Step 1: Import Required Libraries

In [1]:
```
# Import necessary libraries

import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import randint
from datetime import timedelta
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
```

Step 2: Load Data

In [2]:
```
# Load climate data

folder_path = r'D:/Pasture Growth/datasets/'

climate_data = pd.concat([
    pd.read_csv(f'{folder_path}climate_{year}.csv') for year in range(2017,
])

climate_data.head()
```

Out[2]:

	PADDOCK_ID	DATE	RAIN	MAX_TEMP	MIN_
0	E5BEE111190304432B4EEFB0B7FA23A7C23CF8960E4C7F...	2017-03-22	50.2	28.4	
1	4C2EB78791FD1F9A42113DA94A14C1190BD8FBB8E87F96...	2017-03-22	52.5	28.5	
2	9C2AE9804DFBFDFA6791C03D9FCE5DCBF9DA6AE7476477...	2017-03-22	52.5	28.5	
3	04C1E345BBF76EA3E2B076C7E1E17290AD77F6BD159AD7...	2017-03-22	52.5	28.5	
4	9F7943EC335457BE4BF1A456EC8AE9567902A02B4173C8...	2017-03-22	52.5	28.5	

In [3]:
```
climate_data.shape
```

Out[3]: (33276908, 9)

In [4]:
```
# Load TSDM data

tsdm_data = pd.read_csv(r'D:\Pasture Growth\datasets\nrm1010_tsdm.csv')

tsdm_data.head()
```

Out[4]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	5/01/2017	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	20/01/2017	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	4/02/2017	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	19/02/2017	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	6/03/2017	2330.000000

In [5]:
```
tsdm_data.shape
```

Out[5]: (1030320, 3)

In [6]:
```
# Load paddock metadata

paddock_metadata = pd.read_csv(r'D:\Pasture Growth\datasets\paddock_metadata

paddock_metadata.head()
```

Out[6]:

	PADDOCK_ID	CROP_TYPE	LANDSIZE_HA	PA
0	0AE10585CD37DE9A5DF983CF95D7237D534E6CE7EB6081...	natural grasses	29.90	
1	ED2AAAB2838D848AC4E7BDE8BA13941B7105ECF65DF3F4...	natural grasses	16.15	
2	435141E0BE1DBFD4E37DC6E374B4BD4C6D91CFC722E73A...	natural grasses	8.27	
3	AD6846319B84A902CB0B71B3F640643FCDA283D84B2DE8...	natural grasses	22.07	
4	7A85B8453613D37697502D75F3DBD922FBEE91FC6AB8A8...	natural grasses	0.82	

In [7]:
```
paddock_metadata.shape
```

Out[7]: (17966, 5)

Step 3: Data Preprocessing & EDA

In [8]:
```
# Convert DATE to datetime format
```

```
climate_data['DATE'] = pd.to_datetime(climate_data['DATE'], format='%Y-%m-%d

tsdm_data['OBSERVATION_DATE'] = pd.to_datetime(tsdm_data['OBSERVATION_DATE']
```

In [9]:
```
# Check unique values in PASTURE_STATE column

print(paddock_metadata['PASTURE_STATE'].unique())
```

```
['Grazing' 'Laneway' 'Yard' 'Cropping' 'Feedlot' 'Vegetation' 'Hay' 'Pen']
```

In [10]:
```
# Filter the DataFrame where PASTURE_STATE is 'Grazing'

paddock_metadata = paddock_metadata[paddock_metadata['PASTURE_STATE'] == 'Gr
```

In [11]:
```
print(paddock_metadata['PASTURE_STATE'].unique())
```

```
['Grazing']
```

In [12]:
```
paddock_metadata.shape
```

Out[12]:
```
(14956, 5)
```

In [13]:
```
# Convert PADDOCK_ID to lowercase for consistency

tsdm_data['PADDOCK_ID'] = tsdm_data['PADDOCK_ID'].str.lower()
climate_data['PADDOCK_ID'] = climate_data['PADDOCK_ID'].str.lower()
paddock_metadata['PADDOCK_ID'] = paddock_metadata['PADDOCK_ID'].str.lower()

# Strip spaces from PADDOCK_ID

tsdm_data['PADDOCK_ID'] = tsdm_data['PADDOCK_ID'].str.strip()
climate_data['PADDOCK_ID'] = climate_data['PADDOCK_ID'].str.strip()
paddock_metadata['PADDOCK_ID'] = paddock_metadata['PADDOCK_ID'].str.strip()

# Common PADDOCK_ID values across all three datasets

common_ids_tsdm_climate = set(tsdm_data['PADDOCK_ID']).intersection(set(clim
common_ids_all = common_ids_tsdm_climate.intersection(set(paddock_metadata['

# Number of common PADDOCK_IDs

print(f"Number of common PADDOCK_IDs between TSDM and Climate data: {len(com
print(f"Number of common PADDOCK_IDs across all datasets: {len(common_ids_al
```

```
Number of common PADDOCK_IDs between TSDM and Climate data: 4855
Number of common PADDOCK_IDs across all datasets: 4808
```

In [14]:
```
# Merge TSDM data with Climate data

merged_data = pd.merge(tsdm_data, climate_data,
                       left_on=['PADDOCK_ID', 'OBSERVATION_DATE'],
                       right_on=['PADDOCK_ID', 'DATE'],
                       how='inner')

# Merge with paddock metadata
```

```python
merged_data = pd.merge(merged_data, paddock_metadata,
                       on='PADDOCK_ID',
                       how='inner')

merged_data.head()
```

Out[14]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

In [15]:
```python
merged_data.shape
```

Out[15]:
```
(701968, 15)
```

In [16]:
```python
# Remove the DATE and CREATION_DATE column from the merged_data

merged_data.drop(columns=['DATE', 'CREATION_DATE'], inplace=True)

merged_data.columns
```

Out[16]:
```
Index(['PADDOCK_ID', 'OBSERVATION_DATE', 'TSDM_MEAN', 'RAIN', 'MAX_TEMP',
       'MIN_TEMP', 'RH_TMAX', 'RH_TMIN', 'EVAP', 'RADIATION', 'CROP_TYPE',
       'LANDSIZE_HA', 'PASTURE_STATE'],
      dtype='object')
```

In [17]:
```python
merged_data.head()
```

Out[17]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

In [18]:
```python
merged_data.shape
```

Out[18]:
```
(701968, 13)
```

In [19]:
```python
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 701968 entries, 0 to 701967
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   PADDOCK_ID        701968 non-null  object
 1   OBSERVATION_DATE  701968 non-null  datetime64[ns]
 2   TSDM_MEAN         687426 non-null  float64
 3   RAIN              701968 non-null  float64
 4   MAX_TEMP          701968 non-null  float64
 5   MIN_TEMP          701968 non-null  float64
 6   RH_TMAX           701968 non-null  float64
 7   RH_TMIN           701968 non-null  float64
 8   EVAP              701968 non-null  float64
 9   RADIATION         701968 non-null  float64
 10  CROP_TYPE         701092 non-null  object
 11  LANDSIZE_HA       701968 non-null  float64
 12  PASTURE_STATE     701968 non-null  object
dtypes: datetime64[ns](1), float64(9), object(3)
memory usage: 69.6+ MB
```

In [20]:

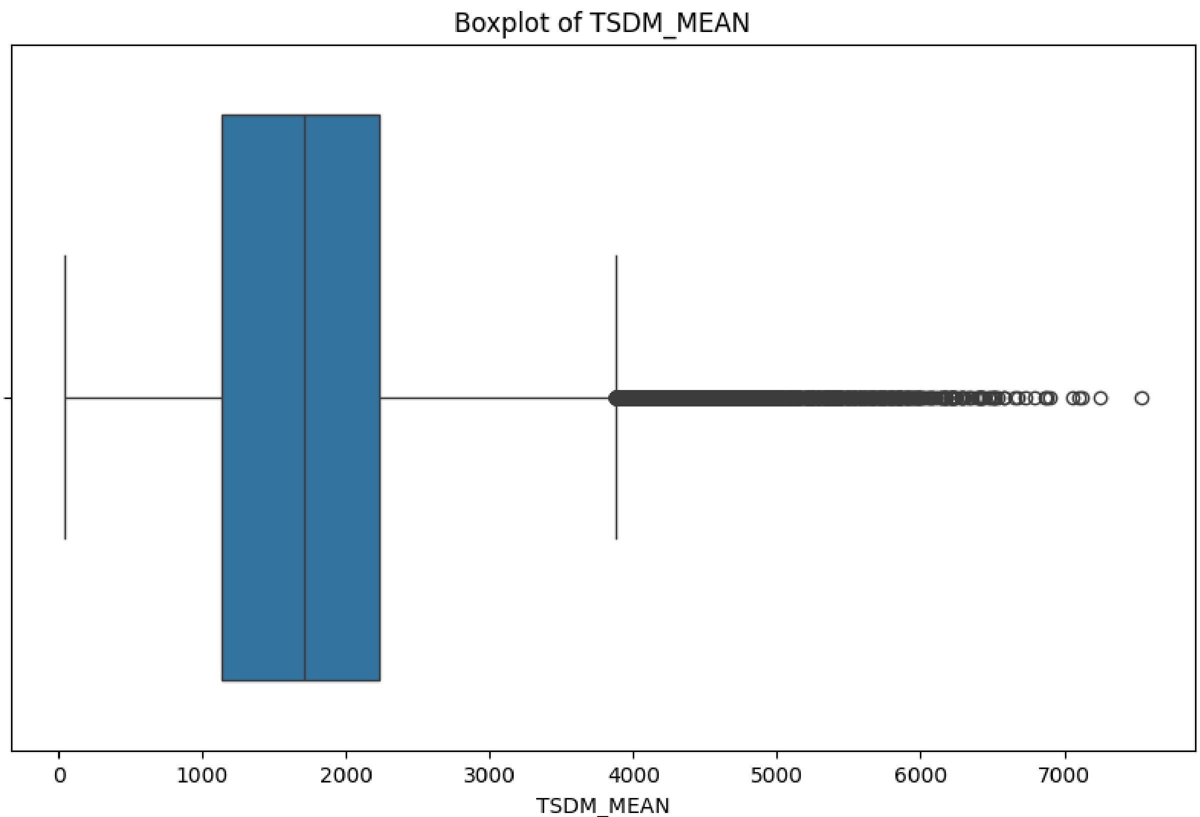
```
# Count of missing values in each column

merged_data.isnull().sum()
```

Out[20]:

```
PADDOCK_ID              0
OBSERVATION_DATE        0
TSDM_MEAN           14542
RAIN                    0
MAX_TEMP                0
MIN_TEMP                0
RH_TMAX                 0
RH_TMIN                 0
EVAP                    0
RADIATION               0
CROP_TYPE             876
LANDSIZE_HA             0
PASTURE_STATE           0
dtype: int64
```

In [21]:

```
# Check for outlier in TSDM_MEAN column

plt.figure(figsize=(10, 6))
sns.boxplot(x=merged_data['TSDM_MEAN'])
plt.title('Boxplot of TSDM_MEAN')
plt.show()
```



In [22]:
```
# Impute missing TSDM_MEAN values with the median

merged_data['TSDM_MEAN'].fillna(merged_data['TSDM_MEAN'].median(), inplace=T
```

In [23]:
```
# Count of missing values in each column

merged_data.isnull().sum()
```

