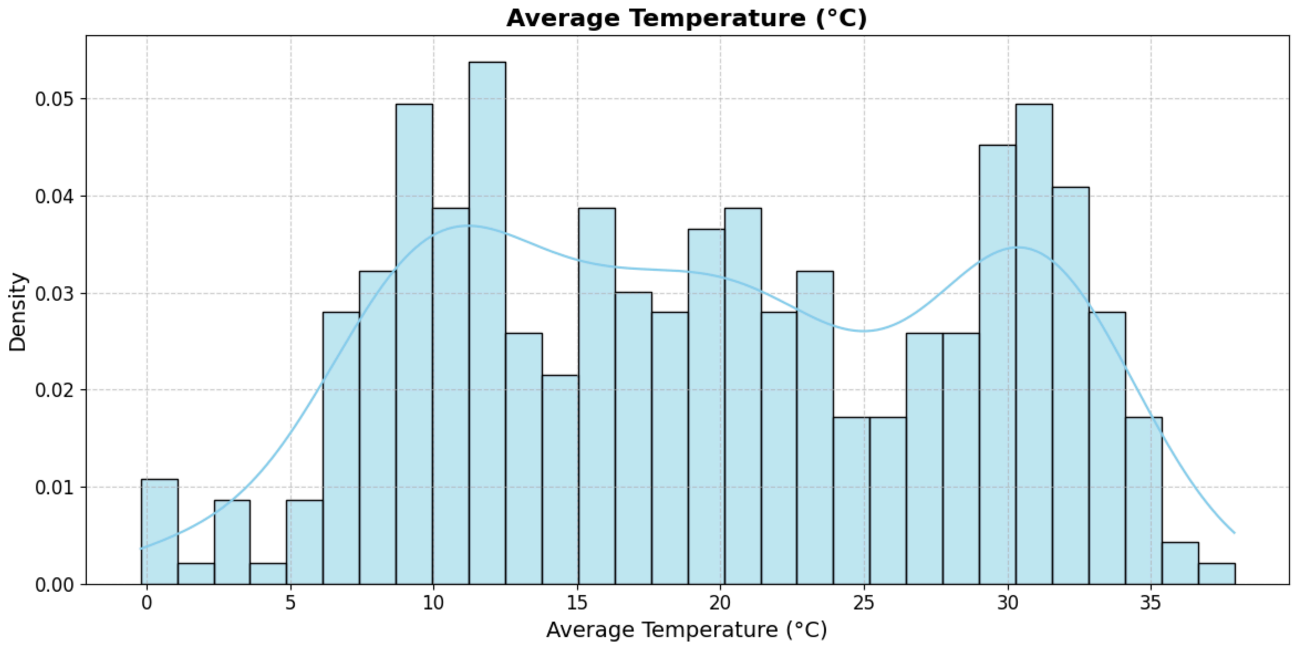
(KDE) for each column. For each variable, I calculated and reported the skewness, kurtosis, and performed the Shapiro-Wilk normality test to assess whether the data followed a normal distribution. The results of these statistical analyses were compiled into a table, which was saved as a CSV file. The plots, including both histograms and KDEs, were saved in a PDF file with a caption, providing a comprehensive visual summary of the distribution trends. The final output includes both the figure and the statistical analysis table, which were saved in the specified output folder for further review and documentation.

