DCS 550

Week 12

Project Milestone 1 + Milestone 2 + Milestone 3

Name: Aniruddha Joshi

Date: Mar 01, 2024

In [ ]:
```
#Defining Libraries required for import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import sklearn as sk
import textblob as tb
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem import PorterStemmer
import operator
import unicodedata
import sys
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix,
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
import geopandas as gpd
from geopy.geocoders import Nominatim
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix,
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import confusion_matrix
from sklearn.inspection import DecisionBoundaryDisplay
```

Table of Contents

Milestone 1

Introduction about the data

Data Reference: https://www.kaggle.com/datasets/goyaladi/climate-insights-dataset/data

Below data represents the climate change behaviour. It consists of more than 10000 records/rows of data.

It consiste of below columns:

1. Date : the date of the record
2. Location : the location of the record
3. Country : the country of the record

4. Temperature : the temperature recorded
5. CO2 Emissions : the CO2 emissions recorded
6. Sea Level Rise : the sea level rise recorded
7. Precipitation : the precipitation recorded
8. Humidity : the humidity recorded
9. Wind Speed : the wind speed recorded

Begin Milestone 1 with a 250-500-word narrative describing your original idea for the analysis/model building business problem.

Questions to explore:

My original idea for analyzing this climate change data delves into three major questions: 1. predicting sea level rise, 2. understanding environmental parameter relationships, and 3. identifying impactful countries on emissions and sea levels. Here's how:

Based on the environmental information available online, Complex interactions exist between temperature, humidity, wind speed, and precipitation, creating positive and negative feedback loops that can amplify or dampen climate change effects. For example, warmer temperatures lead to increased evaporation, causing more precipitation, which can further increase humidity and contribute to warming. Large-scale climate patterns like El Niño and La Niña influence regional variations in temperature, precipitation, and wind speed, making it crucial to consider these when analyzing relationships between environmental parameters.

Correlation Analysis: Explore the strength and direction of relationships between CO2 emissions, temperature, humidity, wind speed, and sea level rise. This will help understand which factors have the most significant influence.

Clustering Analysis: Identify groups of locations with similar environmental parameter patterns, allowing for targeted interventions and resource allocation.

Timeseries Forecasting: An attempt will be made to predict the sea level rise based on CO2 emission, temperature rise.

Regression Analysis: Build models to quantify the relationship between temperature and wind speed to predict optimal wind energy production potential across different locations.

By analyzing this data through these approaches, we can gain valuable insights into the dynamics of climate change, predict future sea level rise, optimize wind energy production, and identify countries with the greatest responsibility for mitigation efforts. This knowledge hopefully helps the policymakers, environmental organizations, and individuals to take informed action toward better future.

In [ ]:
```
#Load the dataset as a Pandas data frame.

climate_change_df = pd.read_csv('climate_change_data.csv')       #https://www.kaggle.com/datasets/goyaladi/climate-ir
```

In [ ]:
```
#take a peak at the data
climate_change_df.head()
```

Out[ ]:

	Date	Location	Country	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed
0	2000-01-01 00:00:00.000000000	New Williamtown	Latvia	10.688986	403.118903	0.717506	13.835237	23.631256	18.492026
1	2000-01-01 20:09:43.258325832	North Rachel	South Africa	13.814430	396.663499	1.205715	40.974084	43.982946	34.249300
2	2000-01-02 16:19:26.516651665	West Williamland	French Guiana	27.323718	451.553155	-0.160783	42.697931	96.652600	34.124261
3	2000-01-03 12:29:09.774977497	South David	Vietnam	12.309581	422.404983	-0.475931	5.193341	47.467938	8.554563
4	2000-01-04 08:38:53.033303330	New Scottburgh	Moldova	13.210885	410.472999	1.135757	78.695280	61.789672	8.001164

In [ ]:
```
#Get the oervall information about the columns and thier types
climate_change_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Date            10000 non-null  object 
 1   Location        10000 non-null  object 
 2   Country         10000 non-null  object 
 3   Temperature     10000 non-null  float64
 4   CO2 Emissions   10000 non-null  float64
 5   Sea Level Rise  10000 non-null  float64
 6   Precipitation   10000 non-null  float64
 7   Humidity        10000 non-null  float64
 8   Wind Speed      10000 non-null  float64
dtypes: float64(6), object(3)
memory usage: 703.3+ KB
```

In [ ]:
```
#Create a new columns with MM-DD-YYYY and MM-YYYY for the data aggreggation. For grouping the measurements

#climate_change_df['Date_MMDDYY'] = pd.to_datetime(climate_change_df['Date'], format='%Y-%m-%d', errors='coerce')


climate_change_df['NewDate']  = pd.to_datetime(climate_change_df['Date']) # climate_change_df['Date'].apply(lambda x.
climate_change_df['Date_MMDDYY'] = climate_change_df['NewDate'].dt.strftime('%m-%d-%Y')
climate_change_df['Date_MMYY'] = climate_change_df['NewDate'].dt.strftime('%m-%Y')
climate_change_df['Date_MMYY'] = pd.to_datetime(climate_change_df['Date_MMYY'])
```

```
C:\Users\joshi\AppData\Local\Temp\ipykernel_4908\2397599314.py:9: UserWarning: Could not infer format, so each elemen
t will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please sp
ecify a format.
  climate_change_df['Date_MMYY'] = pd.to_datetime(climate_change_df['Date_MMYY'])
```

In [ ]:
```
#The sort the values based on time so that the timeserial plts can be plotted on top of each other.
climate_change_df.sort_values(by=['NewDate'], inplace=True)
```

In [ ]:
```
#Get the information about the new columns and make sure that they are date type
climate_change_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            10000 non-null  object
 1   Location        10000 non-null  object
 2   Country         10000 non-null  object
 3   Temperature     10000 non-null  float64
 4   CO2 Emissions   10000 non-null  float64
 5   Sea Level Rise  10000 non-null  float64
 6   Precipitation   10000 non-null  float64
 7   Humidity        10000 non-null  float64
 8   Wind Speed      10000 non-null  float64
 9   NewDate         10000 non-null  datetime64[ns]
 10  Date_MMDDYY     10000 non-null  object
 11  Date_MMYY       10000 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(6), object(4)
memory usage: 937.6+ KB
```

