Project Title	Climate Change Modeling
Tools	Jupyter Notebook and VS code
Technologies	Machine learning
Domain	Data Science
Project Difficulties level	Advanced

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click here to download data set]

About Dataset

Overview

This dataset encompasses over 500 user comments collected from high-performing posts on NASA's Facebook page dedicated to climate change (https://web.facebook.com/NASAClimateChange/). The comments, gathered from various posts between 2020 and 2023, offer a diverse range of public opinions and sentiments about climate change and NASA's related activities.

Data Science Applications

Despite not being a large dataset, it offers valuable opportunities for analysis and Natural Language Processing (NLP). Potential applications include:

- **Sentiment Analysis:** Gauge public opinion on climate change and NASA's communication strategies.
- **Trend Analysis:** Identify shifts in public sentiment over the specified period.
- **Engagement Analysis:** Understand the correlation between the content of a post and user engagement.
- **Topic Modeling:** Discover prevalent themes in public discourse about climate change.

Column Descriptors

1. **Date:** The date and time when the comment was posted.
2. **LikesCount:** The number of likes each comment received.
3. **ProfileName:** The anonymized name of the user who posted the comment.
4. **CommentsCount:** The number of responses each comment received.
5. **Text:** The actual text content of the comment.

Ethical Considerations and Data Privacy

All profile names in this dataset have been hashed using SHA-256 to ensure privacy while maintaining data usability. This approach aligns with ethical data mining practices, ensuring that individual privacy is respected without compromising the dataset's analytical value.

Acknowledgements

We extend our gratitude to NASA and their Facebook platform for facilitating open discussions on climate change. Their commitment to fostering public engagement and awareness on this critical global issue is deeply appreciated.

Note to Data Scientists

As data scientists analyze this dataset, it is crucial to approach the data impartially. Climate change is a subject with diverse viewpoints, and it is important to handle the data and any derived insights in a manner that respects these different perspectives.

Climate Change Modeling Machine Learning Project

Project Overview

The Climate Change Modeling project aims to develop a machine learning model to predict and understand various aspects of climate change. This can include predicting temperature changes, sea level rise, extreme weather events, and other related phenomena. The project involves analyzing historical climate data, identifying trends, and making future projections to help in planning and mitigation efforts.

Project Steps

1. **Understanding the Problem**
    - The goal is to predict and model various climate change indicators, such as temperature anomalies, precipitation patterns, and sea level changes, using historical climate data and machine learning techniques.
2. **Dataset Preparation**
    - **Data Sources**: Collect data from sources like NOAA (National Oceanic and Atmospheric Administration), NASA, IPCC (Intergovernmental Panel on Climate Change), and other climate research organizations.
    - **Features**: Include variables like temperature, precipitation, CO_2 levels, solar radiation, sea level, and other relevant environmental factors.

    - **Labels**: Climate change indicators such as temperature anomalies, sea level rise, frequency of extreme weather events.
3. **Data Exploration and Visualization**
    - Load and explore the dataset using descriptive statistics and visualization techniques.
    - Use libraries like Pandas for data manipulation and Matplotlib/Seaborn for visualization.
    - Identify trends, patterns, and correlations in the data.
4. **Data Preprocessing**
    - Handle missing values through imputation or removal.
    - Standardize or normalize continuous features.
    - Encode categorical variables using techniques like one-hot encoding.
    - Split the dataset into training, validation, and testing sets.
5. **Feature Engineering**
    - Create new features that may be useful for prediction, such as rolling averages or lagged variables.
    - Perform feature selection to identify the most relevant features for the model.
6. **Model Selection and Training**
    - Choose appropriate machine learning algorithms based on the problem. Common choices include:
        - Linear Regression
        - Decision Trees
        - Random Forest
        - Gradient Boosting Machines (e.g., XGBoost)
        - Neural Networks
        - Long Short-Term Memory (LSTM) networks for time series data
    - Train multiple models to find the best-performing one.
7. **Model Evaluation**