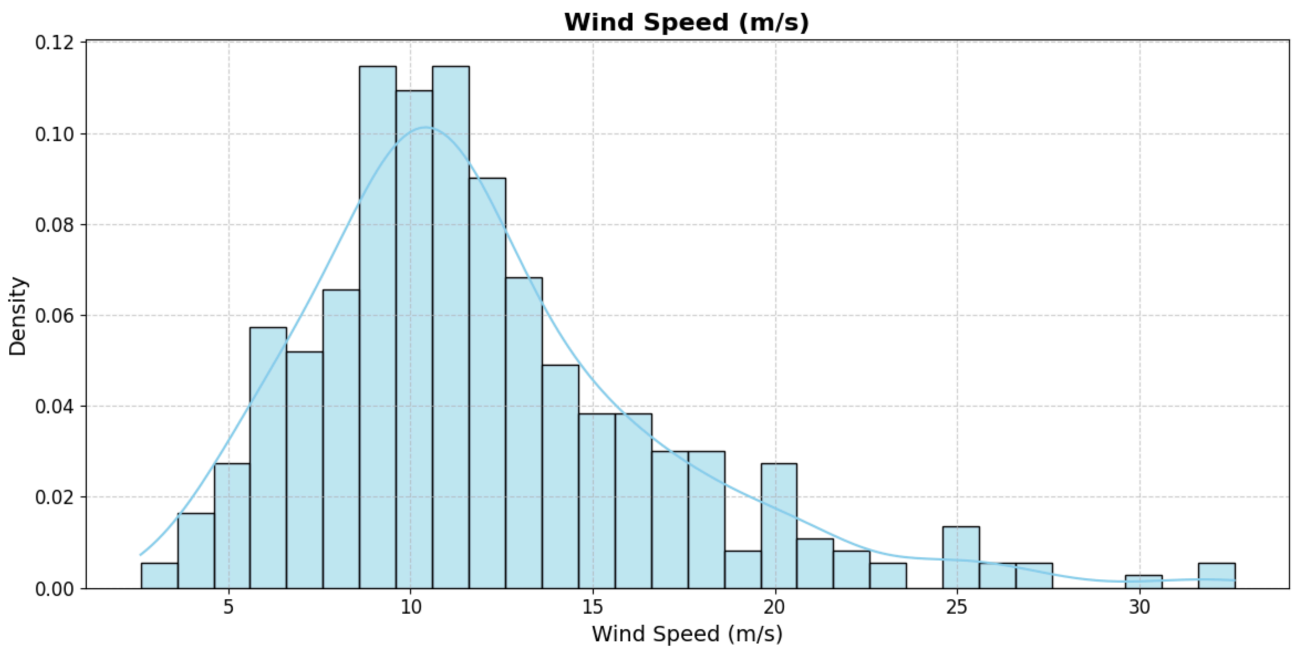
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import shapiro, skew, kurtosis
import seaborn as sns
from matplotlib.backends.backend_pdf import PdfPages

def load_and_prepare_data(file_path):
    """
    Load the Excel file and preprocess the data.
    - Converts the 'date' column to datetime.
    - Sets 'date' as the index.
    """
    data = pd.read_excel(file_path)
    data['date'] = pd.to_datetime(data['date'])
    data.set_index('date', inplace=True)
    return data

def save_figure_with_caption(fig, caption, save_path):
    """
    Save a Matplotlib figure as a PDF with a caption.
    """
    with PdfPages(save_path) as pdf:
        pdf.savefig(fig, bbox_inches='tight')
        plt.figtext(0.5, -0.01, caption, wrap=True, horizontalalignment='center', fontsize=1
        pdf.close()

def analyze_distributions(data, columns, output_folder=None):
    """
    Analyze and visualize the distribution of specified data columns.
    - Plot histograms and KDEs for each column in subplots.
    - Calculate skewness, kurtosis, and normality test for each column.
    - Save the figure and table as files.
    """
    stats_results = {}
    num_columns = len(columns)