Out[23]:
```
PADDOCK_ID            0
OBSERVATION_DATE      0
TSDM_MEAN             0
RAIN                  0
MAX_TEMP              0
MIN_TEMP              0
RH_TMAX               0
RH_TMIN               0
EVAP                  0
RADIATION             0
CROP_TYPE           876
LANDSIZE_HA           0
PASTURE_STATE         0
dtype: int64
```

In [24]:
```
# Impute missing CROP_TYPE values with the mode

merged_data['CROP_TYPE'].fillna(merged_data['CROP_TYPE'].mode()[0], inplace=
```

In [25]:
```
# Count of missing values in each column
```

```python
merged_data.isnull().sum()
```

Out[25]:

```
PADDOCK_ID          0
OBSERVATION_DATE    0
TSDM_MEAN           0
RAIN                0
MAX_TEMP            0
MIN_TEMP            0
RH_TMAX             0
RH_TMIN             0
EVAP                0
RADIATION           0
CROP_TYPE           0
LANDSIZE_HA         0
PASTURE_STATE       0
dtype: int64
```

In [26]:

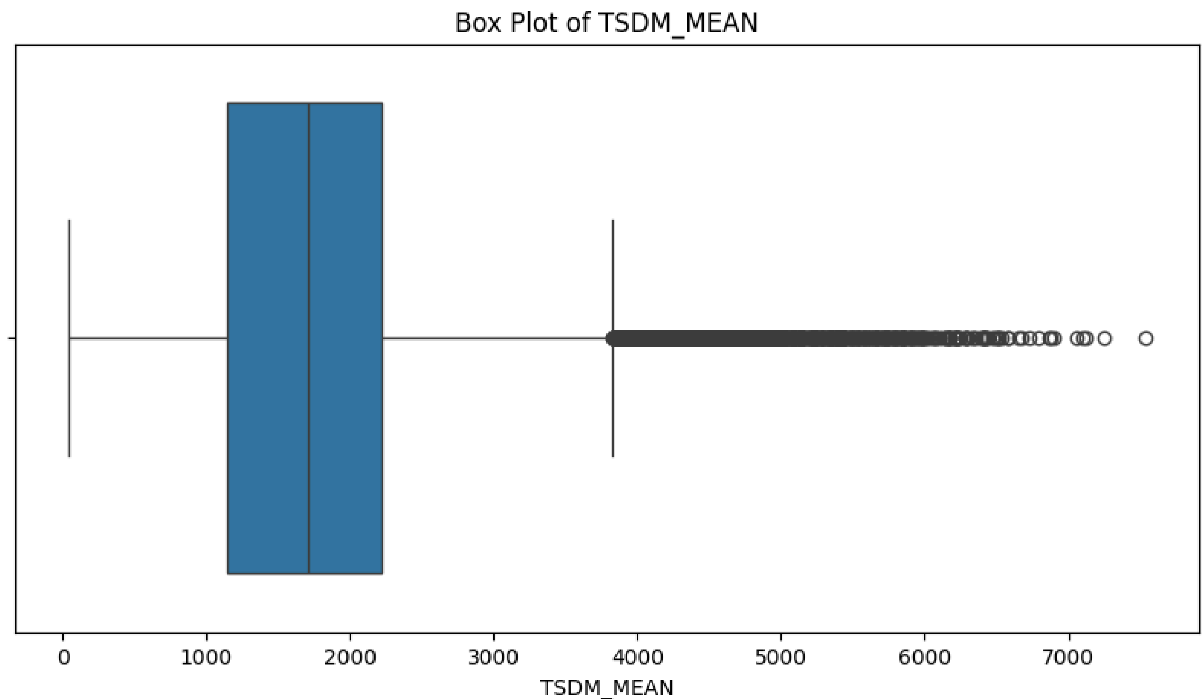
```python
# check for stats

merged_data.describe()
```

Out[26]:

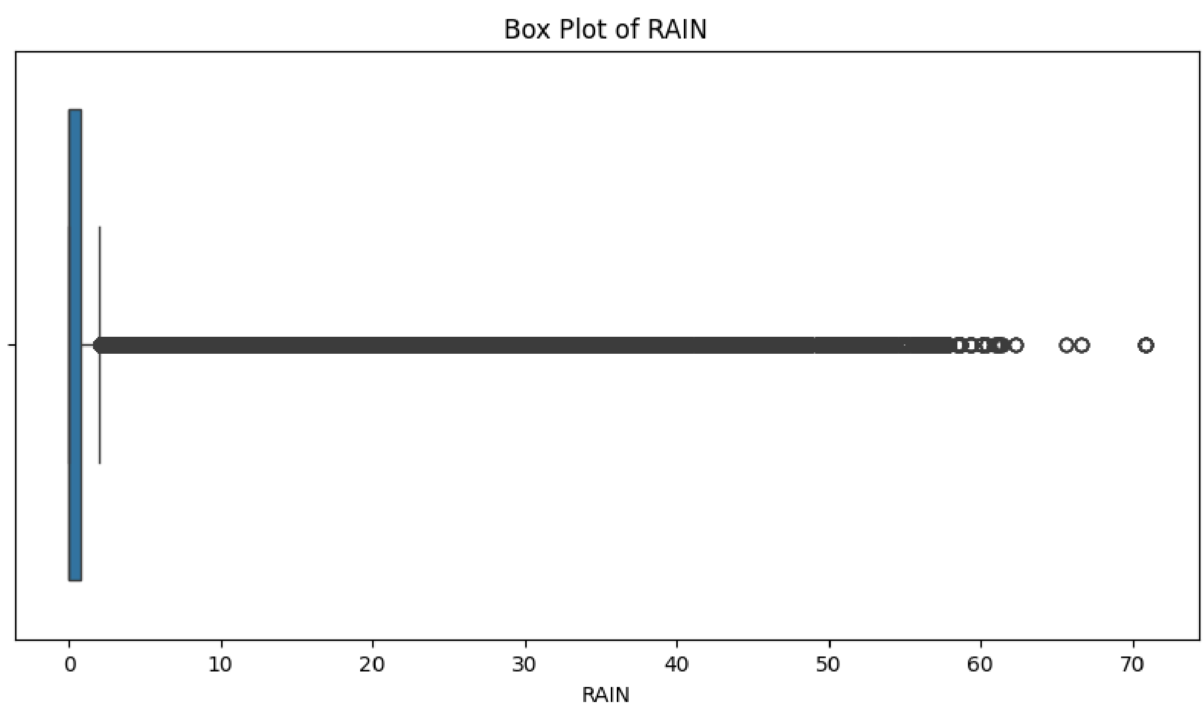
	OBSERVATION_DATE	TSDM_MEAN	RAIN	MAX_TEMP	MIN_TEMP
count	701968	701968.000000	701968.000000	701968.000000	701968.000000
mean	2019-12-28 12:00:00	1720.159833	1.979314	21.352619	8.203790
min	2017-01-05 00:00:00	41.833333	0.000000	2.500000	-6.000000
25%	2018-06-29 00:00:00	1151.583333	0.000000	15.500000	3.400000
50%	2019-12-28 12:00:00	1708.714286	0.000000	20.700000	7.400000
75%	2021-06-28 00:00:00	2223.285714	0.800000	26.900000	13.400000
max	2022-12-20 00:00:00	7532.500000	70.800000	45.600000	27.000000
std	NaN	695.954315	5.666996	7.597632	6.186689

In [27]:

```python
# Box Plot of TSDM_MEAN

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['TSDM_MEAN'])
plt.title('Box Plot of TSDM_MEAN')
plt.xlabel('TSDM_MEAN')
plt.show()
```

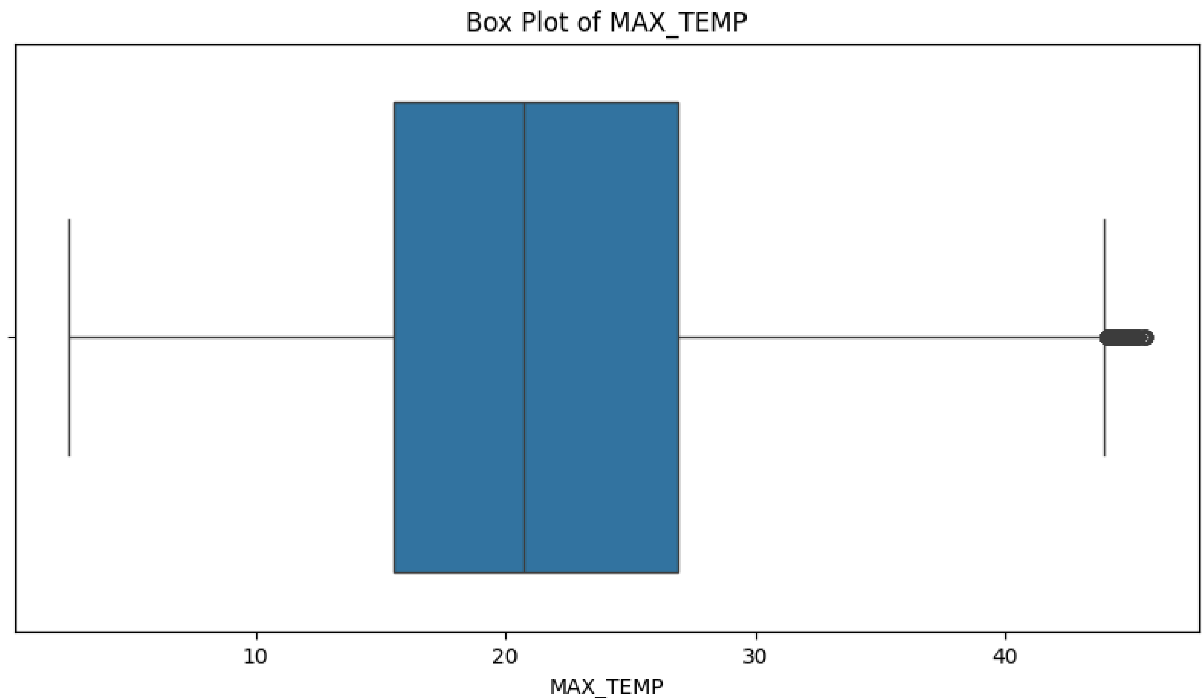


In [28]:
```
# Box Plot of RAIN

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['RAIN'])
plt.title('Box Plot of RAIN')
plt.xlabel('RAIN')
plt.show()
```