    - Evaluate the models using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
    - Use cross-validation to ensure the model generalizes well to unseen data.
    - Visualize model performance using plots like residual plots and predicted vs. actual plots.
8. **Future Projections**
    - Use the trained model to make future projections of climate change indicators.
    - Validate the projections using available data and compare them with scientific forecasts and models.
9. **Scenario Analysis**
    - Conduct scenario analysis to understand the impact of different factors (e.g., CO_2 emission scenarios) on climate change.
    - Use the model to simulate different scenarios and assess their potential impact.
10. **Deployment (Optional)**
    - Deploy the model using a web framework like Flask or Django.
    - Create a user-friendly interface where users can input data and receive climate change predictions and scenarios.
11. **Documentation and Reporting**
    - Document the entire process, including data exploration, preprocessing, feature engineering, model training, evaluation, and projections.
    - Create a final report or presentation summarizing the project, results, and insights.

Example: You can get the basic idea how you can create a project from here

Sample Code

Here’s a basic example using Python and scikit-learn to model climate change indicators

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
# Example: Using a mock dataset with climate data
data = pd.read_csv('climate_data.csv')

# Explore the dataset
print(data.head())
print(data.describe())

# Preprocess the data
# Separate features and labels
X = data.drop('temperature_anomaly', axis=1)
y = data['temperature_anomaly']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'R2: {r2}')

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
```

```
plt.xlabel('Actual Temperature Anomaly')
plt.ylabel('Predicted Temperature Anomaly')
plt.title('Actual vs Predicted Temperature Anomaly')
plt.show()

# Future projections (mock example)
# Assuming we have future data for the same features
future_data = pd.read_csv('future_climate_data.csv')
future_data_scaled = scaler.transform(future_data)
future_predictions = model.predict(future_data_scaled)

print(future_predictions)
```

This code demonstrates loading a climate dataset, preprocessing the data, training a Random Forest regressor, evaluating the model, and making future projections.

Additional Tips

- Incorporate domain expertise to ensure the model's predictions are realistic and scientifically valid.
- Use advanced time series forecasting techniques like LSTM networks for more accurate long-term predictions.
- Continuously update the model with new data to improve its accuracy and relevance over time.
- Collaborate with climate scientists to validate and interpret the model's predictions.

Example: You can get the basic idea how you can create a project from here

Sample code with output

```
%%capture
# Install relevant libraries
!pip install geopandas folium
```

```
In [2]:
# Import libraries
import pandas as pd
import numpy as np
import random
import os
from tqdm.notebook import tqdm

import geopandas as gpd
from shapely.geometry import Point
import folium

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error
pd.options.display.float_format = '{:.5f}'.format
pd.options.display.max_rows = None

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# You can ignore the Shapely GEOS warning :-)
```

```
/opt/conda/lib/python3.7/site-packages/geopandas/_compat.py:115
: UserWarning: The Shapely GEOS version (3.9.1-CAPI-1.14.2) is
incompatible with the GEOS version PyGEOS was compiled with
(3.10.4-CAPI-1.16.2). Conversions between both will be slow.
  shapely_geos_version, geos_capi_version_string
```

In [3]:

```
# Set seed for reproducability
SEED = 2023
random.seed(SEED)
np.random.seed(SEED)
```

2. Loading and previewing data

In [4]:

```
DATA_PATH = '/kaggle/input/playground-series-s3e20'
# Load files
train = pd.read_csv(os.path.join(DATA_PATH, 'train.csv'))
test = pd.read_csv(os.path.join(DATA_PATH, 'test.csv'))
samplesubmission = pd.read_csv(os.path.join(DATA_PATH,
'sample_submission.csv'))