    # Prepare subplots
    fig, axs = plt.subplots(num_columns, 1, figsize=(12, 6 * num_columns), sharex=False)
    if num_columns == 1:
        axs = [axs]  # Ensure axs is iterable even for 1 subplot
```

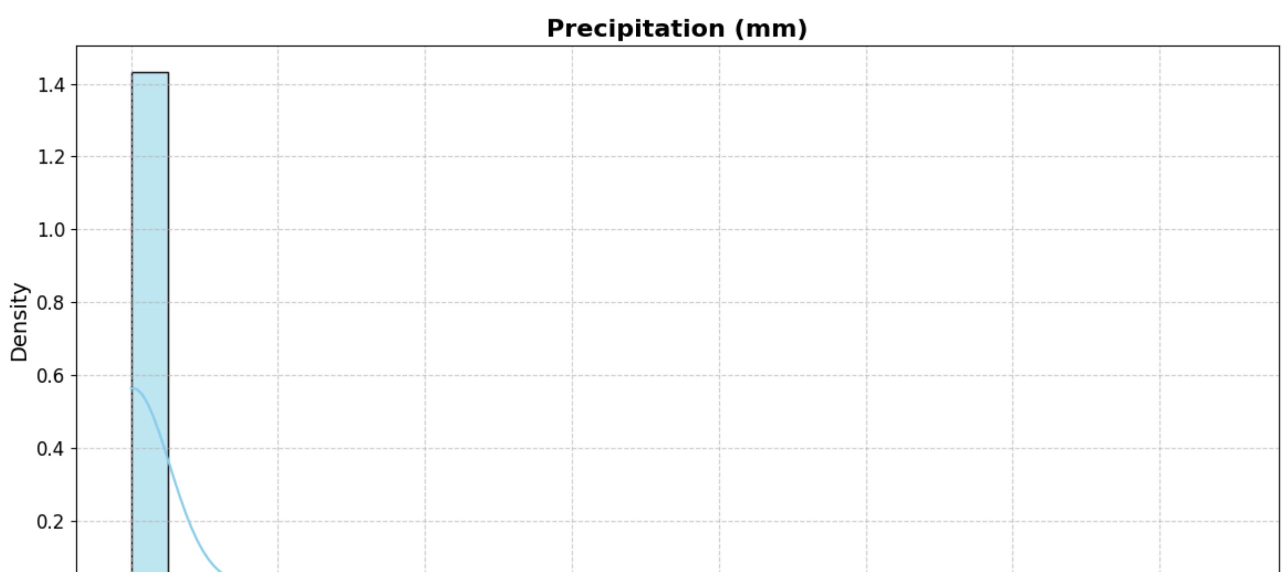
```
# Define descriptive titles for each column
column_titles = {
    'tavg': 'Average Temperature (°C)',
    'prcp': 'Precipitation (mm)',
    'wspd': 'Wind Speed (m/s)'
}

for i, column in enumerate(columns):
    col_data = data[column].dropna()

    # Calculate skewness, kurtosis, and normality
    col_skewness = skew(col_data)
    col_kurtosis = kurtosis(col_data)
    shapiro_stat, shapiro_p = shapiro(col_data)
    normality = "Yes" if shapiro_p > 0.05 else "No"

    # Add results to table
    stats_results[column] = {
        "Skewness": col_skewness,
        "Kurtosis": col_kurtosis,
        "Shapiro-Wilk Test Stat": shapiro_stat,
        "Shapiro-Wilk P-value": shapiro_p,
        "Normal Distribution": normality
    }

    # Plot histogram and KDE
    sns.histplot(col_data, bins=30, kde=True, color='skyblue', edgecolor='black', stat='
    axs[i].set_title(column_titles[column], fontsize=16, fontweight='bold')  # Updated t
    axs[i].set_xlabel(column_titles[column], fontsize=14)
    axs[i].set_ylabel('Density', fontsize=14)
    axs[i].grid(True, linestyle='--', alpha=0.6)

plt.tight_layout()

# Save the figure
if output_folder:
    fig_path = f"{output_folder}/Weather_Distributions.pdf"
    caption = "Figure 6: Distributions of temperature, wind speed, and precipitation wit
    save_figure_with_caption(fig, caption, fig_path)
plt.show()