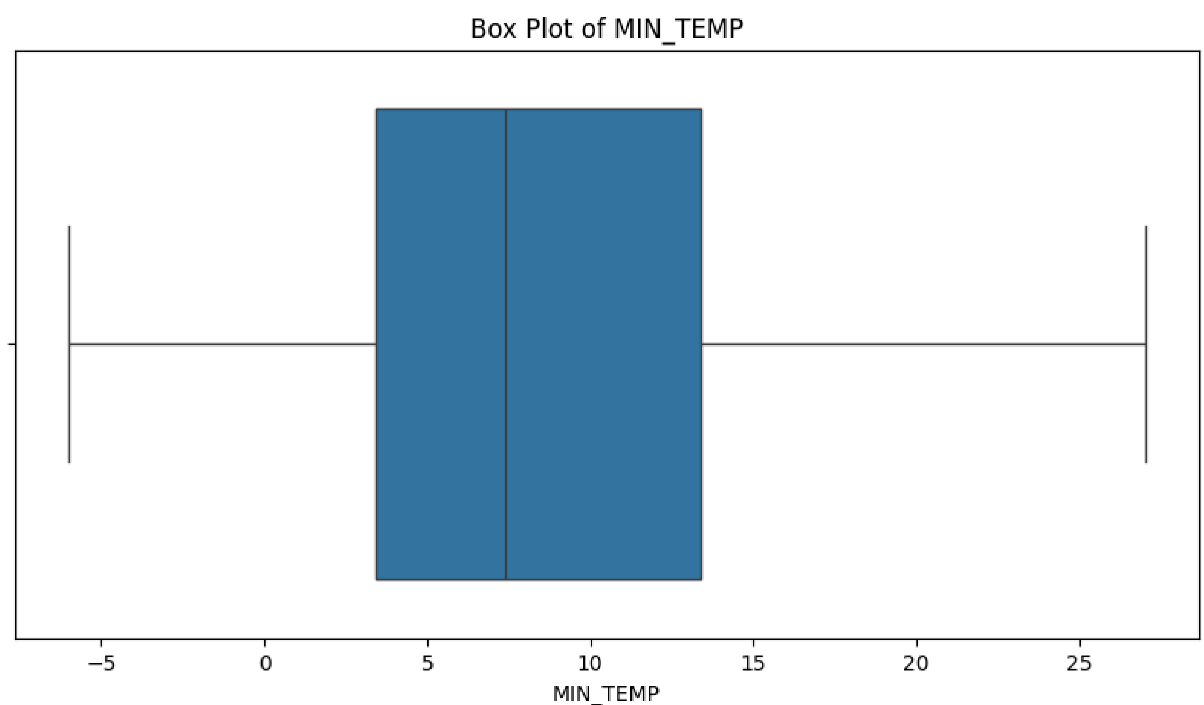
In [29]:
```
# Box Plot of MAX_TEMP

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['MAX_TEMP'])
```

```
plt.title('Box Plot of MAX_TEMP')
plt.xlabel('MAX_TEMP')
plt.show()
```

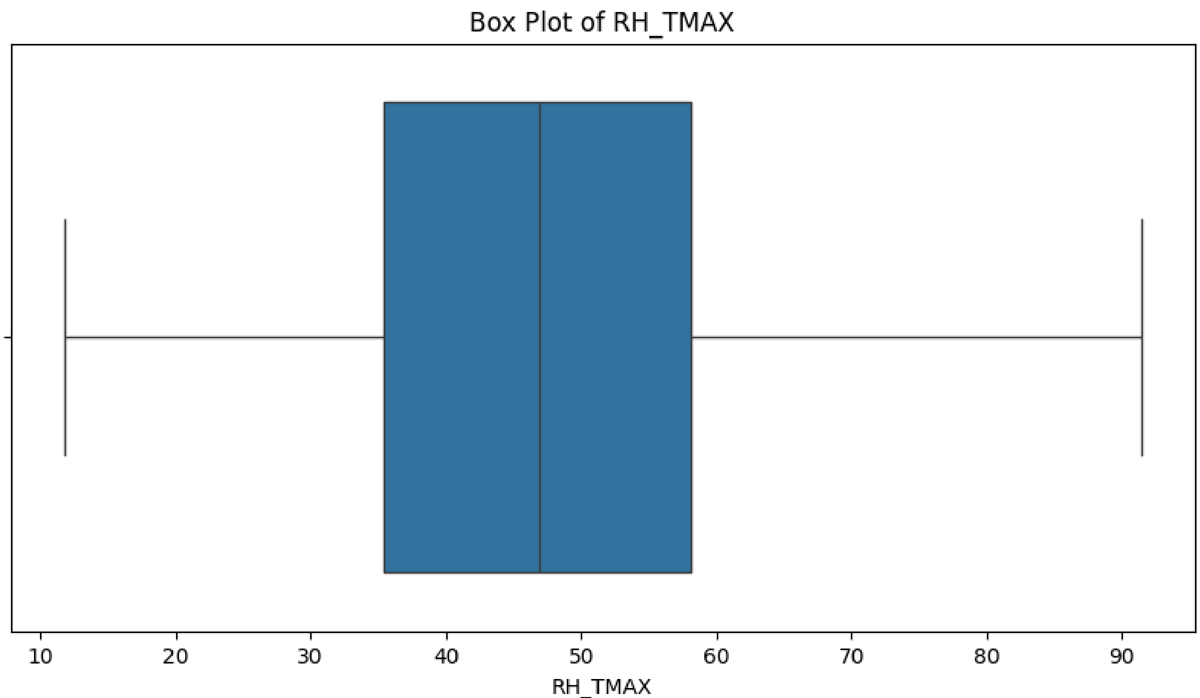


In [30]:

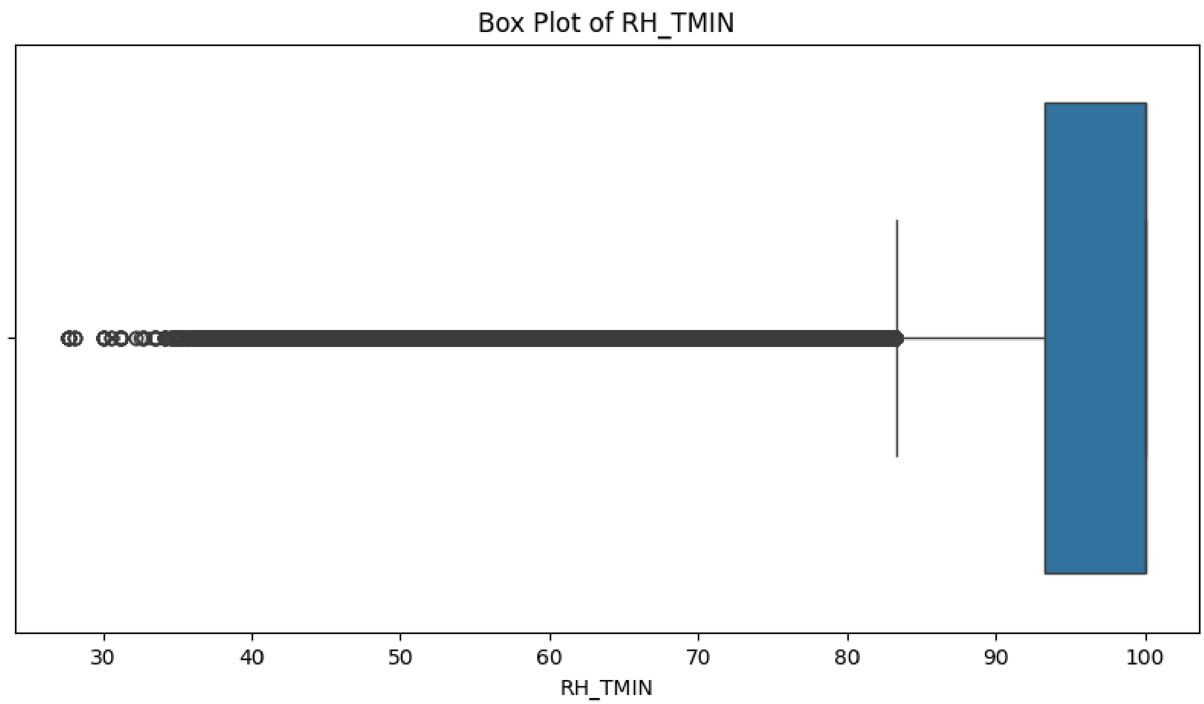
```
# Box Plot of MIN_TEMP

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['MIN_TEMP'])
plt.title('Box Plot of MIN_TEMP')
plt.xlabel('MIN_TEMP')
plt.show()
```



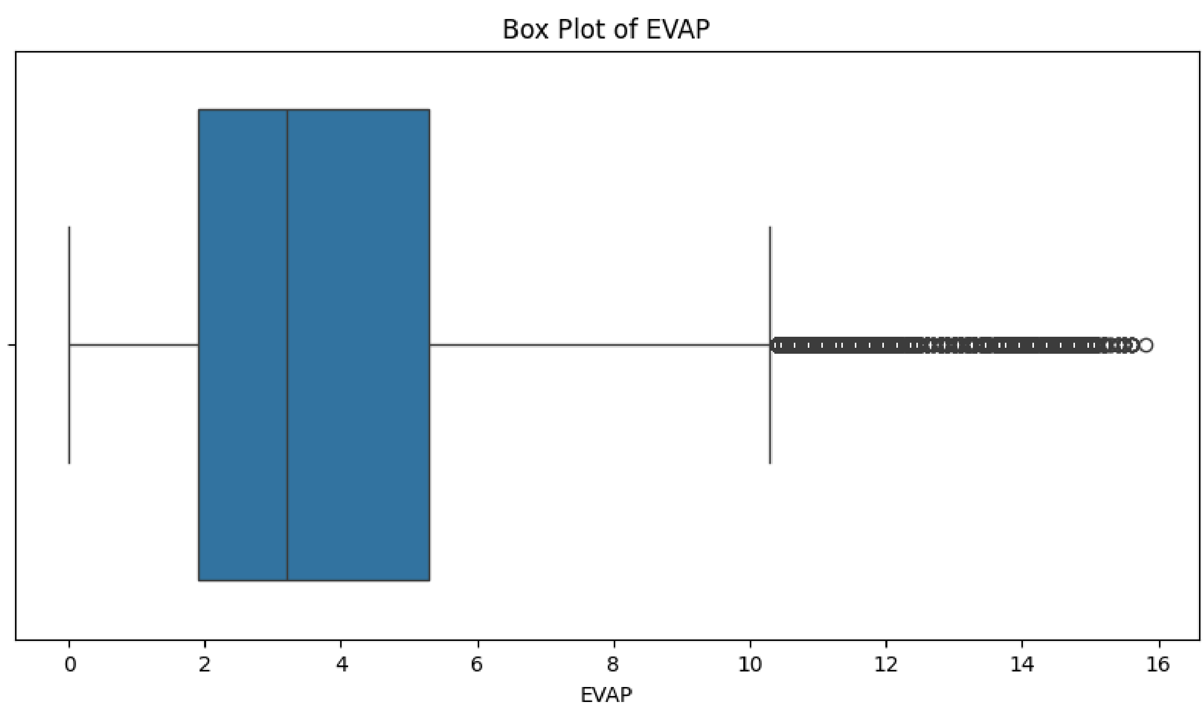
In [31]:
```
# Box Plot of RH_TMAX

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['RH_TMAX'])
plt.title('Box Plot of RH_TMAX')
plt.xlabel('RH_TMAX')
plt.show()
```



In [32]:
```
# Box Plot of RH_TMIN

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['RH_TMIN'])
plt.title('Box Plot of RH_TMIN')
plt.xlabel('RH_TMIN')
plt.show()
```