In [ ]:
```
#Find oout how data is distributed? are there any missing values? min/max ranges the meand to identify if therer are
climate_change_df.describe()
```

Out[ ]:

	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed	NewDate	Date_MMYY
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000	10000
mean	14.936034	400.220469	-0.003152	49.881208	49.771302	25.082066	2011-07-02 00:00:00	2011-06-16 18:32:58.560000
min	-3.803589	182.131220	-4.092155	0.010143	0.018998	0.001732	2000-01-01 00:00:00	2000-01-01 00:00:00
25%	11.577991	367.109330	-0.673809	24.497516	24.713250	12.539733	2005-10-01 00:00:00	2005-09-23 12:00:00
50%	14.981136	400.821324	0.002332	49.818967	49.678412	24.910787	2011-07-02 00:00:00	2011-07-01 00:00:00
75%	18.305826	433.307905	0.675723	74.524991	75.206390	37.670260	2017-04-01 00:00:00	2017-03-08 18:00:00
max	33.976956	582.899701	4.116559	99.991900	99.959665	49.997664	2022-12-31 00:00:00	2022-12-01 00:00:00
std	5.030616	49.696933	0.991349	28.862417	28.929320	14.466648	NaN	NaN

In [ ]:

```
#Group by based on the number MM-YYYY.
#Get the montly averages and us that to plot and identify the relationship
climate_grouped_df= climate_change_df.groupby(['Date_MMYY'])
climate_meandata_df=climate_grouped_df[['Temperature','CO2 Emissions','Sea Level Rise','Precipitation','Humidity','W:
```

In [ ]:

```
#Get the index back Date_MMYY as its needed for timeseries
climate_meandata_df = climate_meandata_df.reset_index()
```

In [ ]:

```
#See how grouped data looks like
climate_meandata_df.info()
```

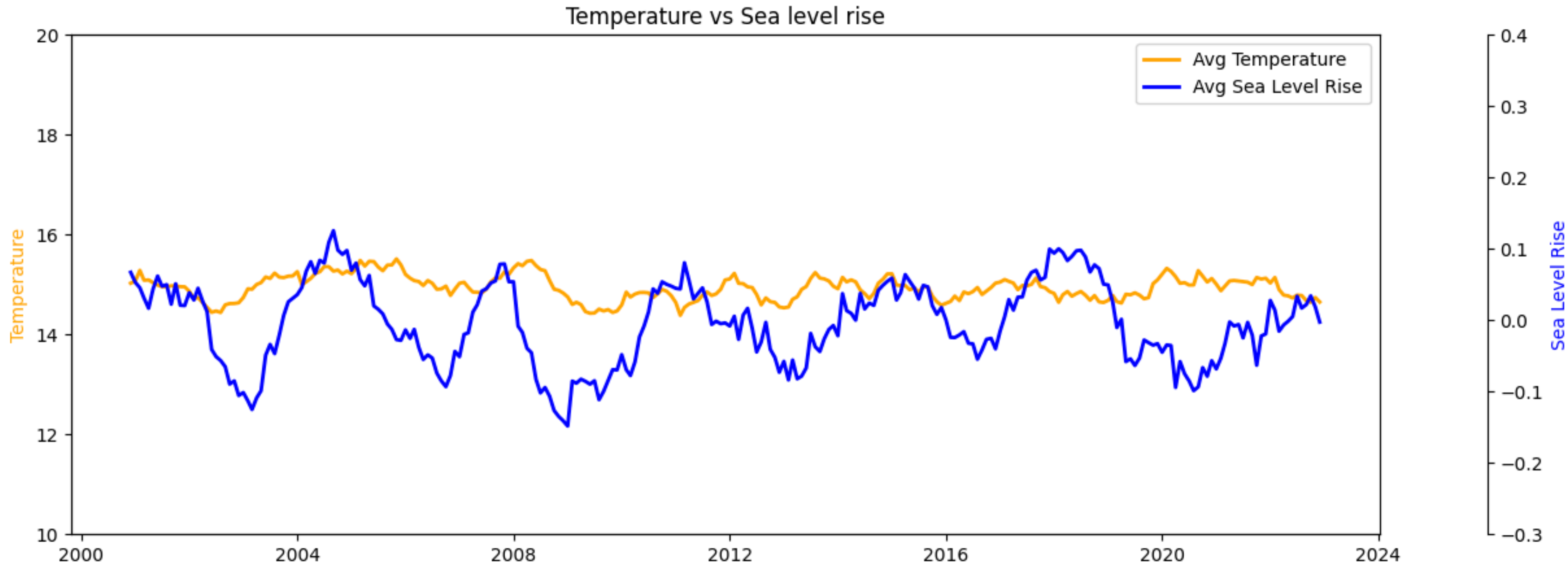
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 276 entries, 0 to 275
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date_MMYY       276 non-null    datetime64[ns]
 1   Temperature     276 non-null    float64
 2   CO2 Emissions   276 non-null    float64
 3   Sea Level Rise  276 non-null    float64
 4   Precipitation   276 non-null    float64
 5   Humidity        276 non-null    float64
 6   Wind Speed      276 non-null    float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 15.2 KB
```

In [ ]:

```
# Plot the graph betrween the timeseries, temperature and Sea level

#first generate the monthly averages or else we see randon spikes

#plt.plot( climate_meandata_df['Date_MMYY'], climate_meandata_df[['Temperature','CO2 Emissions','Sea Level Rise','Pre