# Preview train dataset
train.head()
```

Out[4]:

	ID_LAT_LON_YE	latitude	longitude	year	week_no	SulphurDioxide_SO2_column_number_dens	SulphurDioxide_SO2_column_number_density_	SulphurDioxide_SO2_slant_column_number_den	SulphurDioxide_cloud	SulphurDioxide_sensor_azimuth_	...	Cloud_cloud_t	Cloud_cloud_base_pre	Cloud_cloud_	Cloud_cloud_o	Cloud_surfa	Cloud_sensor_azimut	Cloud_sensor_ze	Cloud_solar_azim

	AR_WEEK					ity	amf	sity	_fraction	angle		op_height	ssure	base_height	ptical_depth	ce_albedo	h_angle	nith_angle	uth_angle
0	ID_-0.510_29.290_2019_00	-0.510000	29.290000	2019	0	-0.00011	0.60302	-0.00007	0.255567	-98.59389	...	3664.43622	61085.80957	2615.12048	15.56853	0.27229	-12.62899	35.63242	-138.78642
1	ID_-0.	-0.	29.	201	1	0.00002	0.72821	0.00001	0.1309	16.59286	...	365	66969.	317	8.69	0.2	30.359	39.5	-145.

	510_29.290__2019_01	51000	29000	9					9			1.119031	47873	4.57242	060	5683	38	5763	18393	
2	ID_-0.510_29.290__2019_02	-0.51000	29.29000	2019	2	0.00051	0.74820	0.00038	0.11002	72.79584	...	4216.98649	606068.89445	3516.28267	21.10341	0.25110	15.37788	30.40182	-142.51954	
3	ID_-0.	-0.	29.	201	3	NaN	NaN	NaN	NaN	NaN	...	522	51064.	418	15.3	0.2	-11.29	24.3	-132.	

	510_29.290_2_2019_03	51000	29000	9								8.50774	54734	0.97332	8690	6204	340	8036	66583	
4	ID_-0.510_29.290_2_2019_04	-0.51000	29.29000	2019	4	-0.00008	0.67630	-0.00005	0.12116	4.12127	...	3980.59812	63751.12578	3355.71011	8.11469	0.23585	38.53226	37.39298	-141.50981	

5 rows × 76 columns

In [5]:

```
# Preview test dataset
```

```
test.head()
```

Out[5]:

	ID_LAT_LON_YEAR_WEEK	latitude	longitude	year	week_no	SulphurDioxide_SO2_column_number_density	SulphurDioxide_SO2_column_number_density_amf	SulphurDioxide_SO2_slant_column_number_density	SulphurDioxide_cloud_fraction	SulphurDioxide_sensor_azimuth_angle	...	Cloud_cloud_top_pressure	Cloud_cloud_top_height	Cloud_cloud_base_pressure	Cloud_cloud_base_height	Cloud_cloud_optical_depth	Cloud_surface_albedo	Cloud_sensor_azimuth_angle	Cloud_sensor_zenith_angle	Cloud_solar_azimuth_angle

0	ID_-0.510_29.290_2022_00	-0.5100	29.2900	2022	0	NaN	NaN	NaN	NaN	NaN	...	3602.02734	8472.31348	41047.93750	7472.31348	7.93562	0.24077	-100.11379	33.69704	-133.04755
1	ID_-0.51	-0.51	29.29	2022	1	0.00046	0.69116	0.00032	0.00000	76.23920	...	48539.	6476.	54915.	5476.1	11.448	0.293	-30.51032	42.402	-138.63

	0_29.290_2022_01	000	000									73724	14732	70858	4716	44	12		59	282
2	ID_-0.510_29.29	-0.51000	29.29000	2022	2	0.00016	0.60511	0.00011	0.07987	-42.05534	...	341333.08047	8984.79570	39006.09375	7984.79570	10.75318	0.26713	39.08736	45.93648	-144.78499

	0_2022_02																			
3	ID_-0.510_29.290_2022	-0.51000	29.29000	2022	3	0.00035	0.69692	0.00024	0.20103	72.16957	...	50854.99108	6014.72406	57646.36837	5014.72412	11.76456	0.30468	-24.46513	42.14042	-135.02789

	_03																			
4	ID_-0.510_29.290_2022_04	-0.5100	29.2900	2022	4	-0.00032	0.58053	-0.00018	0.20435	76.19086	...	46594.68514	6849.28048	52896.54187	5849.28039	13.06532	0.28422	-12.90785	30.12264	-135.5012

5 rows × 75 columns

In [6]:

```
# Preview sample submission file
samplesubmission.head()
```

Out[6]:

	ID_LAT_LON_YEAR_WEEK	emission
0	ID_-0.510_29.290_2022_00	81.94000
1	ID_-0.510_29.290_2022_01	81.94000
2	ID_-0.510_29.290_2022_02	81.94000
3	ID_-0.510_29.290_2022_03	81.94000

4	ID_-0.510_29.290_2022_04	81.94000

In [7]:

```
# Check size and shape of datasets
train.shape, test.shape, samplesubmission.shape
```

Out[7]:

```
((79023, 76), (24353, 75), (24353, 2))
```

In [8]:

```
# Train to test sets ratio
(test.shape[0]) / (train.shape[0] + test.shape[0])
```

Out[8]:

```
0.23557692307692307
```

3. Statistical summaries

In [9]:

```
# Train statistical summary
train.describe(include = 'all')
```

Out[9]:

	ID_LAT_LON_YEAR_WEEK	latitude	longitude	year	week_no	SulphurDioxide_SO2_column_number_density	SulphurDioxide_SO2_column_number_density_amf	SulphurDioxide_SO2_slant_column_number_density	SulphurDioxide_cloud_fraction	SulphurDioxide_sensor_azimuth_angle	...	Cloud_cloud_top_height	Cloud_cloud_base_pressure	Cloud_cloud_base_height	Cloud_cloud_optical_depth	Cloud_surface_albedo	Cloud_sensor_azimuth_angle	Cloud_sensor_zenith_angle	Cloud_solar_azimuth_angle	Cloud_solar_zenith_angle
c	7	7	7	7	7	644	6441	6441	64	64	.	7	7	7	7	7	78	7	7	7

ount	9023	9023.00000	9023.00000	9023.00000	9023.00000	14.00000	4.00000	4.00000	414.00000	414.00000	...	8539.00000	8539.0000	8539.0000	8539.0000	8539.00000	539.00000	8539.0000	8539.0000	8539.0000
unique	79023	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	ID_-0.510_2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	9.290_2019_00																			