# Save and print the table
stats_table = pd.DataFrame(stats_results).T
print("\nStatistical Analysis Table:")
print(stats_table)

if output_folder:
    table_path = f"{output_folder}/Weather_Stats_Table.csv"
    stats_table.to_csv(table_path)
    print(f"Statistical analysis table saved at: {table_path}")
```

```
# Specify the path of the Excel file and the output folder
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'
output_folder = r'E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950'

# Load and prepare data
data = load_and_prepare_data(file_path)

# Analyze the distributions of temperature, wind speed, and precipitation
columns_to_analyze = ['tavg', 'wspd', 'prcp']
analyze_distributions(data, columns=columns_to_analyze, output_folder=output_folder)
```

```
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\1915983428.py:22: MatplotlibDepreca
  with PdfPages(save_path) as pdf:
```







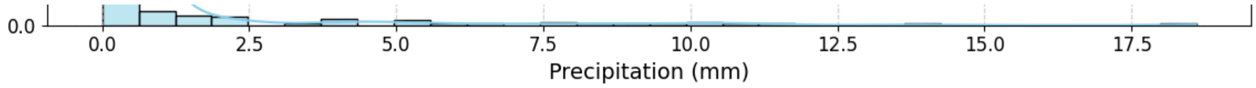


Figure 6: Distributions of temperature, wind speed, and precipitation with KDE.

```
Statistical Analysis Table:
      Skewness  Kurtosis Shapiro-Wilk Test Stat Shapiro-Wilk P-value  \
tavg   0.00983  -1.15549               0.956811                  0.0
wspd   1.18579  2.034058               0.927751                  0.0
prcp  5.042631   28.8941                0.29523                  0.0