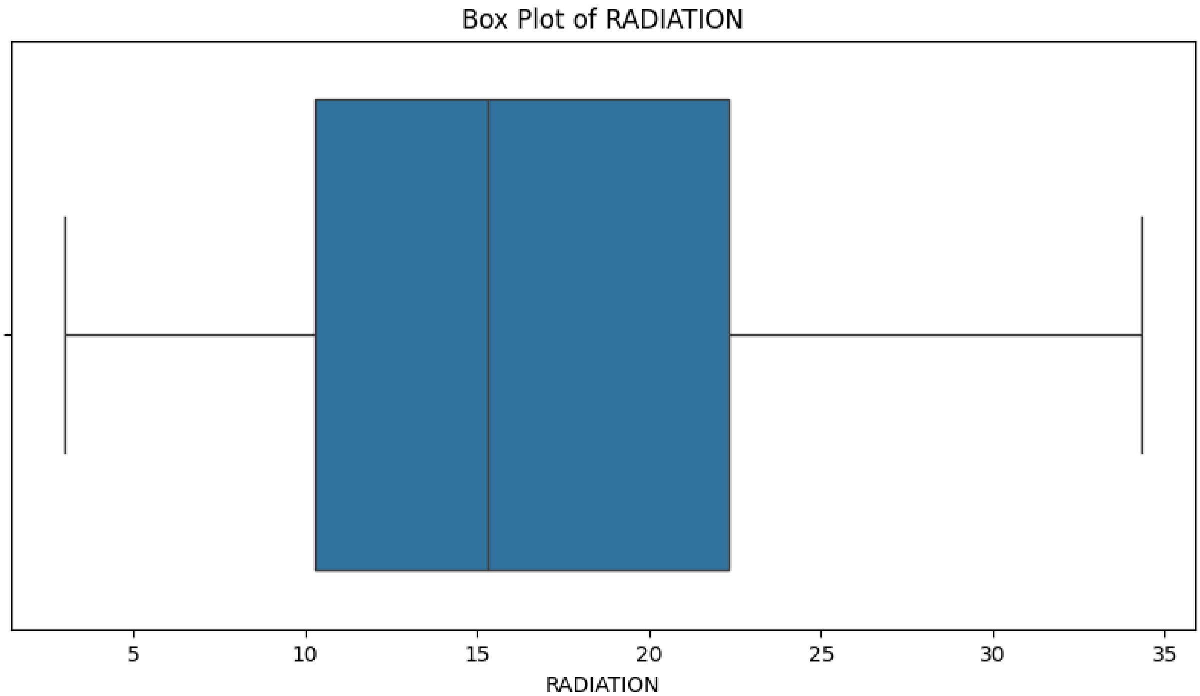


In [33]:
```
# Box Plot of EVAP

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['EVAP'])
plt.title('Box Plot of EVAP')
plt.xlabel('EVAP')
plt.show()
```



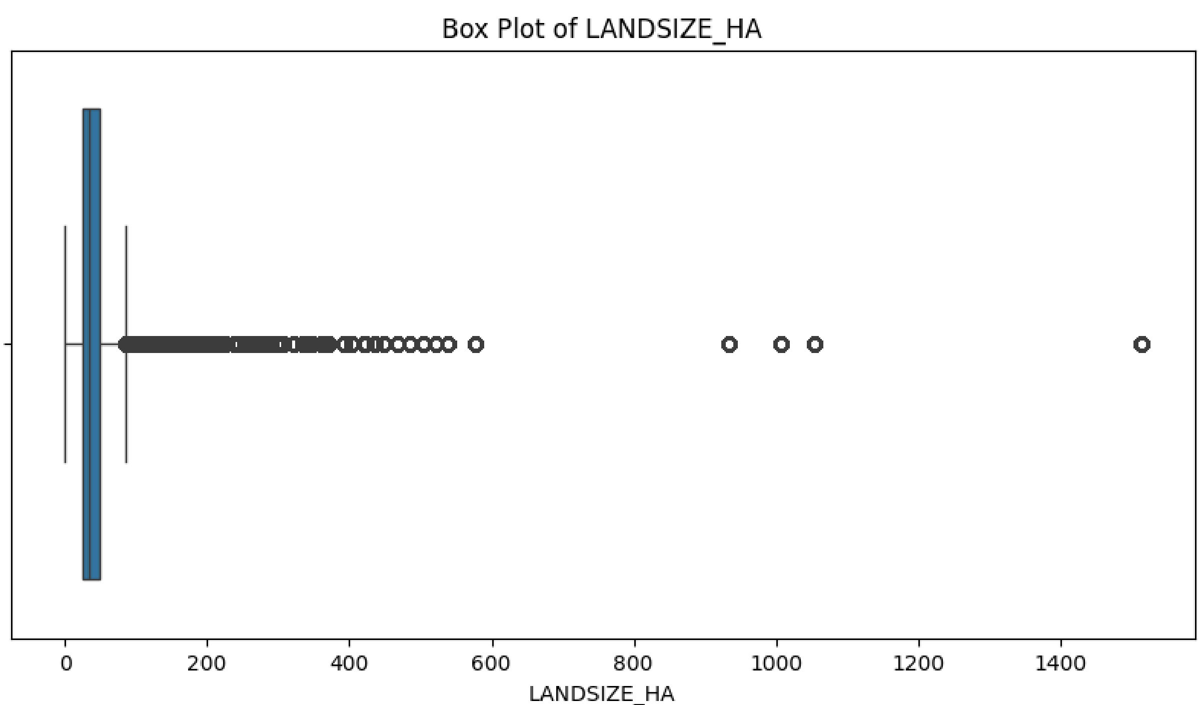
In [34]:
```
# Box Plot of RADIATION

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['RADIATION'])
```

```
plt.title('Box Plot of RADIATION')
plt.xlabel('RADIATION')
plt.show()
```



In [35]:

```
# Box Plot of LANDSIZE_HA

plt.figure(figsize=(10, 5))
sns.boxplot(x=merged_data['LANDSIZE_HA'])
plt.title('Box Plot of LANDSIZE_HA')
plt.xlabel('LANDSIZE_HA')
plt.show()
```



In [36]:
```
merged_data.describe()
```
Out[36]:

	OBSERVATION_DATE	TSDM_MEAN	RAIN	MAX_TEMP	MIN_TEMP
count	701968	701968.000000	701968.000000	701968.000000	701968.000000
mean	2019-12-28 12:00:00	1720.159833	1.979314	21.352619	8.203790
min	2017-01-05 00:00:00	41.833333	0.000000	2.500000	-6.000000
25%	2018-06-29 00:00:00	1151.583333	0.000000	15.500000	3.400000
50%	2019-12-28 12:00:00	1708.714286	0.000000	20.700000	7.400000
75%	2021-06-28 00:00:00	2223.285714	0.800000	26.900000	13.400000
max	2022-12-20 00:00:00	7532.500000	70.800000	45.600000	27.000000
std	NaN	695.954315	5.666996	7.597632	6.186689

In [37]:
```
merged_data.head()
```
Out[37]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

In [38]:
```
# Function to cap outliers using IQR

def cap_outliers_iqr(merged_data):

    exclude_cols = ['MIN_TEMP', 'RH_TMAX', 'RADIATION']

    # Iterate over each numerical column in the DataFrame
    for col in merged_data.select_dtypes(include=['float64']).columns:

        if col in exclude_cols:
            continue

        Q1 = merged_data[col].quantile(0.25)
        Q3 = merged_data[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        print(f"Column: {col}")
        print(f"  Q1: {Q1}")
        print(f"  Q3: {Q3}")
        print(f"  IQR: {IQR}")
        print(f"  Lower Bound: {lower_bound}")
```

```python
        print(f"  Upper Bound: {upper_bound}")

        # Cap the outliers
        merged_data[col] = merged_data[col].clip(lower=lower_bound, upper=up

    return merged_data

final_merge = cap_outliers_iqr(merged_data)
```

```
Column: TSDM_MEAN
  Q1: 1151.5833333333333
  Q3: 2223.285714285714
  IQR: 1071.702380952381
  Lower Bound: -455.9702380952383
  Upper Bound: 3830.839285714286
Column: RAIN
  Q1: 0.0
  Q3: 0.8
  IQR: 0.8
  Lower Bound: -1.2000000000000002
  Upper Bound: 2.0
Column: MAX_TEMP
  Q1: 15.5
  Q3: 26.9
  IQR: 11.399999999999999
  Lower Bound: -1.5999999999999979
  Upper Bound: 44.0
Column: RH_TMIN
  Q1: 93.3
  Q3: 100.0
  IQR: 6.700000000000003
  Lower Bound: 83.25
  Upper Bound: 110.05000000000001
Column: EVAP
  Q1: 1.9
  Q3: 5.3
  IQR: 3.4
  Lower Bound: -3.1999999999999997
  Upper Bound: 10.399999999999999
Column: LANDSIZE_HA
  Q1: 25.749104597749998
  Q3: 49.98700938925
  IQR: 24.2379047915
  Lower Bound: -10.607752589500002
  Upper Bound: 86.3438665765
```

In [39]:
```python
final_merge.head()
```

Out[39]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

In [40]:

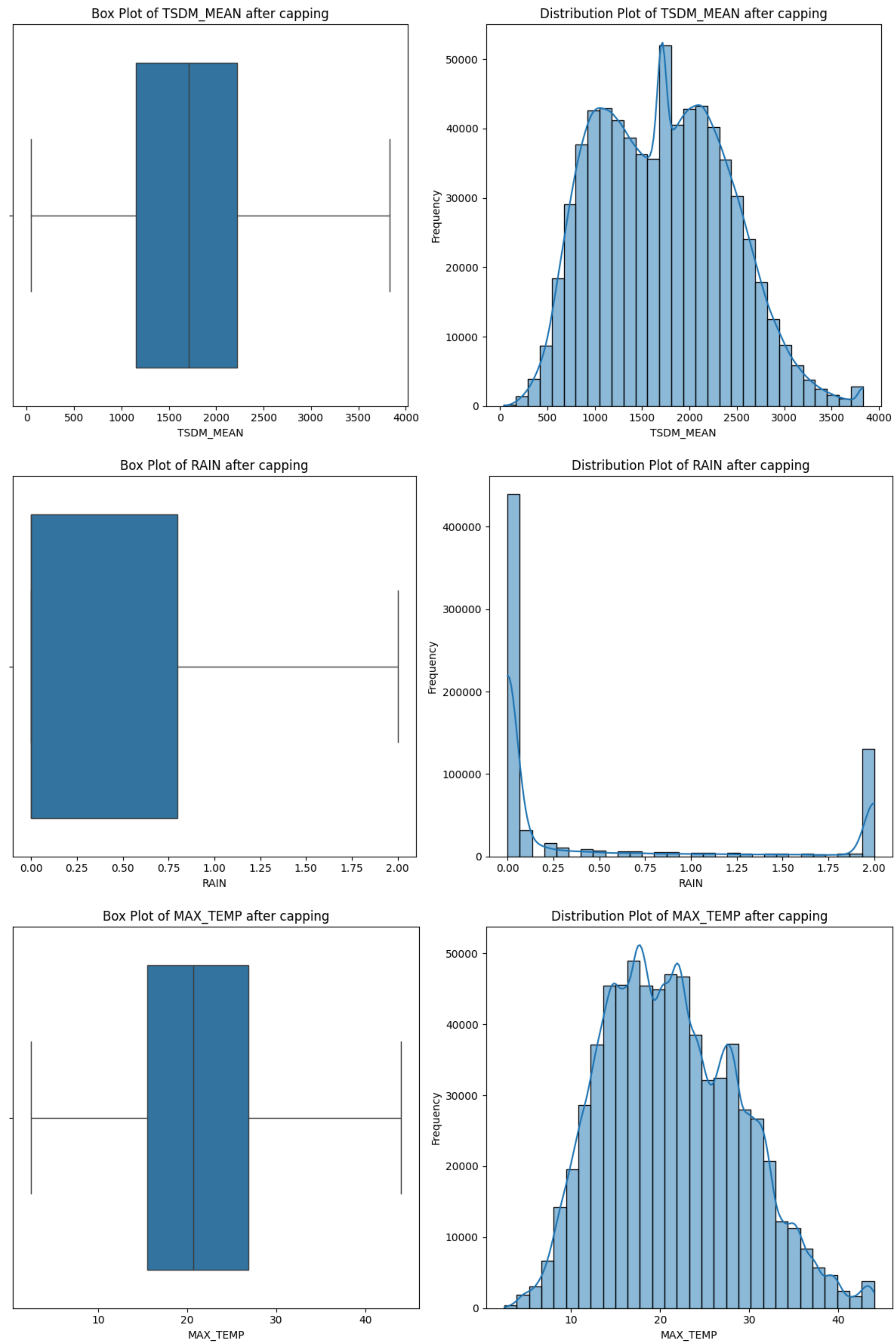
```
final_merge.describe()
```

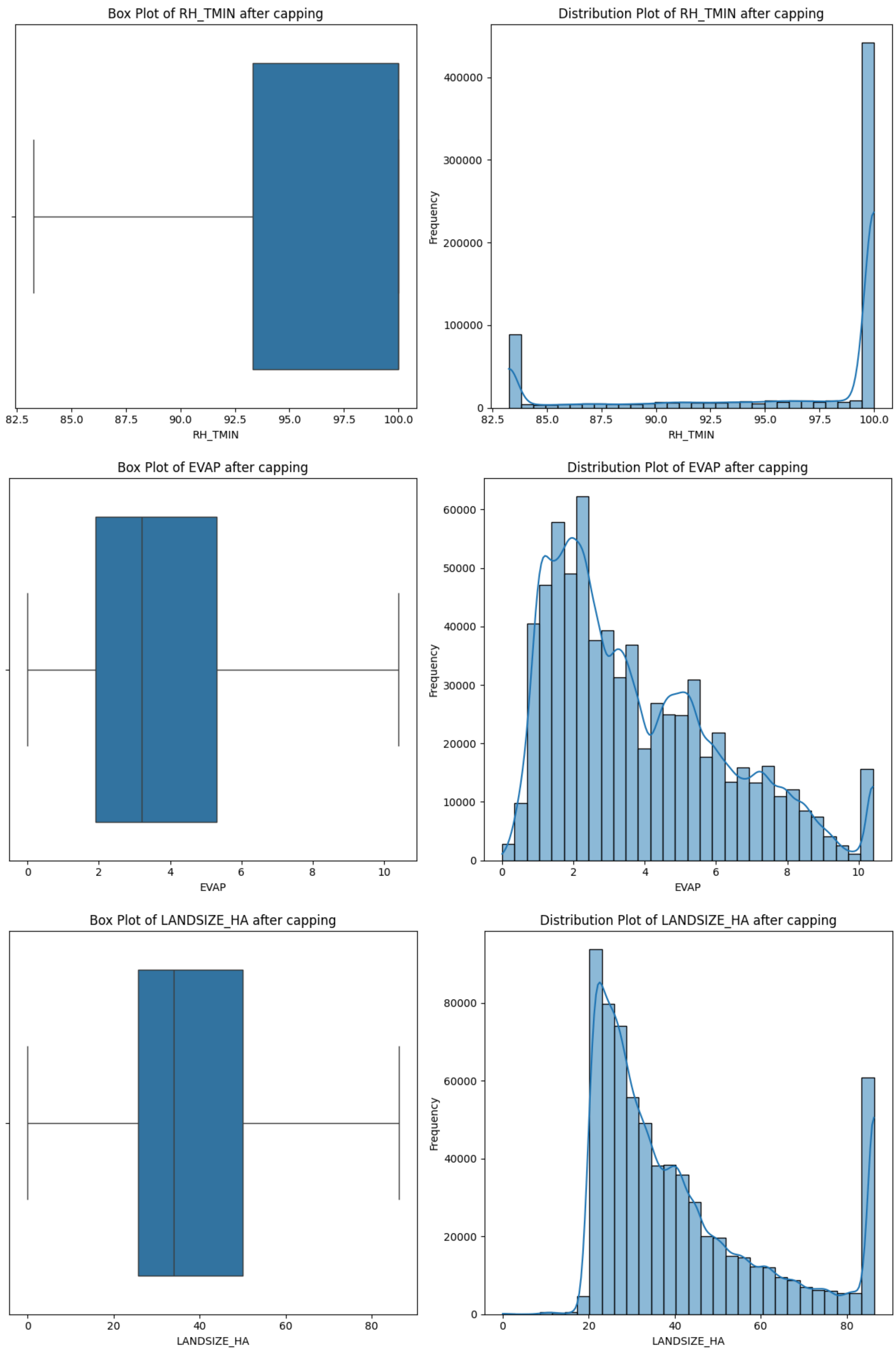
Out[40]:

	OBSERVATION_DATE	TSDM_MEAN	RAIN	MAX_TEMP	MIN_TEMP
count	701968	701968.000000	701968.000000	701968.000000	701968.000000
mean	2019-12-28 12:00:00	1718.284044	0.490724	21.352095	8.203790
min	2017-01-05 00:00:00	41.833333	0.000000	2.500000	-6.000000
25%	2018-06-29 00:00:00	1151.583333	0.000000	15.500000	3.400000
50%	2019-12-28 12:00:00	1708.714286	0.000000	20.700000	7.400000
75%	2021-06-28 00:00:00	2223.285714	0.800000	26.900000	13.400000
max	2022-12-20 00:00:00	3830.839286	2.000000	44.000000	27.000000
std	NaN	688.505806	0.795765	7.596043	6.186689

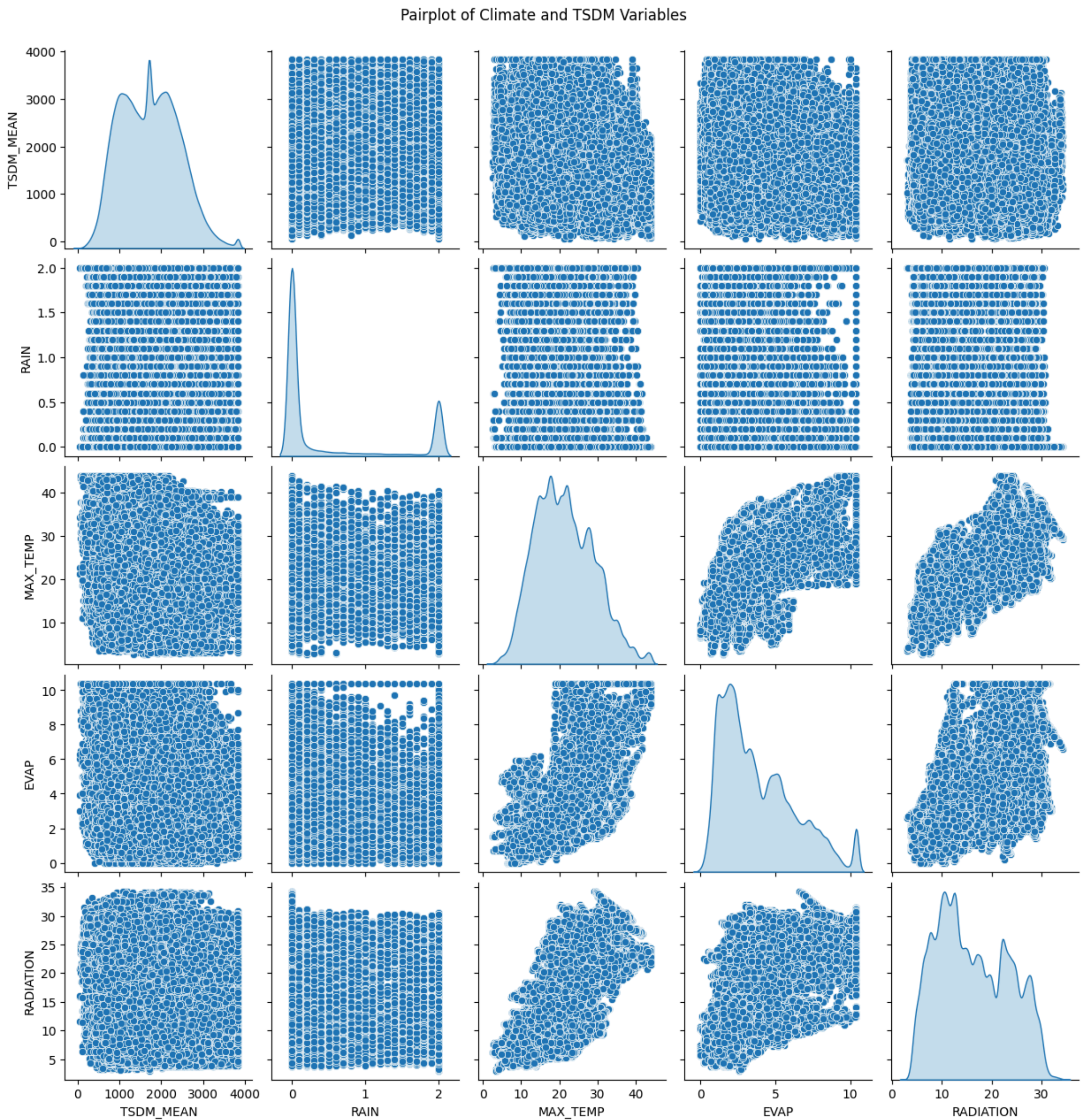
In [41]:

```
# Columns to plot

columns_to_plot = ['TSDM_MEAN', 'RAIN', 'MAX_TEMP', 'RH_TMIN', 'EVAP', 'LAND

# Box plots and distribution plots after capping

for column in columns_to_plot:
    plt.figure(figsize=(12, 6))

    # Box Plot
    plt.subplot(1, 2, 1)
    sns.boxplot(x=final_merge[column])
    plt.title(f'Box Plot of {column} after capping')
    plt.xlabel(column)

    # Distribution Plot
    plt.subplot(1, 2, 2)
    sns.histplot(final_merge[column], kde=True, bins=30)
    plt.title(f'Distribution Plot of {column} after capping')
    plt.xlabel(column)
    plt.ylabel('Frequency')

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


Box Plot of TSDM_MEAN after capping
0
500
1000
1500
2000
2500
3000
3500
4000
TSDM_MEAN
Distribution Plot of TSDM_MEAN after capping
Frequency
0
10000
20000
30000
40000
50000
0
500
1000
1500
2000
2500
3000
3500
4000
TSDM_MEAN
Box Plot of RAIN after capping
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
RAIN
Distribution Plot of RAIN after capping
Frequency
0
100000
200000
300000
400000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
RAIN
Box Plot of MAX_TEMP after capping
10
20
30
40
MAX_TEMP
Distribution Plot of MAX_TEMP after capping
Frequency
0
10000
20000
30000
40000
50000
10
20
30
40
MAX_TEMP


Box Plot of RH_TMIN after capping
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
RH_TMIN
Distribution Plot of RH_TMIN after capping
Frequency
0
100000
200000
300000
400000
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
RH_TMIN
Box Plot of EVAP after capping
0
2
4
6
8
10
EVAP
Distribution Plot of EVAP after capping
Frequency
0
10000
20000
30000
40000
50000
60000
0
2
4
6
8
10
EVAP
Box Plot of LANDSIZE_HA after capping
0
20
40
60
80
LANDSIZE_HA
Distribution Plot of LANDSIZE_HA after capping
Frequency
0
20000
40000
60000
80000
0
20
40
60
80
LANDSIZE_HA

In [42]:
```
# Pairplot