freq	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	-1.89107	29.88015	2020.000	26.00000	0.00005	0.83485	0.00004	0.15842	-7.92587	...	5592.377	59420.297	4670.4308	19.13924	0.27146	-10.78483	40.43698	-86.80058	27.92598

				00								48	46	7						
std	NaN	0.69452	0.81038	0.81650	15.29716	0.00027	0.18538	0.00021	0.07136	64.26337	...	1428.50300	9051.16361	1359.25158	13.54705	0.04943	30.37446	6.42822	37.83727	4.40384
min	NaN	-3.29900	28.22800	2019.00000	0.00000	-0.00100	0.24182	-0.00089	0.00000	-179.53706	...	1050.66178	24779.03370	1050.49682	1.84453	0.01770	-102.73973	2.99887	-153.46421	10.81829

25%	NaN	-2.45100	29.26200	2019.00000	13.00000	-0.00010	0.70582	-0.00008	0.110553	-56.78238	...	4595.40052	53175.77993	3680.85634	9.97457	0.24145	-30.30917	35.82991	-125.99116	24.68676
50%	NaN	-1.88200	29.88300	2020.00000	26.00000	0.00002	0.80912	0.00002	0.161185	-12.44173	...	5573.85431	59332.53255	4621.75517	15.13069	0.27275	-12.67391	41.11963	-84.64435	28.33363
75	Na	-1.	30.	202	39.	0.00015	0.94279	0.00012	0.2118	72.0599	..	654	656	557	23.7	0.3	9.4022	44.4	-48.1	31.4

%	N	30300	47100	1.00000	00000				2	9	.	2.30364	63.84268	2.98322	8503	0289	0	4627	3270	9988
max	NaN	-0.510000	31.532000	2021.00000	52.000000	0.00419	1.88524	0.00424	0.30000	122.09520	...	12384.23946	89291.61558	11384.23946	250.00000	0.73651	78.22304	65.95125	-22.65317	42.06044

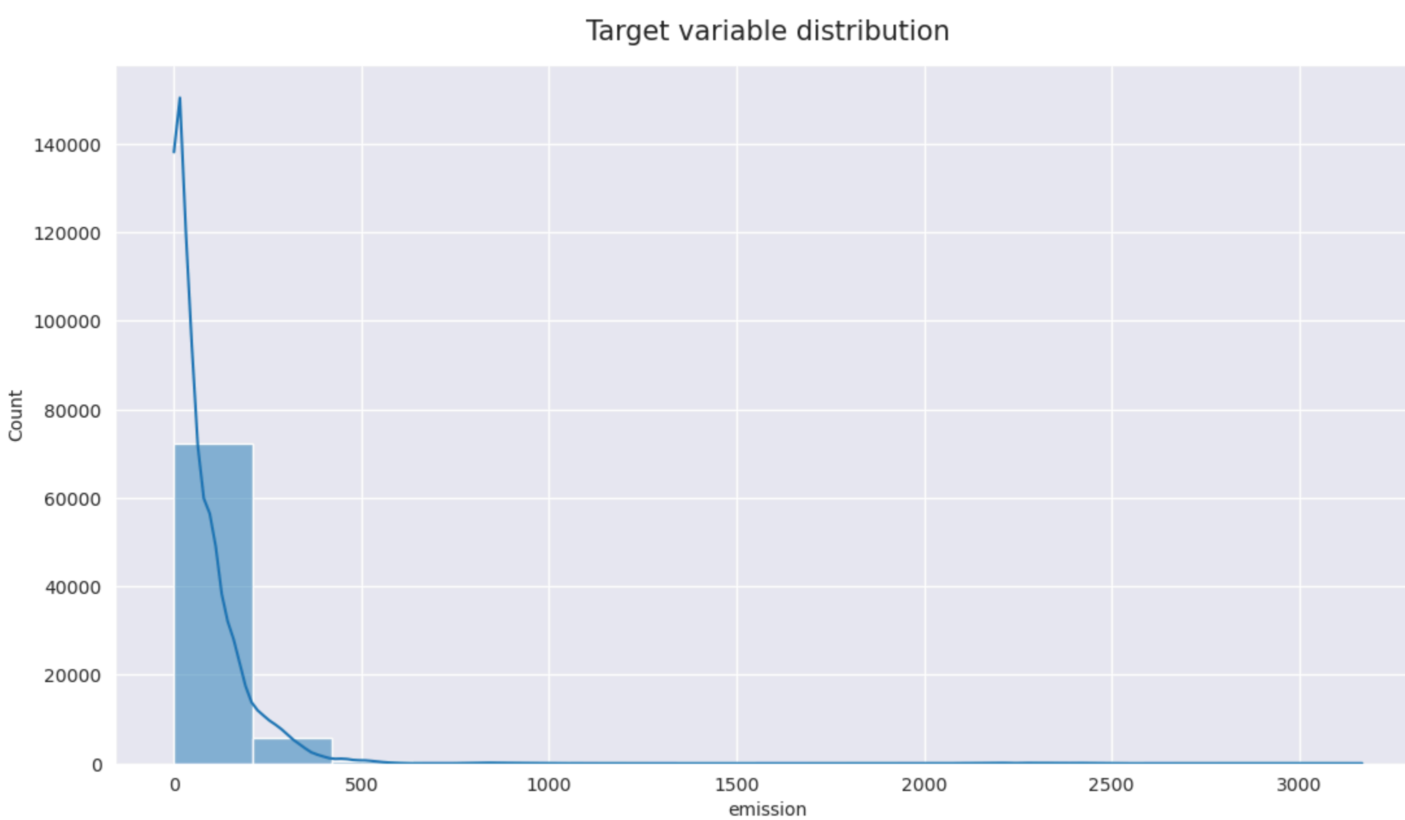
11 rows × 76 columns

From the above statistical summary, we can deduce some of the following insights:

- The train data provided ranges from year *2019 to 2021*
- Minimum recorded CO_2 emissions is *0.32064* and a maximum of *3167.76800*
- Week of the year starts from 0 to 52

- The latitude and longitudes ranges show that the regions are mostly within Rwanda