# Create a figure and axis
plt.figure(figsize=(10, 6))
fig, ax1 = plt.subplots(figsize=(13, 5))

#Get the rolling mean of the temperature
t_average = climate_meandata_df['Temperature'].rolling(window=12).mean()
ax1.plot(climate_meandata_df['Date_MMYY'], t_average, color='orange', label='Avg Temperature', linewidth=2)
ax1.set_ylim(10,20)


ax1.set_ylabel('Temperature', color='orange')

# Create a second y-axis for the third time series
ax2 = ax1.twinx()
ax2.spines['right'].set_position(('outward', 60))  # Adjust position of the axis

seaLevel_average = climate_meandata_df['Sea Level Rise'].rolling(window=12).mean()
ax2.plot(climate_meandata_df['Date_MMYY'], seaLevel_average, color='blue', label='Avg Sea Level Rise', linewidth=2)


#ax2.plot(climate_meandata_df['Date_MMYY'], climate_meandata_df['Sea Level Rise'], color='orange', label='Sea Level R
```

```
ax2.set_ylabel('Sea Level Rise', color='blue')
ax2.set_ylim(-0.3, 0.4)

# Set titles and legends
ax1.set_title('Temperature vs Sea level rise')
lines = [ax1.get_lines()[0], ax2.get_lines()[0]]
plt.legend(lines, [line.get_label() for line in lines], loc='upper right')
plt.xticks(fontsize=8)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

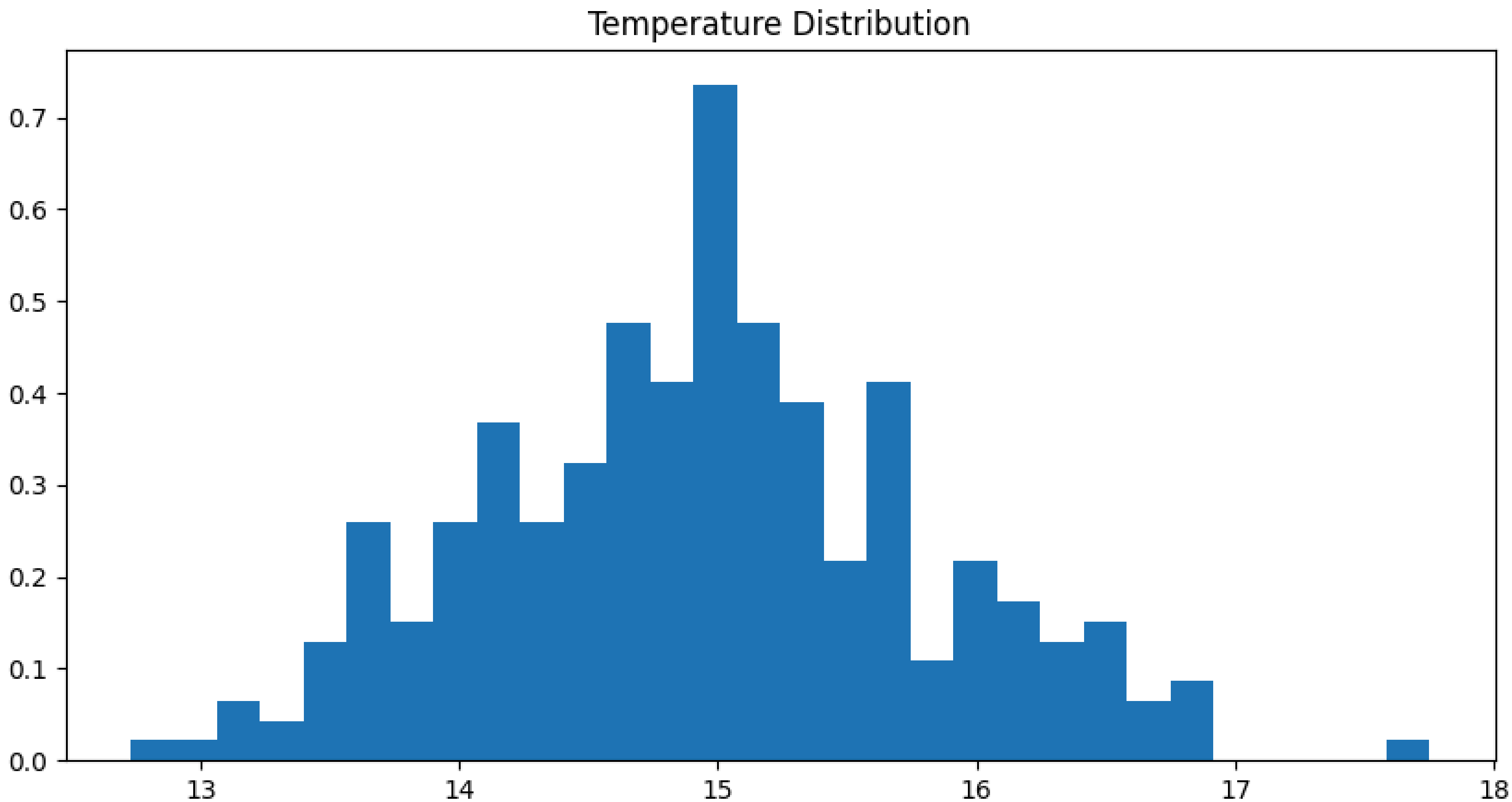


Explanation from the trend line

The purpose of the trend line to identify if the two variables Temperatures vs Sea level are correlated. Based on the correlation calculated at the end, there is a 0.16 correlation between Temperatures and Sea level. Although that is not significant, however it's based on the instantaneous reading and in true case increasing the temperature might take months for the sea level to rise.

The graph was plotted to identify if there are such patters. In the initial section of the graph it does show increasing temperatures increases the sea level. However, they are not perfectly correlated.