selected_columns = ['TSDM_MEAN', 'RAIN', 'MAX_TEMP', 'EVAP', 'RADIATION']
sns.pairplot(final_merge[selected_columns], diag_kind='kde')
plt.suptitle('Pairplot of Climate and TSDM Variables', y=1.02)
plt.show()
```



Step 4: Feature Engineering

4.1 Date Features

In [43]:
```
# Convert 'OBSERVATION_DATE' to datetime

final_merge['OBSERVATION_DATE'] = pd.to_datetime(final_merge['OBSERVATION_DA
```

```python
# Extracting features from the date

final_merge['Year'] = final_merge['OBSERVATION_DATE'].dt.year
final_merge['Month'] = final_merge['OBSERVATION_DATE'].dt.month
```

In [44]:

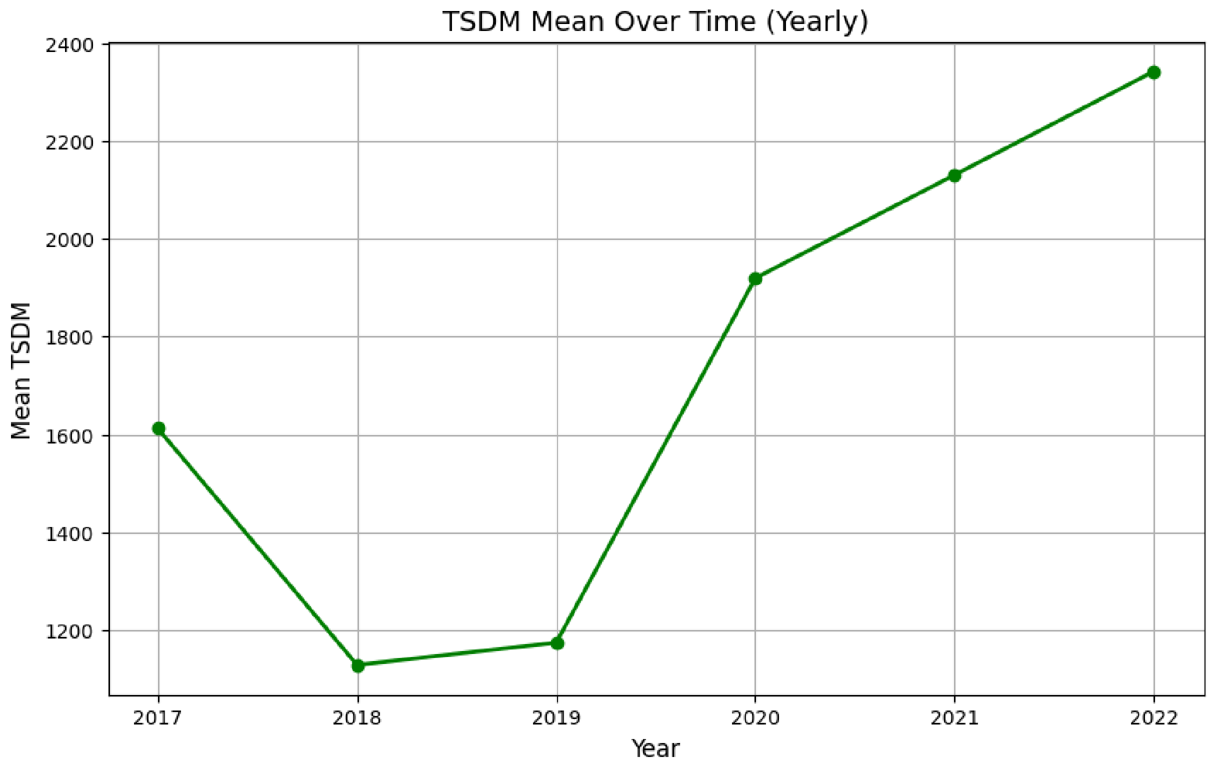
```python
final_merge.head()
```

Out[44]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

In [45]:

```python
# Group by year and calculate mean of TSDM_MEAN

yearly_tsdm_mean = final_merge.groupby('Year')['TSDM_MEAN'].mean()

# Line plot for TSDM Mean over time yearly

plt.figure(figsize=(10, 6))
plt.plot(yearly_tsdm_mean.index, yearly_tsdm_mean.values, marker='o', color=
plt.title('TSDM Mean Over Time (Yearly)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Mean TSDM', fontsize=12)
plt.grid(True)
plt.show()
```



4.2 Ratio Features

In [46]:
```
# Creating difference features

final_merge['Temp_Range'] = final_merge['MAX_TEMP'] - final_merge['MIN_TEMP'
```

In [47]:
```
final_merge.head()
```

Out[47]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

4.3 Interaction Features

In [48]:
```
# Interaction feature

final_merge['Temp_Rain_Interaction'] = final_merge['MAX_TEMP'] * final_merge
```

In [49]:
```
final_merge.head()
```

Out[49]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	1585.000000
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	1620.333333
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	1850.500000
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	2303.000000
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	2330.000000

4.4 Scaling Numerical Features

In [50]:

```
# Initialize the StandardScaler

scaler = StandardScaler()

# List of numerical columns to scale

columns_to_scale = ['TSDM_MEAN', 'RAIN', 'MAX_TEMP', 'MIN_TEMP', 'RH_TMAX',
                    'RH_TMIN', 'EVAP', 'RADIATION', 'LANDSIZE_HA',
                    'Temp_Range', 'Temp_Rain_Interaction']

# Scale the numerical columns

final_merge[columns_to_scale] = scaler.fit_transform(final_merge[columns_to_

print(final_merge[columns_to_scale].head())
```

```
   TSDM_MEAN      RAIN  MAX_TEMP  MIN_TEMP   RH_TMAX   RH_TMIN      EVAP  \
0  -0.193585 -0.616670  1.388606  1.486452 -0.772311 -1.798212  1.317150
1  -0.142266 -0.491005  1.270123  1.615762  0.267118  0.632316  2.100872
2   0.192033 -0.616670  2.362798  2.213821 -0.824609 -0.281171  0.822168
3   0.849254 -0.616670  1.019993  1.082358 -1.628696 -2.099989  2.265866
4   0.888469 -0.365339  1.112146  1.033867 -0.177418  0.632316  0.368434

   RADIATION  LANDSIZE_HA  Temp_Range  Temp_Rain_Interaction
0   1.473871    -0.285487    0.264898              -0.570730
1  -0.484690    -0.285487   -0.068259              -0.378665
2   0.924346    -0.285487    0.833226              -0.570730
3   0.698900    -0.285487    0.206106              -0.570730
4   0.698900    -0.285487    0.402081              -0.201470
```

In [51]:

```
final_merge.head()
```

Out[51]:

	PADDOCK_ID	OBSERVATION_DATE	TSDM_MEAN
0	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-05	-0.193585
1	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-01-20	-0.142266
2	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-04	0.192033
3	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-02-19	0.849254
4	deaef36d983eab83c6d11ad98ffdf0daab623cc0221043...	2017-03-06	0.888469

In [52]:

```
final_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 701968 entries, 0 to 701967
Data columns (total 17 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   PADDOCK_ID             701968 non-null  object
 1   OBSERVATION_DATE       701968 non-null  datetime64[ns]
 2   TSDM_MEAN              701968 non-null  float64
 3   RAIN                   701968 non-null  float64
 4   MAX_TEMP               701968 non-null  float64
 5   MIN_TEMP               701968 non-null  float64
 6   RH_TMAX                701968 non-null  float64
 7   RH_TMIN                701968 non-null  float64
 8   EVAP                   701968 non-null  float64
 9   RADIATION              701968 non-null  float64
 10  CROP_TYPE              701968 non-null  object
 11  LANDSIZE_HA            701968 non-null  float64
 12  PASTURE_STATE          701968 non-null  object
 13  Year                   701968 non-null  int32
 14  Month                  701968 non-null  int32
 15  Temp_Range             701968 non-null  float64
 16  Temp_Rain_Interaction  701968 non-null  float64
dtypes: datetime64[ns](1), float64(11), int32(2), object(3)
memory usage: 85.7+ MB
```

In [53]:

```
final_merge.isnull().sum()
```

Out[53]:
```
PADDOCK_ID                0
OBSERVATION_DATE          0
TSDM_MEAN                 0
RAIN                      0
MAX_TEMP                  0
MIN_TEMP                  0
RH_TMAX                   0
RH_TMIN                   0
EVAP                      0
RADIATION                 0
CROP_TYPE                 0
LANDSIZE_HA               0
PASTURE_STATE             0
Year                      0
Month                     0
Temp_Range                0
Temp_Rain_Interaction     0
dtype: int64
```

Step 5: Feature Selection

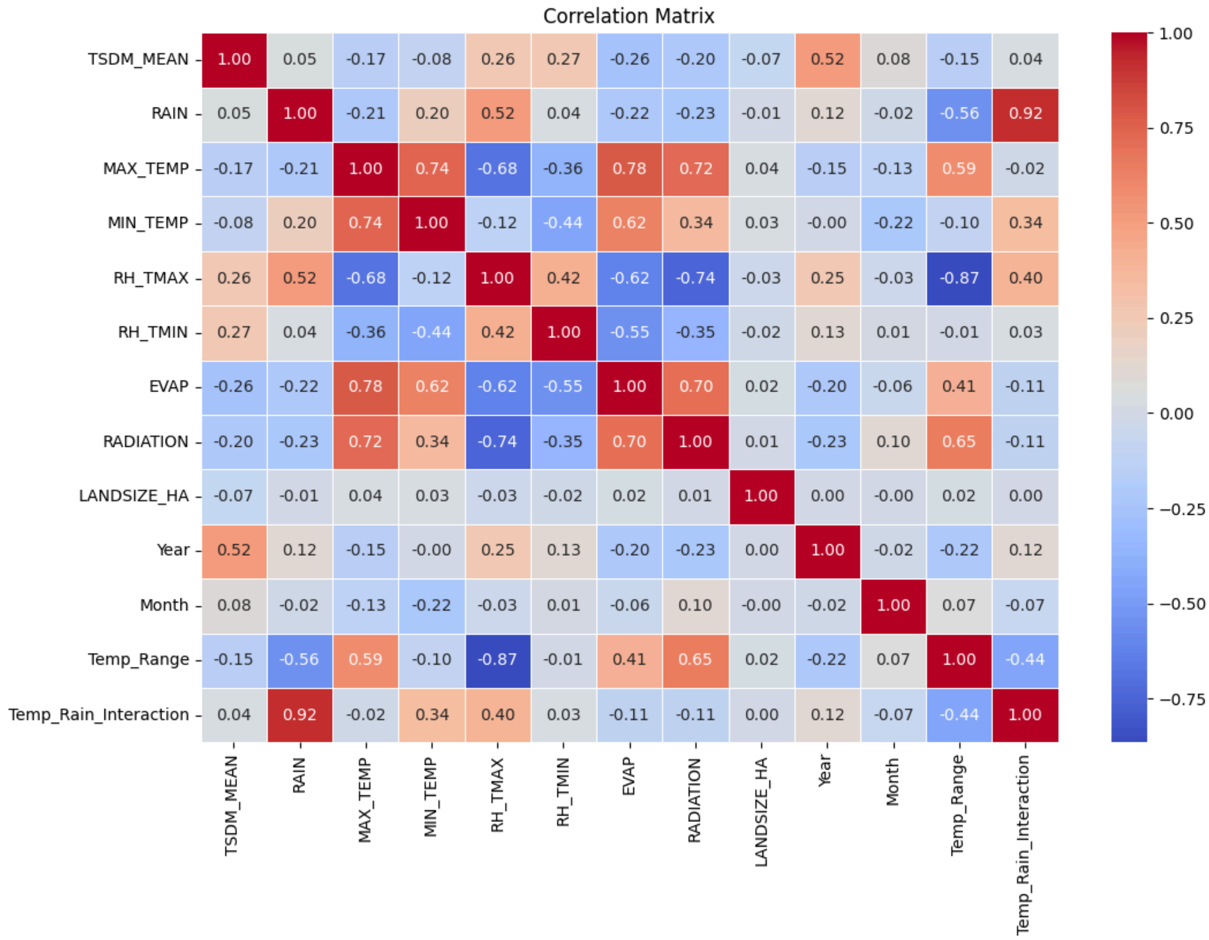
5.1 Correlation Matrix

In [54]:
```python
# Select only numeric columns for correlation analysis