     Normal Distribution
tavg                  No
wspd                  No
prcp                  No
Statistical analysis table saved at: E:\CompAppTools\FinalProject-MH\Final_Project_CM
```

Weather Data Analysis: Temperature and Precipitation Trends in Tehran

In this analysis, I processed and visualized weather data for Tehran by focusing on the trends in temperature (minimum, maximum, and average) and precipitation over time. First, I loaded and prepared the data by converting the 'date' column to a datetime format and setting it as the index. I then created a combined plot displaying temperature trends in light colors and precipitation with dark purple bars. The plot highlights extreme values for both temperature and precipitation, including the highest and lowest recorded values with annotations. I also calculated the correlation between temperature and precipitation and displayed it on the plot. Furthermore, I identified and printed the months that had less precipitation than the overall average. The plot is saved as a PDF file with a caption describing its contents, and the results, including months with low precipitation, are printed for further analysis. Finally, I displayed the plot, showing the temperature and precipitation trends over time with the necessary annotations and labels for clarity.

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

def load_and_prepare_data(file_path):
    """
    Load the Excel file and preprocess the data.
    - Converts the 'date' column to datetime.
    - Sets 'date' as the index.

    Args:
        file_path (str): Path to the Excel file.

    Returns:
        pd.DataFrame: Preprocessed DataFrame.
    """
    data = pd.read_excel(file_path)
    data['date'] = pd.to_datetime(data['date'])
    data.set_index('date', inplace=True)
    return data

def plot_temperature_and_precipitation(data, save_path=None):
    """
    Plot temperature trends (avg, min, max) and precipitation on the same graph.
    - Temperature will be in light colors.
    - Precipitation will be darker (dark purple) and with thicker bars.
    Also identifies and highlights extreme values (max/min temperature and precipitation).
    - Computes and displays the correlation between temperature and precipitation.
    - Displays months with less than the mean precipitation.
```

```
    Args:
        data (pd.DataFrame): DataFrame containing temperature and precipitation data.
        save_path (str): Optional path to save the figure as a PDF.
    """

    # Find extreme values (max/min for temperature and precipitation)
    max_temp = data['tmax'].max()
    max_temp_date = data['tmax'].idxmax()
    min_temp = data['tmin'].min()
    min_temp_date = data['tmin'].idxmin()

    max_precip = data['prcp'].max()
    max_precip_date = data['prcp'].idxmax()
    min_precip = data['prcp'].min()
    min_precip_date = data['prcp'].idxmin()

    # Calculate monthly mean precipitation
    monthly_precip = data['prcp'].resample('M').mean()

    # Calculate overall mean precipitation
    mean_precip = data['prcp'].mean()

    # Identify months with less than the mean precipitation
    months_below_mean = monthly_precip[monthly_precip < mean_precip].index.strftime('%b')

    # Calculate correlation between temperature and precipitation
    temp_precip_correlation = data[['tavg', 'prcp']].corr().iloc[0, 1]

    # Create the plot with a larger figure size
    fig, ax1 = plt.subplots(figsize=(20, 12))  # Bigger figure size

    # Plot temperature trends with light colors
    ax1.plot(data.index, data['tmin'], label='Min Temperature (°C)', color='lightblue', line
    ax1.plot(data.index, data['tmax'], label='Max Temperature (°C)', color='lightcoral', lin