In [ ]:
```
#Plotting histograms
plt.figure(figsize=(10, 5))
plt.hist(climate_meandata_df['Temperature'], bins=30,density=True,label='Avg Temperature')
plt.title('Temperature Distribution')

plt.show()
#the temperaure is approximately linearlly distributed
```

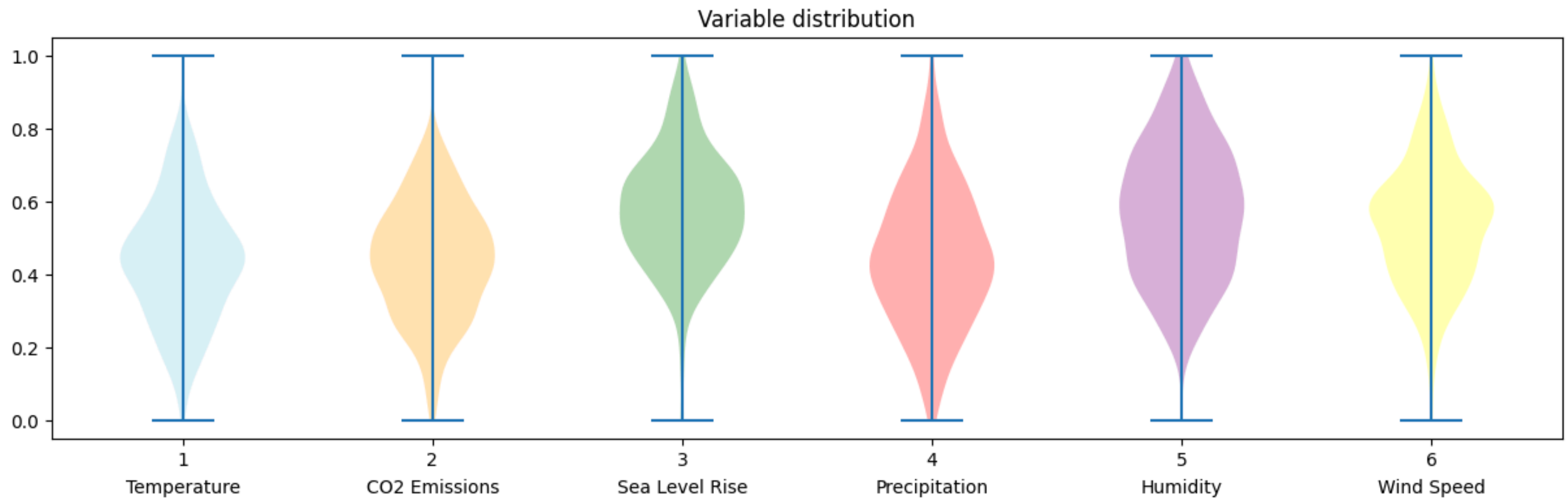


Explanation

The histogram of temperature shows that the readings are normally distributed, and the graph is not skewed.

In [ ]:
```
# Normalize specific columns using Min-Max normalization
#copy the database into another dataframe -- keep the original data intact
climate_meandata_normalized_df=climate_meandata_df.copy()
columns_to_normalize = ['Temperature','CO2 Emissions','Sea Level Rise','Precipitation','Humidity','Wind Speed']
#normalize the data
climate_meandata_normalized_df[columns_to_normalize] = (climate_meandata_df[columns_to_normalize] - climate_meandata_
```

In [ ]:
```
#Plot the violin plot

#plt.style.use('_mpl-gallery')
colors = ['skyblue', 'orange', 'green', 'red', 'purple', 'yellow']
variables = ['Temperature', 'CO2 Emissions', 'Sea Level Rise', 'Precipitation', 'Humidity', 'Wind Speed']

plt.figure(figsize=(15, 4))
violins = plt.violinplot(climate_meandata_normalized_df[['Temperature','CO2 Emissions','Sea Level Rise','Precipitatio
plt.title('Variable distribution')
# Adding Legends for each violin plot
#plt.legend(violins['bodies'], ['Temperature', 'CO2 Emissions', 'Sea Level Rise', 'Precipitation', 'Humidity', 'Wind

# Customizing colors for each violin plot
for i, pc in enumerate(violins['bodies']):
    pc.set_facecolor(colors[i])

# Adding variable names below each violin plot
for i, variable in enumerate(variables):
    plt.text(i + 1, -0.2, variable, ha='center')

plt.show()
```



Explanation from violin plot

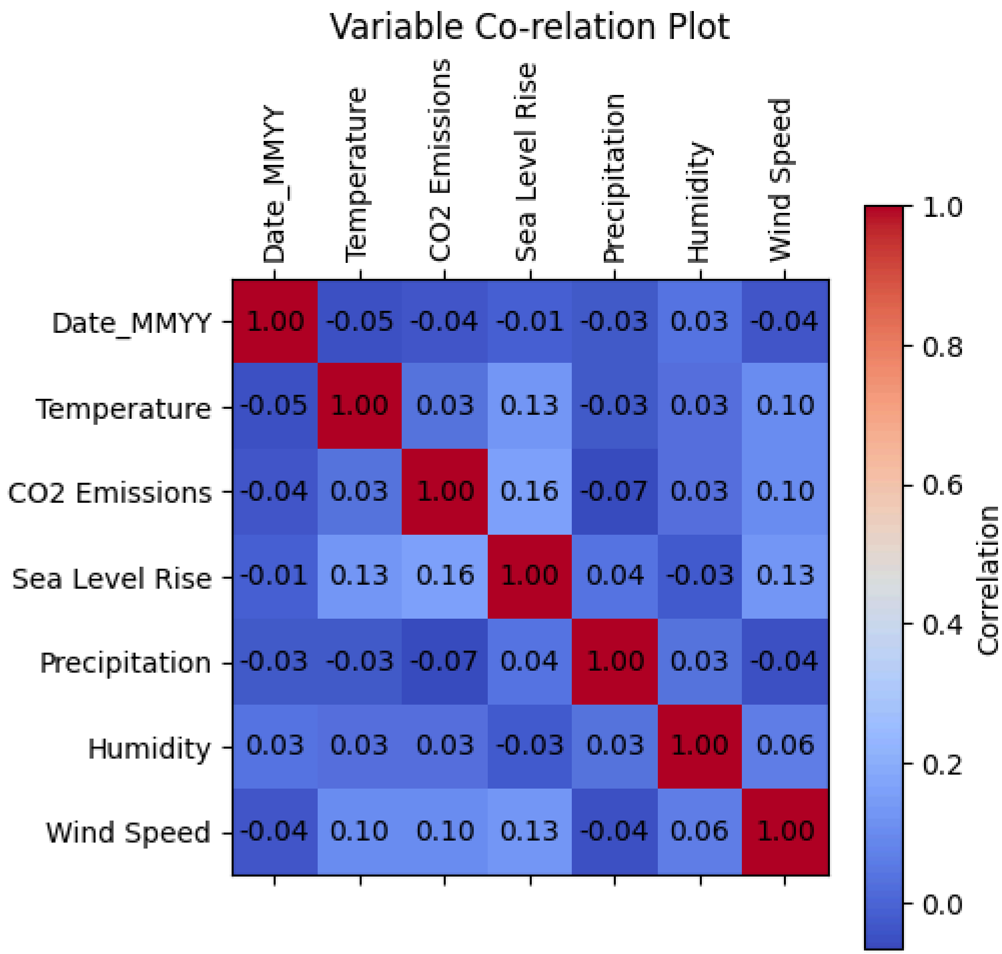
The plot shows that most of the noromalized data is normally distributed. Some parameters like Sea Level Rise and Humidity are skewed. There might be some outliers in Temperature.