numeric_data = final_merge.select_dtypes(include=['int32', 'float64'])

numeric_data.head()
```

Out[54]:

	TSDM_MEAN	RAIN	MAX_TEMP	MIN_TEMP	RH_TMAX	RH_TMIN	EVAP	RADIA
0	-0.193585	-0.616670	1.388606	1.486452	-0.772311	-1.798212	1.317150	1.47
1	-0.142266	-0.491005	1.270123	1.615762	0.267118	0.632316	2.100872	-0.48
2	0.192033	-0.616670	2.362798	2.213821	-0.824609	-0.281171	0.822168	0.92
3	0.849254	-0.616670	1.019993	1.082358	-1.628696	-2.099989	2.265866	0.69
4	0.888469	-0.365339	1.112146	1.033867	-0.177418	0.632316	0.368434	0.69

In [55]:
```python
# Calculate the correlation matrix

correlation_matrix = numeric_data.corr()

# Plotting the heatmap

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", line
plt.title('Correlation Matrix')
plt.show()
```




```
In [56]: target_correlations = correlation_matrix['TSDM_MEAN'].sort_values(ascending=

         print("Feature Correlations with TSDM_MEAN:")

         print(target_correlations)
```

```
Feature Correlations with TSDM_MEAN:
TSDM_MEAN                1.000000
Year                     0.519108
RH_TMIN                  0.268196
RH_TMAX                  0.260854
Month                    0.081826
RAIN                     0.045899
Temp_Rain_Interaction    0.040114
LANDSIZE_HA             -0.072778
MIN_TEMP                -0.081271
Temp_Range              -0.147897
MAX_TEMP                -0.165543
RADIATION               -0.197527
EVAP                    -0.261396
Name: TSDM_MEAN, dtype: float64
```

Step 6: Modelling

In [57]:
```python
# Target variable

y = numeric_data['TSDM_MEAN']  # Target variable

# Selected features

selected_feature = [
    'Year',
    'EVAP',
    'RADIATION',
    'Temp_Range',
    'RH_TMAX',
    'MIN_TEMP',
    'MAX_TEMP',
]

X_selected_df = numeric_data[selected_feature]

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_selected_df, y, test_s
```

6.1 Random Forest Regressor

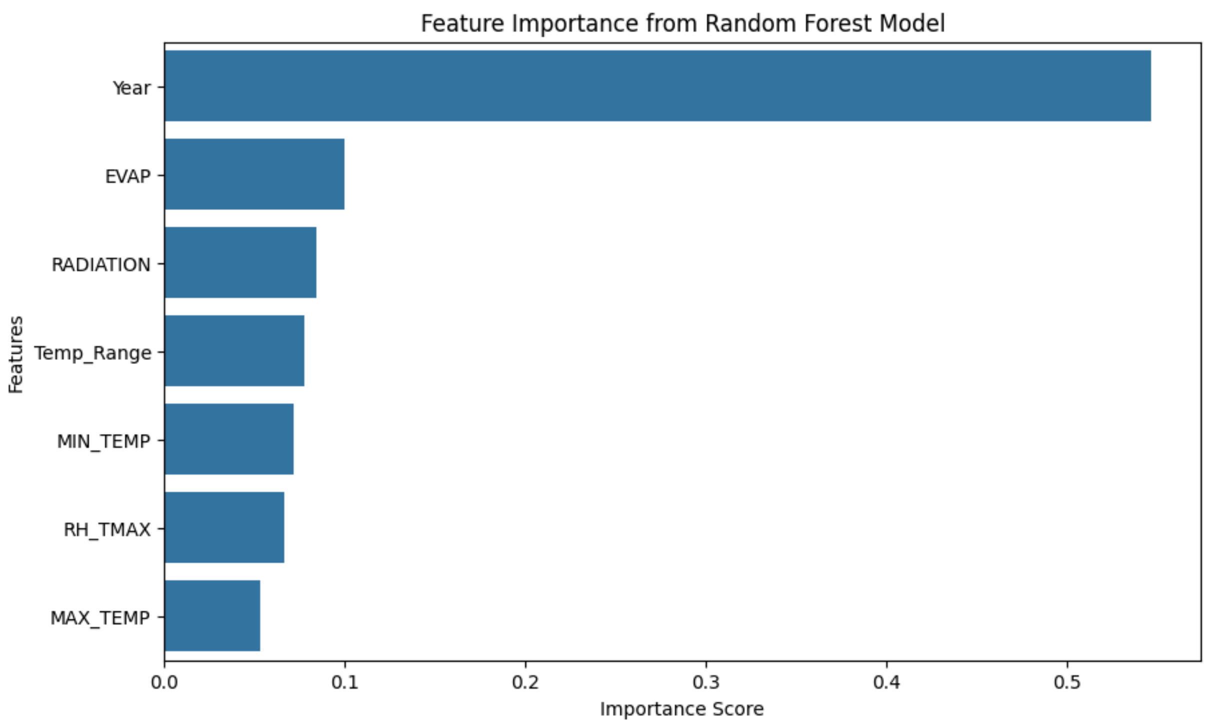
In [58]:
```python
# Initialize and train the Random Forest Regressor

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions

y_pred = rf_model.predict(X_test)
```

In [59]:
```python
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE is the squa
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'R-squared (R²): {r2}')
```

```
Mean Squared Error (MSE): 0.2814475768773077
Root Mean Squared Error (RMSE): 0.5305163304529915
Mean Absolute Error (MAE): 0.38660633427178753
R-squared (R²): 0.7171190995832423
```

In [60]:
```python
# Feature importances from the trained model

importances = rf_model.feature_importances_
```

```
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': importances
})

# Sort the DataFrame by importance

feature_importance_df = feature_importance_df.sort_values(by='Importance', a

plt.figure(figsize=(10, 6))

# Bar plot

sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance from Random Forest Model')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



6.2 XGBoost Regressor

In [61]:
```
# Initialize and train the XGBoost Regressor

xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)

# Make predictions

y_pred = xgb_model.predict(X_test)
```

In [62]:
```
# Evaluate the model
```

```
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE is the squa
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics

print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'R-squared (R²): {r2}')
```

```
Mean Squared Error (MSE): 0.33476189530268
Root Mean Squared Error (RMSE): 0.5785861174472474
Mean Absolute Error (MAE): 0.4348989154947407
R-squared (R²): 0.6635332681875447
```

6.3 Save Random Forest Model

In [63]:
```
# Save the model

with open(r'D:\Pasture Growth\datasets\random_forest_model.pkl', 'wb') as fi
    pickle.dump(rf_model, file)
```

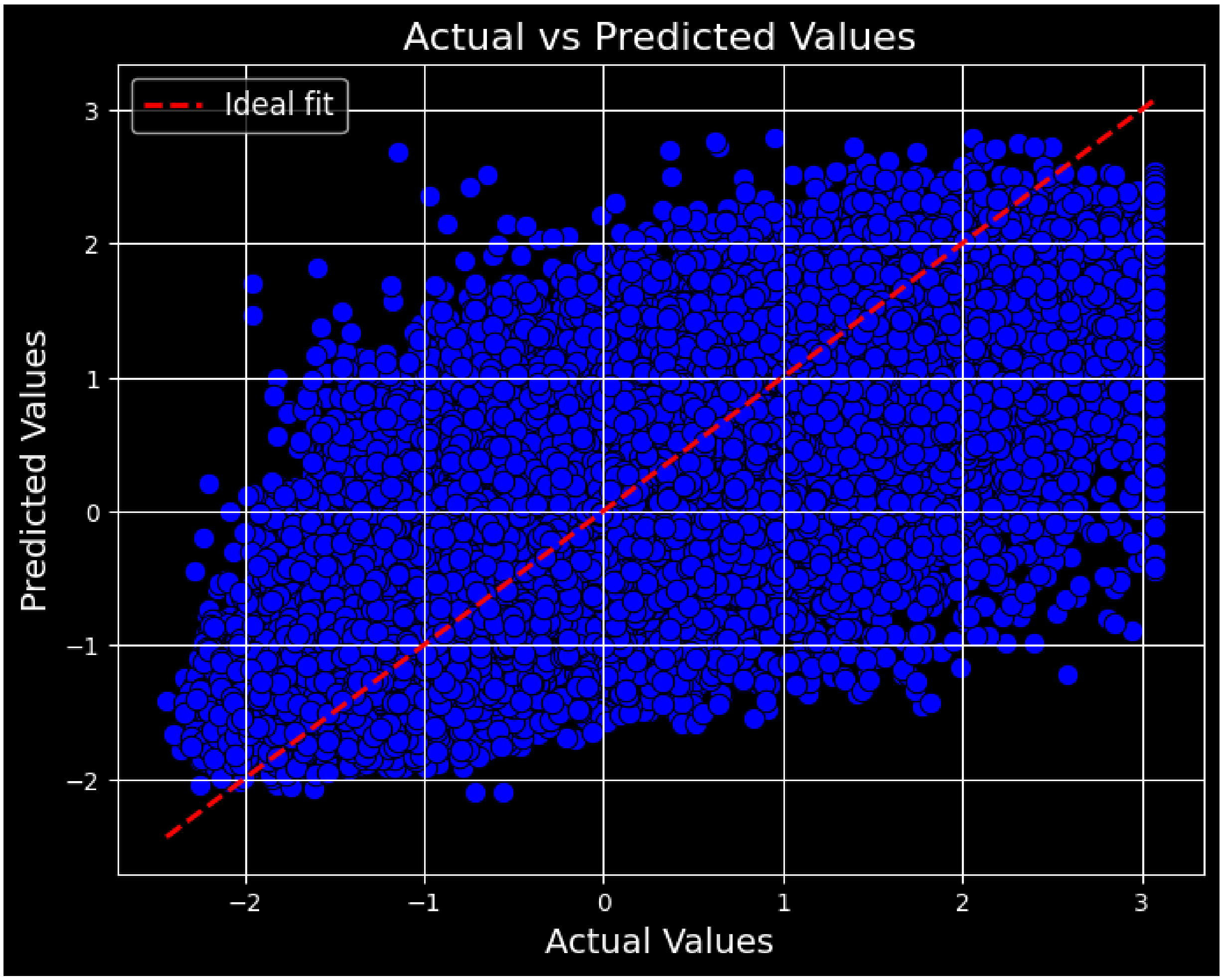
In [64]:
```
# Load the model

with open(r'D:\Pasture Growth\datasets\random_forest_model.pkl', 'rb') as fi
    loaded_rf_model = pickle.load(file)

# Loaded model to make predictions

y_pred_loaded = loaded_rf_model.predict(X_test)
```