    ax1.plot(data.index, data['tavg'], label='Average Temperature (°C)', color='lightgreen',
    ax1.set_xlabel('Date', fontsize=18)
    ax1.set_ylabel('Temperature (°C)', color='black', fontsize=18)
    ax1.tick_params(axis='y', labelcolor='black', labelsize=16)

    # Create a second y-axis for precipitation with dark purple color and thicker bars
    ax2 = ax1.twinx()
    ax2.bar(data.index, data['prcp'], color='purple', alpha=0.8, width=1.0, label='Precipita
    ax2.set_ylabel('Precipitation (mm)', color='blue', fontsize=18)
    ax2.tick_params(axis='y', labelcolor='blue', labelsize=16)

    # Highlight extreme values on the plot
    ax1.scatter(max_temp_date, max_temp, color='lightgray', label=f'Highest Max Temp ({max_t
    ax1.scatter(min_temp_date, min_temp, color='lightgray', label=f'Lowest Min Temp ({min_te
    ax2.scatter(max_precip_date, max_precip, color='darkgreen', label=f'Highest Precip ({max
    ax2.scatter(min_precip_date, min_precip, color='purple', label=f'Lowest Precip ({min_pre
```

```
    # Annotate extreme values with bold text for precipitation and light color for temperatu
    ax1.annotate(f'{max_temp}°C', (max_temp_date, max_temp), textcoords="offset points", xyt
    ax1.annotate(f'{min_temp}°C', (min_temp_date, min_temp), textcoords="offset points", xyt
    ax2.annotate(f'{max_precip}mm', (max_precip_date, max_precip), textcoords="offset points
    ax2.annotate(f'{min_precip}mm', (min_precip_date, min_precip), textcoords="offset points

    # Add a horizontal line for the mean precipitation (dark red with thick line)
    ax2.axhline(y=mean_precip, color='darkred', linestyle='-', linewidth=4, label=f'Mean Pre

    # Add title and grid
    plt.title('Temperature and Precipitation Trends in Tehran Over Time', fontsize=24, fontw
    plt.grid(True)

    # Move the legends outside the plot further right
    ax1.legend(loc='upper left', bbox_to_anchor=(1.1, 1), fontsize=16)
    ax2.legend(loc='upper left', bbox_to_anchor=(1.1, 0.7), fontsize=16)

    # Format X-axis to show month names
    ax1.set_xticks(data.index[::int(len(data)/12)])  # Show 12 ticks, approximately one per
    ax1.set_xticklabels(data.index.strftime('%b')[::int(len(data)/12)], rotation=45, fontsiz

    # Display correlation in the top-left of the plot
    plt.figtext(0.05, 0.8, f'Temperature-Precipitation Correlation: {temp_precip_correlation

    # Adding the caption to the figure
    caption = (
        "Figure 7 :This plot illustrates the trends in minimum, maximum, and average tempera
        "Temperature trends are shown in light colors, while precipitation is represented wi
        "Extreme values for both temperature and precipitation are highlighted, and the corr
        "temperature and precipitation is displayed in the top left corner."
    )
    plt.figtext(0.5, -0.05, caption, wrap=True, horizontalalignment='center', fontsize=14, c

    # Adjust layout to fit everything properly
    plt.tight_layout()

    # Save the figure as a PDF if save_path is provided
    if save_path:
        plt.savefig(save_path, format='pdf')

    # Show the plot
    plt.show()

    # Print months with less than mean precipitation
    print("Months with less than the mean precipitation:")
    print(list(months_below_mean))

# Call the function to plot the combined temperature and precipitation data
save_path = r'E:\CompAppTools\FinalProject-MH\Final_Project_CMSC6950\Temp_Precip_Plot_with_M
plot_temperature_and_precipitation(data, save_path)
```

```
C:\Users\bmehe\AppData\Local\Temp\ipykernel_83472\2491826702.py:47: FutureWarning: 'M' i
  monthly_precip = data['prcp'].resample('M').mean()
```