In [ ]:
```
#See how ingerenal the normalized data looks like
climate_meandata_normalized_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 276 entries, 0 to 275
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date_MMYY       276 non-null    datetime64[ns]
 1   Temperature     276 non-null    float64
 2   CO2 Emissions   276 non-null    float64
 3   Sea Level Rise  276 non-null    float64
 4   Precipitation   276 non-null    float64
 5   Humidity        276 non-null    float64
 6   Wind Speed      276 non-null    float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 15.2 KB
```

In [ ]:
```
#Plot the heatmap with co-rellation values
#heatmap(climate_meandata_normalized_df[['Temperature','CO2 Emissions','Sea Level Rise','Precipitation','Humidity','W
#plt.show()
plt.figure(figsize=(20, 20))
plt.matshow(climate_meandata_normalized_df.corr(),cmap='coolwarm')
plt.title('Variable Co-relation Plot')
# Add colorbar legend
plt.colorbar(label='Correlation')
# Specify variable names as ticks on x and y axes
variables = list(climate_meandata_normalized_df.columns)
plt.xticks(range(len(variables)), variables, rotation=90)
plt.yticks(range(len(variables)), variables)

# Remove gridlines
plt.grid(False)

corr_matrix = climate_meandata_normalized_df.corr()
for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix)):
        plt.text(j, i, f'{corr_matrix.iloc[i, j]:.2f}', ha='center', va='center', color='black')



plt.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```



Explanation from correlation map

The map shows the correlation between the parameters. The graph shows how change in one parameter affects the other.

Conclusion:

1. So far the CO2 Emissions and Sea Level does show a very small positive correlation (0.16 factor).
2. Surprisingly there is no correlation between temperature and Humidity

3. Overall it seems there is no correlation between Precipitation and Humidity with other parameters - this may tell us that additional analysis is required
4. A time shift e.g. between the temperature and other parameters e.g. 30 days/90 days/120 days may provide some correlation.

Overall, the matplotlib provide various options to plot the data that can provide great insight.

Milestone 2

Some of the Milestone 2 activites are already performed in the Milestone 1; however additional check shall be performed to see if data is missing or incorrect

In [ ]:
```
#check if theere are any NaN or missing values
# Identifying the bad data
print(climate_change_df.isna().sum())
```

```
Date              0
Location          0
Country           0
Temperature       0
CO2 Emissions     0
Sea Level Rise    0
Precipitation     0
Humidity          0
Wind Speed        0
NewDate           0
Date_MMDDYY       0
Date_MMYY         0
dtype: int64
```

There are no missing values

In [ ]:
```
#Get the basic description
print(climate_change_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            10000 non-null  object
 1   Location        10000 non-null  object
 2   Country         10000 non-null  object
 3   Temperature     10000 non-null  float64
 4   CO2 Emissions   10000 non-null  float64
 5   Sea Level Rise  10000 non-null  float64
 6   Precipitation   10000 non-null  float64
 7   Humidity        10000 non-null  float64
 8   Wind Speed      10000 non-null  float64
 9   NewDate         10000 non-null  datetime64[ns]
 10  Date_MMDDYY     10000 non-null  object
 11  Date_MMYY       10000 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(6), object(4)
memory usage: 937.6+ KB
None
```

In [ ]:
```
#check if there are  any outliers
(climate_change_df.describe())
```

Out[ ]:

	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed	NewDate	Date_MMYY
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000	10000
mean	14.936034	400.220469	-0.003152	49.881208	49.771302	25.082066	2011-07-02 00:00:00	2011-06-16 18:32:58.560000
min	-3.803589	182.131220	-4.092155	0.010143	0.018998	0.001732	2000-01-01 00:00:00	2000-01-01 00:00:00
25%	11.577991	367.109330	-0.673809	24.497516	24.713250	12.539733	2005-10-01 00:00:00	2005-09-23 12:00:00
50%	14.981136	400.821324	0.002332	49.818967	49.678412	24.910787	2011-07-02 00:00:00	2011-07-01 00:00:00
75%	18.305826	433.307905	0.675723	74.524991	75.206390	37.670260	2017-04-01 00:00:00	2017-03-08 18:00:00
max	33.976956	582.899701	4.116559	99.991900	99.959665	49.997664	2022-12-31 00:00:00	2022-12-01 00:00:00
std	5.030616	49.696933	0.991349	28.862417	28.929320	14.466648	NaN	NaN

In [ ]:
```
# Dropping any columns not required for the regression and clustering