In [65]:
```
# Evaluate the loaded model

mse_loaded = mean_squared_error(y_test, y_pred_loaded)
rmse_loaded = mean_squared_error(y_test, y_pred_loaded, squared=False)  # RM
mae_loaded = mean_absolute_error(y_test, y_pred_loaded)
r2_loaded = r2_score(y_test, y_pred_loaded)

# Print evaluation metrics

print(f'Mean Squared Error (MSE): {mse_loaded}')
print(f'Root Mean Squared Error (RMSE): {rmse_loaded}')
print(f'Mean Absolute Error (MAE): {mae_loaded}')
print(f'R-squared (R²): {r2_loaded}')
```

```
Mean Squared Error (MSE): 0.2814475768773077
Root Mean Squared Error (RMSE): 0.5305163304529915
Mean Absolute Error (MAE): 0.38660633427178753
R-squared (R²): 0.7171190995832423
```

Scatter plot of actual vs predicted values

In [66]:
```
plt.style.use('dark_background')

# Scatter plot of actual vs predicted values

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred_loaded, color='blue', s=80, alpha=1, edge
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red'
plt.xlabel('Actual Values', fontsize=14)
plt.ylabel('Predicted Values', fontsize=14)
plt.title('Actual vs Predicted Values', fontsize=16)
plt.legend(fontsize=12)
plt.grid(True)
plt.show()
```



Generating Future Data with Month-Based Historical Averages

In [67]:
```
# Extract the last OBSERVATION_DATE from final_merge

last_date = pd.to_datetime(final_merge['OBSERVATION_DATE']).max()

# Generate the next 6 months' dates from the last OBSERVATION_DATE

future_dates = pd.date_range(last_date + timedelta(days=30), periods=6, freq

# Create future_data with OBSERVATION_DATE, Year, and Month columns
```

```
future_data = pd.DataFrame({
    'OBSERVATION_DATE': future_dates,
    'Year': future_dates.year,
    'Month': future_dates.month
})

# Calculate month-based historical means for each feature from numeric_data

monthly_means = numeric_data.groupby('Month').mean()

# Populate future_data with month-based means

for feature in ['EVAP', 'RADIATION', 'Temp_Range', 'RH_TMAX', 'MIN_TEMP', 'M
    future_data[feature] = future_data['Month'].map(monthly_means[feature])

# Display future_data

print(future_data)
```

```
  OBSERVATION_DATE  Year  Month      EVAP  RADIATION  Temp_Range   RH_TMAX  \
0       2023-02-01  2023      2  1.013965   0.730433    0.138253 -0.443325
1       2023-03-01  2023      3  0.313775   0.185656    0.023387 -0.119301
2       2023-04-01  2023      4 -0.327623  -0.233968   -0.044214  0.037131
3       2023-05-01  2023      5 -0.909433  -0.770360   -0.121155  0.429327
4       2023-06-01  2023      6 -1.087614  -1.135736   -0.451711  0.686555
5       2023-07-01  2023      7 -0.913735  -1.036689   -0.543589  0.657553

   MIN_TEMP  MAX_TEMP
0  0.929178  0.849653
1  0.710950  0.594753
2 -0.044463 -0.065915
3 -0.719516 -0.667405
4 -0.908600 -1.043461
5 -0.895355 -1.094392
```

Predicting Future TSDM_MEAN Using Trained Model on Generated Data

In [68]:

```
# Select only the features required for prediction

X_future = future_data[selected_feature]

# Use the trained model to predict TSDM_MEAN for the next 6 months

future_predictions = loaded_rf_model.predict(X_future)

# Add the predictions to the future data DataFrame

future_data['TSDM_MEAN_Predicted'] = future_predictions

# Display the final DataFrame with future dates and predictions

print(future_data[['OBSERVATION_DATE', 'TSDM_MEAN_Predicted']])
```

```
  OBSERVATION_DATE  TSDM_MEAN_Predicted
0       2023-02-01             0.569451
1       2023-03-01             0.816637
2       2023-04-01             0.335159
3       2023-05-01             0.723748
4       2023-06-01             0.832515
5       2023-07-01             1.048704
```

In [69]:
```
print(future_data.columns)
```

```
Index(['OBSERVATION_DATE', 'Year', 'Month', 'EVAP', 'RADIATION', 'Temp_Rang
e',
       'RH_TMAX', 'MIN_TEMP', 'MAX_TEMP', 'TSDM_MEAN_Predicted'],
      dtype='object')
```

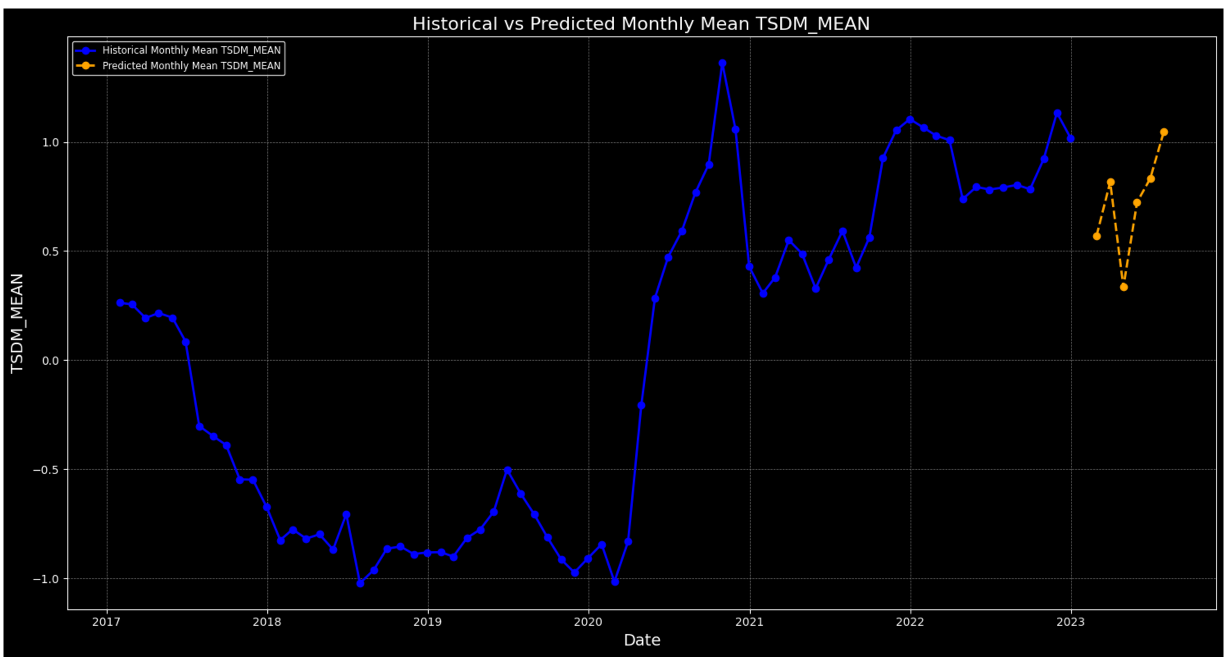
In [70]:
```
future_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   OBSERVATION_DATE     6 non-null      datetime64[ns]
 1   Year                 6 non-null      int32
 2   Month                6 non-null      int32
 3   EVAP                 6 non-null      float64
 4   RADIATION            6 non-null      float64
 5   Temp_Range           6 non-null      float64
 6   RH_TMAX              6 non-null      float64
 7   MIN_TEMP             6 non-null      float64
 8   MAX_TEMP             6 non-null      float64
 9   TSDM_MEAN_Predicted  6 non-null      float64
dtypes: datetime64[ns](1), float64(7), int32(2)
memory usage: 564.0 bytes
```

Historical vs Predicted Monthly Mean TSDM_MEAN

In [71]:
```
# Convert 'OBSERVATION_DATE' to datetime

final_merge['OBSERVATION_DATE'] = pd.to_datetime(final_merge['OBSERVATION_DA
future_data['OBSERVATION_DATE'] = pd.to_datetime(future_data['OBSERVATION_DA

# Set 'OBSERVATION_DATE' as the index for historical data

final_merge.set_index('OBSERVATION_DATE', inplace=True)

# Calculate monthly mean for TSDM_MEAN from historical data

monthly_mean_tsdm = final_merge['TSDM_MEAN'].resample('M').mean()

future_data.set_index('OBSERVATION_DATE', inplace=True)

monthly_predicted_tsdm = future_data['TSDM_MEAN_Predicted'].resample('M').me

# Plotting
```

```
plt.figure(figsize=(15, 8), facecolor='black')

# Plotting Historical Monthly Mean TSDM_MEAN

plt.plot(monthly_mean_tsdm.index, monthly_mean_tsdm,
         color='blue', label='Historical Monthly Mean TSDM_MEAN', linewidth=

# Plotting Predicted Monthly Mean TSDM_MEAN

plt.plot(monthly_predicted_tsdm.index, monthly_predicted_tsdm,
         color='orange', linestyle='--', label='Predicted Monthly Mean TSDM_

plt.title("Historical vs Predicted Monthly Mean TSDM_MEAN", color='white', f
plt.xlabel("Date", color='white', fontsize=14)
plt.ylabel("TSDM_MEAN", color='white', fontsize=14)
plt.legend(frameon=True, loc='upper left', fontsize='small', facecolor='blac
plt.grid(color='grey', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

# Display the monthly means DataFrames

print("Monthly Mean TSDM_MEAN:")
print(monthly_mean_tsdm)
print("\nMonthly Predicted TSDM_MEAN:")
print(monthly_predicted_tsdm)
```



```
Monthly Mean TSDM_MEAN:
OBSERVATION_DATE
2017-01-31    0.263005
2017-02-28    0.255832
2017-03-31    0.193568
2017-04-30    0.215812
2017-05-31    0.195135
                ...
2022-08-31    0.803227
2022-09-30    0.783521
2022-10-31    0.921857
2022-11-30    1.134149
2022-12-31    1.018216
Freq: M, Name: TSDM_MEAN, Length: 72, dtype: float64

Monthly Predicted TSDM_MEAN:
OBSERVATION_DATE
2023-02-28    0.569451
2023-03-31    0.816637
2023-04-30    0.335159
2023-05-31    0.723748
2023-06-30    0.832515
2023-07-31    1.048704
Freq: M, Name: TSDM_MEAN_Predicted, dtype: float64
```