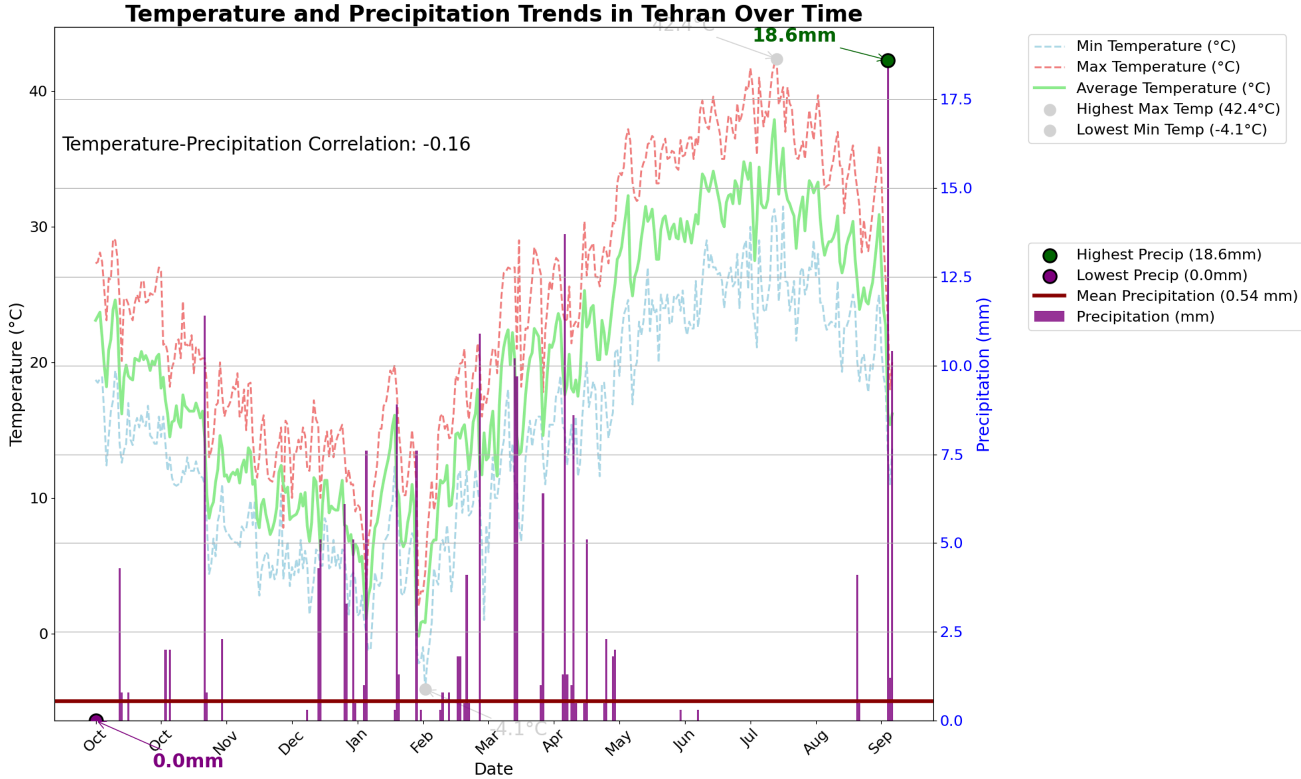


Figure 7 :This plot illustrates the trends in minimum, maximum, and average temperatures in Tehran over time. Temperature trends are shown in light colors, while precipitation is represented with darker purple bars. Extreme values for both temperature and precipitation are highlighted, and the correlation between temperature and precipitation is displayed in the top left corner.

```
Months with less than the mean precipitation:
['Oct', 'Dec', 'Jun', 'Jul', 'Aug']
```

Visualizing Temperature and Precipitation Trends

In this analysis, I aimed to explore the temperature and precipitation trends over time using various time series plots. To achieve this, I utilized several visualization techniques to examine the data comprehensively. Below are the steps I followed, and the corresponding plots I created:

Line Plot (Time Series Plot)

To begin, I created a line plot to visualize the temperature and precipitation trends over time. This plot displays continuous data for both temperature and precipitation, where the x-axis represents the date, and the y-axis shows the values for temperature and precipitation. This plot effectively shows how both variables evolve over the entire period of the dataset.

In this analysis, I processed weather data from Tehran to calculate and visualize seasonal statistics related to temperature, precipitation, and wind speed. The dataset, stored in an Excel file, was loaded and the 'date' column was converted into a datetime format for easier manipulation. I defined the seasonal date ranges for winter, spring, summer, and fall, then segmented the data accordingly. For each season, I calculated key statistics such as the minimum, maximum, mean, median, and standard deviation for temperature, precipitation, and wind speed. These statistics were organized into a table for clearer presentation. Additionally, I created a grouped bar chart to visually compare the average values of temperature, precipitation, and wind speed across the four seasons. The bar chart was annotated with precise values to improve readability. To supplement the visual representation, I added a detailed table beneath the figure summarizing the calculated statistics. Finally, I saved the figure and table as a PDF document for further use.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from tabulate import tabulate  # For pretty-printing tables

# Load the dataset
file_path = r'E:\CompAppTools\Project\TehranWeather.xlsx'
df = pd.read_excel(file_path)

# Check the first few rows of the data to confirm it has loaded correctly
print(df.head())

# Assuming 'date' column is in the format 'YYYY-MM-DD', convert it to datetime
df['date'] = pd.to_datetime(df['date'])

# Define seasonal date ranges
winter = (df['date'] >= '2023-12-21') & (df['date'] <= '2024-03-20')
spring = (df['date'] >= '2024-03-21') & (df['date'] <= '2024-06-20')
summer = (df['date'] >= '2024-06-21') & (df['date'] <= '2024-09-20')
```

```
fall = (df['date'] >= '2024-09-21') & (df['date'] <= '2024-12-20')

# Segment the dataset by seasons
seasons = {
    'Winter': df[winter],
    'Spring': df[spring],
    'Summer': df[summer],
    'Fall': df[fall]
}