#Lets get the copy of the database first
climate_change_analysis_df = climate_change_df.copy()
```

In [ ]:
```
#For regresion analysis, currently the location is not considered for analysis purposes as we want to analyse the tem
climate_change_analysis_df = climate_change_analysis_df.drop('Location', axis=1)
```

In [ ]:
```
#The countr is part of the location hence removing it
climate_change_analysis_df = climate_change_analysis_df.drop('Country', axis=1)
```

In [ ]:
```
#There are other date formats created for the trending purposes, removeing all non applicable date formats
climate_change_analysis_df = climate_change_analysis_df.drop('NewDate', axis=1)
```

In [ ]:
```
#There are other date formats created for the trending purposes, removeing all non applicable date formats
climate_change_analysis_df = climate_change_analysis_df.drop('Date_MMDDYY', axis=1)
```

In [ ]:
```
#There are other date formats created for the trending purposes, removeing all non applicable date formats
climate_change_analysis_df = climate_change_analysis_df.drop('Date_MMYY', axis=1)
```

In [ ]:
```
#There are other date formats created for the trending purposes, removeing all non applicable date formats
climate_change_analysis_df = climate_change_analysis_df.drop('Date', axis=1)
```

In [ ]:
```
#Final dataset for regression analysis
climate_change_analysis_df
```

Out[ ]:

	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed
0	10.688986	403.118903	0.717506	13.835237	23.631256	18.492026
1	13.814430	396.663499	1.205715	40.974084	43.982946	34.249300
2	27.323718	451.553155	-0.160783	42.697931	96.652600	34.124261
3	12.309581	422.404983	-0.475931	5.193341	47.467938	8.554563
4	13.210885	410.472999	1.135757	78.695280	61.789672	8.001164
...	...	...	...	...	...	...
9995	15.020523	391.379537	-1.452243	93.417109	25.293814	6.531866
9996	16.772451	346.921190	0.543616	49.882947	96.787402	42.249014
9997	22.370025	466.042136	1.026704	30.659841	15.211825	18.293708
9998	19.430853	337.899776	-0.895329	18.932275	82.774520	42.424255
9999	12.661928	381.172746	2.260788	78.339658	99.243923	41.856539

10000 rows × 6 columns

Normalizing the data

In [ ]:
```
#Lets do it on th ecopy of the database, we might neeed both scenarios
climate_change_analysis_Norm_df = climate_change_analysis_df.copy()
```

In [ ]:
```
climate_change_analysis_Norm_df = (climate_change_analysis_Norm_df - climate_change_analysis_Norm_df.min()) / (climat
```

In [ ]:
```
climate_change_analysis_Norm_df.describe()
```

Out[ ]:

	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	0.496013	0.544178	0.498130	0.498802	0.497818	0.501647
std	0.133154	0.124004	0.120768	0.288677	0.289465	0.289357
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.407130	0.461559	0.416429	0.244918	0.247089	0.250780
50%	0.497206	0.545677	0.498798	0.498179	0.496889	0.498222
75%	0.585206	0.626738	0.580831	0.745284	0.752320	0.753432
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Milestone 3

In Milestone 3, begin the process of selecting, building, and evaluating a model. Required to train and evaluate at least one model in this milestone. Write step-by-step for performing each of these steps.

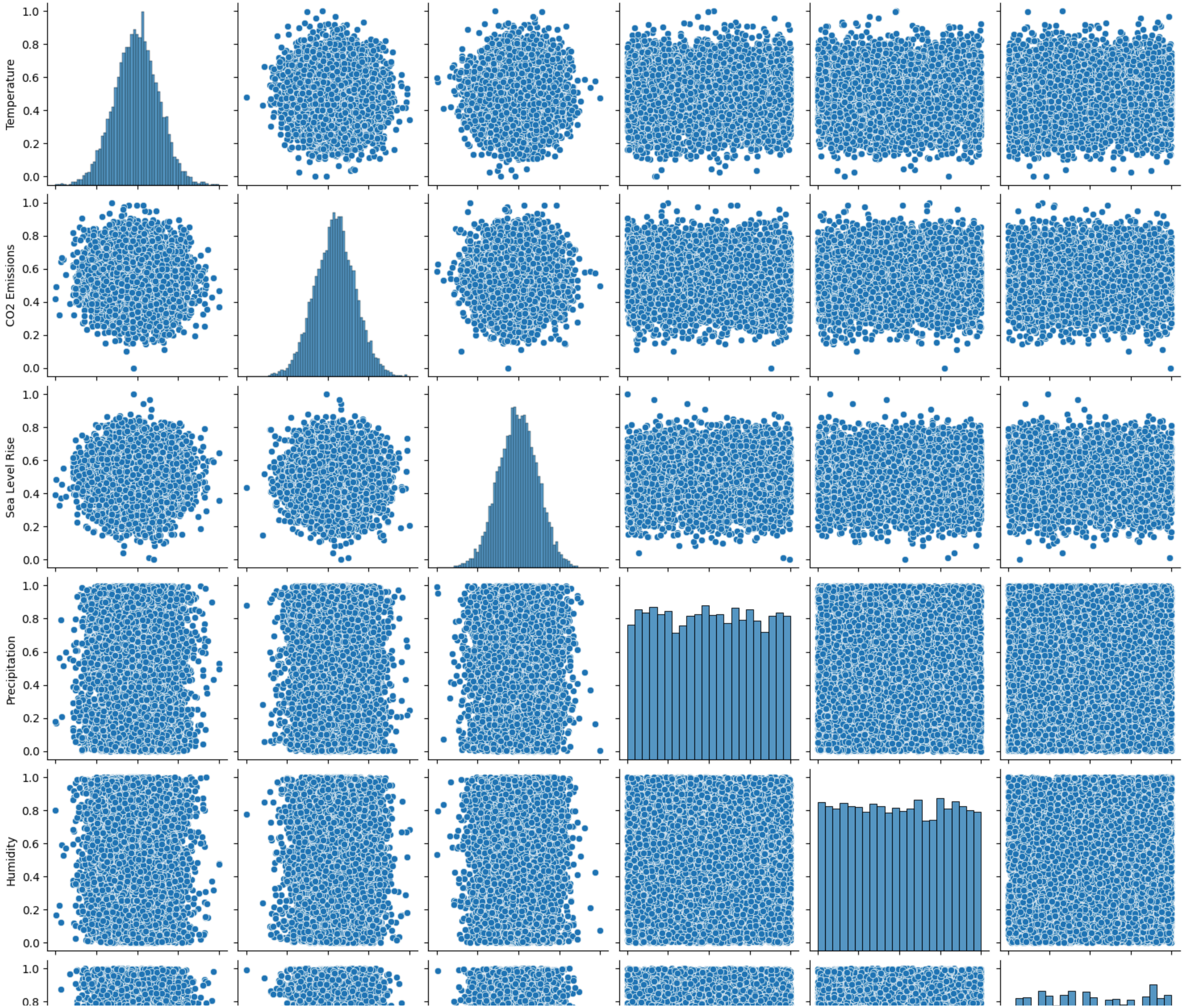
Understand the data structure

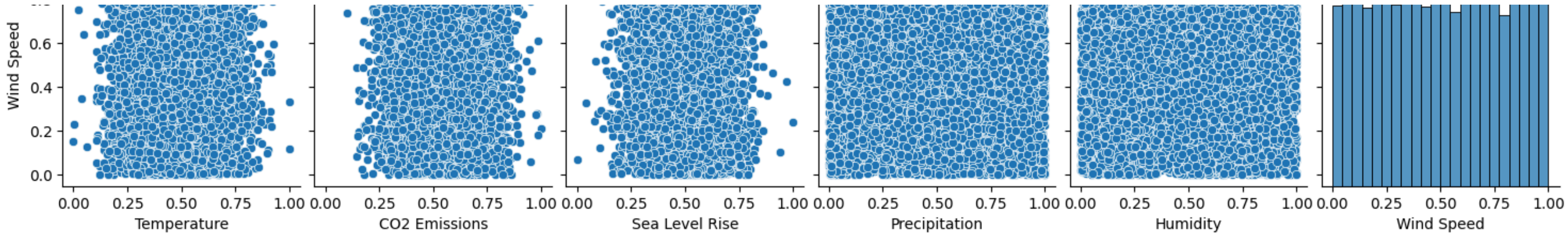
In [ ]:
```
climate_change_analysis_Norm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Temperature     10000 non-null  float64
 1   CO2 Emissions   10000 non-null  float64
 2   Sea Level Rise  10000 non-null  float64
 3   Precipitation   10000 non-null  float64
 4   Humidity        10000 non-null  float64
 5   Wind Speed      10000 non-null  float64
dtypes: float64(6)
memory usage: 468.9 KB
```

In [ ]:

```
# Visualize the data
sns.pairplot(climate_change_analysis_Norm_df[['Temperature', 'CO2 Emissions', 'Sea Level Rise', 'Precipitation', 'Hur
plt.show()
```


Temperature
CO2 Emissions
Sea Level Rise
Precipitation
Humidity
0.0
0.2
0.4
0.6
0.8
1.0



The data is now normalized lets try applying the model

We will apply first the regression model to predict the sea level change based on various parameters such as Temperature, CO2 Emissions, Precipitation, Humidity and Wind Speed

Split the data into a training and test set,

In [ ]:
```
# Split the data into training and testing sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    climate_change_analysis_Norm_df.drop('Sea Level Rise', axis=1),  # Features (drop the target column)
    climate_change_analysis_Norm_df['Sea Level Rise'],  # Target variable
    test_size=0.2,  # 20% for testing
    random_state=10  # Set random state for reproducibility
)
```

In [ ]:
```
# Print the shapes of the resulting sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 5) (2000, 5) (8000,) (2000,)
```

In [ ]:
```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
```

```
print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Squared Error (MSE): 0.01464969652827849
R-squared (R2): -0.0017805158472015137
```

The values of R2 and rmse does not provide any positive outcome. The R2 value indicates that the model does not explain much of the variance in the target variable.

Lets try dropping the columns that are not relevant

In [ ]:
```
#Copy the database
climate_change_analysis_Norm_modified_df = climate_change_analysis_Norm_df.copy()
```

In [ ]:
```
climate_change_analysis_Norm_modified_df = climate_change_analysis_Norm_modified_df.drop(['Precipitation', 'Humidity
```

In [ ]:
```
# Split the data into training and testing sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    climate_change_analysis_Norm_modified_df.drop('Sea Level Rise', axis=1),  # Features (drop the target column)
    climate_change_analysis_Norm_modified_df['Sea Level Rise'],  # Target variable
    test_size=0.2,  # 20% for testing
    random_state=10  # Set random state for reproducibility
)
```

In [ ]:
```
# Print the shapes of the resulting sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 3) (2000, 3) (8000,) (2000,)
```

In [ ]:
```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Squared Error (MSE): 0.014625093942665621
R-squared (R2): -9.813349485310319e-05
```

The results are still not good. :Lets try keeping only CO2 emission

In [ ]:
```
climate_change_analysis_Norm_modified_df = climate_change_analysis_Norm_modified_df.drop(['Temperature', 'Wind Speed
```

In [ ]:
```
# Split the data into training and testing sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    climate_change_analysis_Norm_modified_df.drop('Sea Level Rise', axis=1),  # Features (drop the target column)
    climate_change_analysis_Norm_modified_df['Sea Level Rise'],  # Target variable
    test_size=0.2,  # 20% for testing
    random_state=10  # Set random state for reproducibility
)
```

In [ ]:
```
# Print the shapes of the resulting sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 1) (2000, 1) (8000,) (2000,)
```

In [ ]:
```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Squared Error (MSE): 0.014633658110506391
R-squared (R2): -0.0006837713243310617
```

The R2 value indicates that the model does not explain much of the variance in the target variable. Lets try different models.

Now, lets different models to see the reuslts

In [ ]:
```
#Copy the database
climate_change_analysis_Norm_modified_df = climate_change_analysis_Norm_df.copy()
```

In [ ]:
```
# Split the data into training and testing sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    climate_change_analysis_Norm_modified_df.drop('Sea Level Rise', axis=1),  # Features (drop the target column)
    climate_change_analysis_Norm_modified_df['Sea Level Rise'],  # Target variable
    test_size=0.2,  # 20% for testing
    random_state=10  # Set random state for reproducibility
)
```

In [ ]:
```
# Print the shapes of the resulting sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 5) (2000, 5) (8000,) (2000,)
```

Lets try Random Forest Regression

In [ ]:
```
# Random Forest Regression
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_pred)
rf_r2 = r2_score(y_test, rf_pred)
print('Random Forest Regression - MSE:', rf_mse)
print('Random Forest Regression - R2 Score:', rf_r2)
```