# Initialize an empty list to store statistics
season_stats = []

# Calculate the statistics for each season
for season, data in seasons.items():
    temp_stats = {
        'Season': season,
        'Min Temp (°C)': data['tavg'].min(),
        'Max Temp (°C)': data['tavg'].max(),
        'Mean Temp (°C)': data['tavg'].mean(),
        'Median Temp (°C)': data['tavg'].median(),
        'Std Temp (°C)': data['tavg'].std(),
        'Min Precip (mm)': data['prcp'].min(),
        'Max Precip (mm)': data['prcp'].max(),
        'Mean Precip (mm)': data['prcp'].mean(),
        'Median Precip (mm)': data['prcp'].median(),
        'Std Precip (mm)': data['prcp'].std(),
        'Min Wind Speed (m/s)': data['wspd'].min(),
        'Max Wind Speed (m/s)': data['wspd'].max(),
        'Mean Wind Speed (m/s)': data['wspd'].mean(),
        'Median Wind Speed (m/s)': data['wspd'].median(),
        'Std Wind Speed (m/s)': data['wspd'].std(),
    }
    season_stats.append(temp_stats)

# Convert to DataFrame for better display
season_stats_df = pd.DataFrame(season_stats)

# Adding a caption for the table
print("\nTable 2: Seasonal Statistics of Temperature, Precipitation, and Wind Speed")
print(tabulate(season_stats_df, headers='keys', tablefmt='grid'))

# Plotting grouped bar chart for temperature, precipitation, and wind speed
x = np.arange(len(season_stats_df['Season']))  # the label locations
width = 0.25  # the width of the bars

fig, ax = plt.subplots(figsize=(14, 8))

# Plotting offset bars for temperature, precipitation, and wind speed
temp_bars = ax.bar(x - width, season_stats_df['Mean Temp (°C)'], width, label='Mean Temp', c
precip_bars = ax.bar(x, season_stats_df['Mean Precip (mm)'], width, label='Mean Precip', col
```

```
wspd_bars = ax.bar(x + width, season_stats_df['Mean Wind Speed (m/s)'], width, label='Mean W

# Add labels, title, and legend
ax.set_xlabel('Season')
ax.set_ylabel('Values')
ax.set_title('Seasonal Temperature, Precipitation, and Wind Speed Statistics')
ax.set_xticks(x)
ax.set_xticklabels(season_stats_df['Season'])
ax.legend()

# Annotate bar values for each variable
for bar in temp_bars:
    ax.annotate(f'{bar.get_height():.1f}', xy=(bar.get_x() + bar.get_width() / 2, bar.get_he
                xytext=(0, 3), textcoords="offset points", ha='center', fontsize=9)
for bar in precip_bars:
    ax.annotate(f'{bar.get_height():.1f}', xy=(bar.get_x() + bar.get_width() / 2, bar.get_he
                xytext=(0, 3), textcoords="offset points", ha='center', fontsize=9)
for bar in wspd_bars:
    ax.annotate(f'{bar.get_height():.1f}', xy=(bar.get_x() + bar.get_width() / 2, bar.get_he
                xytext=(0, 3), textcoords="offset points", ha='center', fontsize=9)

# Adding a table to the figure
table_data = season_stats_df.set_index('Season').round(2)  # Format values to 2 decimal poir
table = plt.table(cellText=table_data.values,
                  colLabels=table_data.columns,
                  rowLabels=table_data.index,
                  cellLoc='center',
                  loc='bottom',
                  bbox=[0, -0.5, 1, 0.3])  # Adjust bbox for positioning

table.auto_set_font_size(False)
table.set_fontsize(8)

# Adding a comprehensive caption below the figure and table
caption_text = (
    "Figure 8 and Table 2: Seasonal Temperature, Precipitation, and Wind Speed Statistics\n"
    "The figure shows the average temperature (°C), precipitation (mm), and wind speed (m/s)
    "The table below summarizes detailed seasonal statistics, including minimum, maximum, me
    "The data is derived from Tehran's weather dataset, highlighting seasonal variations in 
)

fig.text(0.5, -0.01, caption_text, ha='center', fontsize=14, wrap=True)

# Save the figure as a PDF
output_path = r'E:\CompAppTools\Project\Seasonal_Statistics.pdf'
plt.savefig(output_path, format='pdf', bbox_inches='tight')

fig.tight_layout()
plt.show()
```