```
Random Forest Regression - MSE: 0.015375657750402266
Random Forest Regression - R2 Score: -0.05142344231877671
```

R2 indicates that the model does not explain much of the variance in the target variable.

Lets try the Decision Tree Regressor

In [ ]:
```
# Decision Tree Regression
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
```

```
dt_mse = mean_squared_error(y_test, dt_pred)
dt_r2 = r2_score(y_test, dt_pred)
print('Decision Tree Regression - MSE:', dt_mse)
print('Decision Tree Regression - R2 Score:', dt_r2)
```

```
Decision Tree Regression - MSE: 0.0291847513265374
Decision Tree Regression - R2 Score: -0.9957215620361106
```

R2 indicates that the model does not explain much of the variance in the target variable.

Lets try SVR

In [ ]:
```
# Support Vector Regression
svr_model = SVR()
svr_model.fit(X_train, y_train)
svr_pred = svr_model.predict(X_test)
svr_mse = mean_squared_error(y_test, svr_pred)
svr_r2 = r2_score(y_test, svr_pred)
print('Support Vector Regression - MSE:', svr_mse)
print('Support Vector Regression - R2 Score:', svr_r2)
```

```
Support Vector Regression - MSE: 0.015271349112850106
Support Vector Regression - R2 Score: -0.04429057369364098
```

R2 indicates that the model does not explain much of the variance in the target variable.

None of the models provide great results. Now lets try converting the sea level increase column to Boolean / classification

e.g. if sea level > 0 then its true (sea level increased) if sea level < 0 then the seal level is decreased

In [ ]:
```
#Add a boolean column for sea level detection
climate_change_analysis_Norm_modified_df['DiscreteSeaLevel'] = np.where(climate_change_analysis_Norm_modified_df['Sea
```

In [ ]:
```
climate_change_analysis_Norm_modified_df.head()
```

Out[ ]:

	Temperature	CO2 Emissions	Sea Level Rise	Precipitation	Humidity	Wind Speed	DiscreteSeaLevel
0	0.383599	0.551410	0.585921	0.138276	0.236263	0.369836	True
1	0.466325	0.535302	0.645396	0.409714	0.439900	0.685007	True
2	0.823898	0.672263	0.478927	0.426956	0.966910	0.682506	True
3	0.426494	0.599533	0.440535	0.051841	0.474771	0.171071	True
4	0.450350	0.569760	0.636873	0.786995	0.618073	0.160002	True

In [ ]:

```
#dropping the conitineous variable and keeping only categorical /boolean variable for sea level increase/decrease
climate_change_analysis_Norm_modified_df = climate_change_analysis_Norm_modified_df.drop(['Sea Level Rise'], axis=1)
```

In [ ]:

```
climate_change_analysis_Norm_modified_df.head()
```

Out[ ]:

	Temperature	CO2 Emissions	Precipitation	Humidity	Wind Speed	DiscreteSeaLevel
0	0.383599	0.551410	0.138276	0.236263	0.369836	True
1	0.466325	0.535302	0.409714	0.439900	0.685007	True
2	0.823898	0.672263	0.426956	0.966910	0.682506	True
3	0.426494	0.599533	0.051841	0.474771	0.171071	True
4	0.450350	0.569760	0.786995	0.618073	0.160002	True

In [ ]:

```
# Split the data into training and testing sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(
    climate_change_analysis_Norm_modified_df.drop('DiscreteSeaLevel', axis=1),  # Features (drop the target column)
    climate_change_analysis_Norm_modified_df['DiscreteSeaLevel'],  # Target variable
    test_size=0.2,  # 20% for testing
    random_state=10  # Set random state for reproducibility
)
```

Now let's apply the classification models

In [ ]:
```
# Create and train the Decision Tree Classifier model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

        True       1.00      1.00      1.00      2000

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

The decision tree classified is now predicting the sea level increase/decrease outcome very well. Let's try applying other models and see the results

In [ ]:
```
# Create and train the RandomForestClassifier model
model =RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

        True       1.00      1.00      1.00      2000

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

In [ ]:

```
# Create a KNeighborsClassifier with k=3 (you can adjust k as needed)
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the evaluation metrics
print(f'Accuracy: {accuracy}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(class_report)
```

```
Accuracy: 1.0
Confusion Matrix:
[[2000]]
Classification Report:
              precision    recall  f1-score   support

        True       1.00      1.00      1.00      2000

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

Conclusion:

In general the Regression models do not provide great accuracy. However, converting the sea level increase/decrease categorical (boolean) variable do provide great accuracy. In genreal we tested most of the possible models and found that the DecisionTreeClassifier, RandomForestClassifier and KNeighborsClassifier provides great results.

Based on the models identified, it's difficult to provide exact sealevel increase in %. but it seems it can predict if the sealevel can increase/decrease (its dirction).

I would propose to use the DecisionTreeClassifier for the model prediction as it provides accurate prediction.