```
In [10]:
# Target variable distribution
sns.set_style('darkgrid')
plt.figure(figsize = (13, 7))
sns.histplot(train.emission, kde = True, bins = 15)
plt.title('Target variable distribution', y = 1.02, fontsize =
15)
display(plt.show(), train.emission.skew())
```



```
None
```

```
10.173825825101622
```

The target variable is skewed to the right with a a degree of ~7.

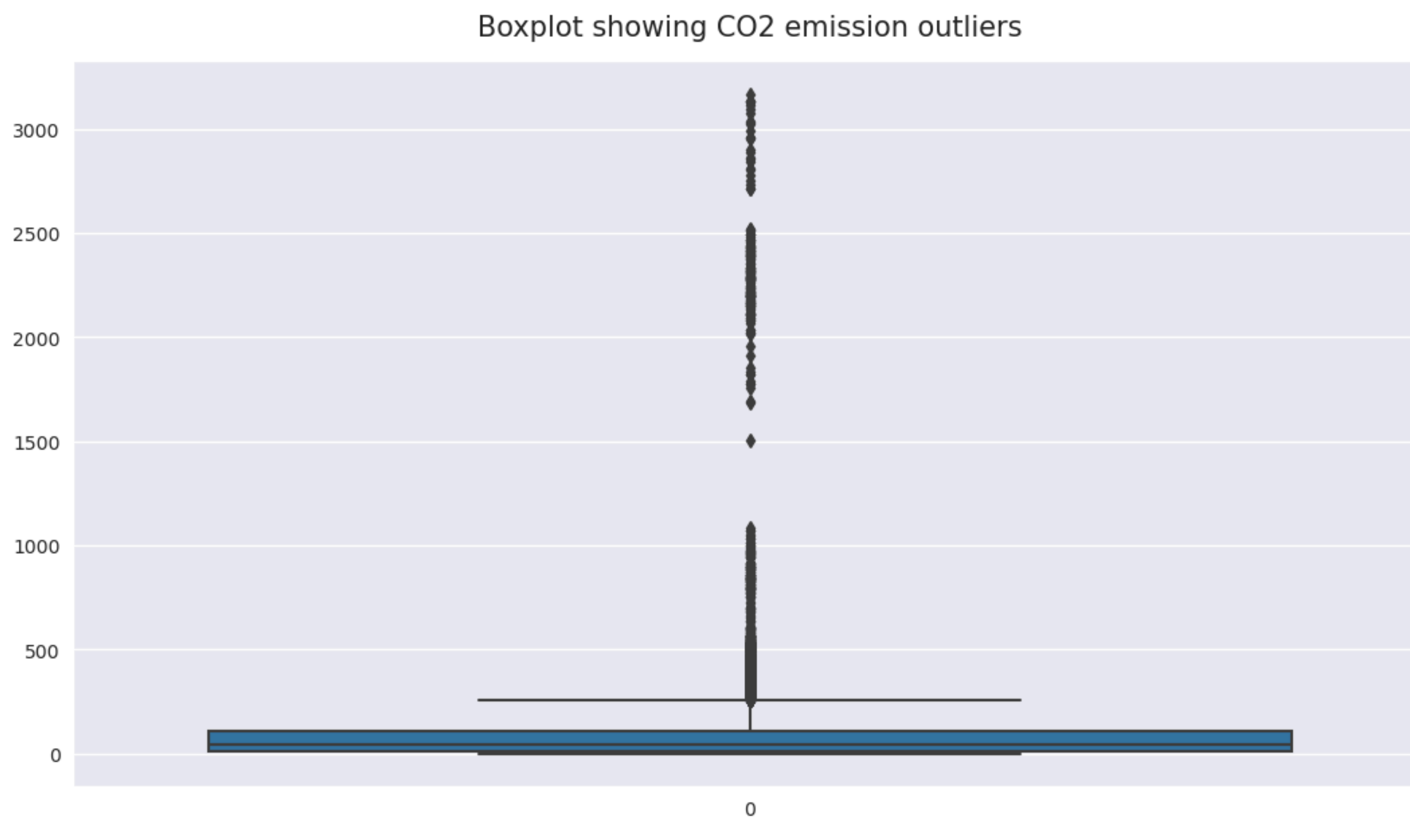
Some of the techniques used to handle skewness include:

- Log transform
- Box-cox transform
- Square root transform
- *etc*

4. Outliers

In [11]:

```python
# Plotting boxplot for the CO2 emissions
sns.set_style('darkgrid')
plt.figure(figsize = (13, 7))
sns.boxplot(train.emission)
plt.title('Boxplot showing CO2 emission outliers', y = 1.02,
fontsize = 15)
plt.show()
```



Outliers are those data points which differ significantly from other observations present in given dataset.

Suggestions on how to handle outliers:

- Transforming the outliers by scaling - log transformation, box-cox transformation ...
- Dropping outliers
- Imputation by replacing outliers with mean, median ...

5. Geo Visualisation - EDA

In [12]:

```
# Combine train and test for easy visualisation
```

```
train_coords = train.drop_duplicates(subset = ['latitude',
'longitude'])
test_coords = test.drop_duplicates(subset = ['latitude',
'longitude'])
train_coords['set_type'], test_coords['set_type'] = 'train',
'test'

all_data = pd.concat([train_coords, test_coords], ignore_index
= True)
# Create point geometries

geometry = gpd.points_from_xy(all_data.longitude,
all_data.latitude)
geo_df = gpd.GeoDataFrame(
    all_data[["latitude", "longitude", "set_type"]],
geometry=geometry
)

# Preview the geopandas df
geo_df.head()
```

Out[12]:

	latitude	longitude	set_type	geometry
0	-0.51000	29.29000	train	POINT (29.29000 -0.51000)
1	-0.52800	29.47200	train	POINT (29.47200 -0.52800)
2	-0.54700	29.65300	train	POINT (29.65300 -0.54700)
3	-0.56900	30.03100	train	POINT (30.03100 -0.56900)
4	-0.59800	29.10200	train	POINT (29.10200 -0.59800)

In [13]:

```
# Create a canvas to plot your map on
all_data_map = folium.Map(prefer_canvas=True)
```

```
# Create a geometry list from the GeoDataFrame
geo_df_list = [[point.xy[1][0], point.xy[0][0]] for point in
geo_df.geometry]

# Iterate through list and add a marker for each volcano,
color-coded by its type.
i = 0
for coordinates in geo_df_list:
    # assign a color marker for the type set
    if geo_df.set_type[i] == "train":
        type_color = "green"
    elif geo_df.set_type[i] == "test":
        type_color = "orange"

    # Place the markers
    all_data_map.add_child(
        folium.CircleMarker(
            location=coordinates,
            radius = 1,
            weight = 4,
            zoom =10,
            popup=
            "Set: " + str(geo_df.set_type[i]) + "<br>"
            "Coordinates: " + str([round(x, 2) for x in
```

```
geo_df_list[i]]),
            color =  type_color),
        )
    i = i + 1
all_data_map.fit_bounds(all_data_map.get_bounds())
all_data_map
```

Out[13]:

Make this Notebook Trusted to load map: File -> Trust Notebook

6. Missing values and duplicates

In [14]:

```
# Check for missing values
train.isnull().sum().any(), test.isnull().sum().any()
```

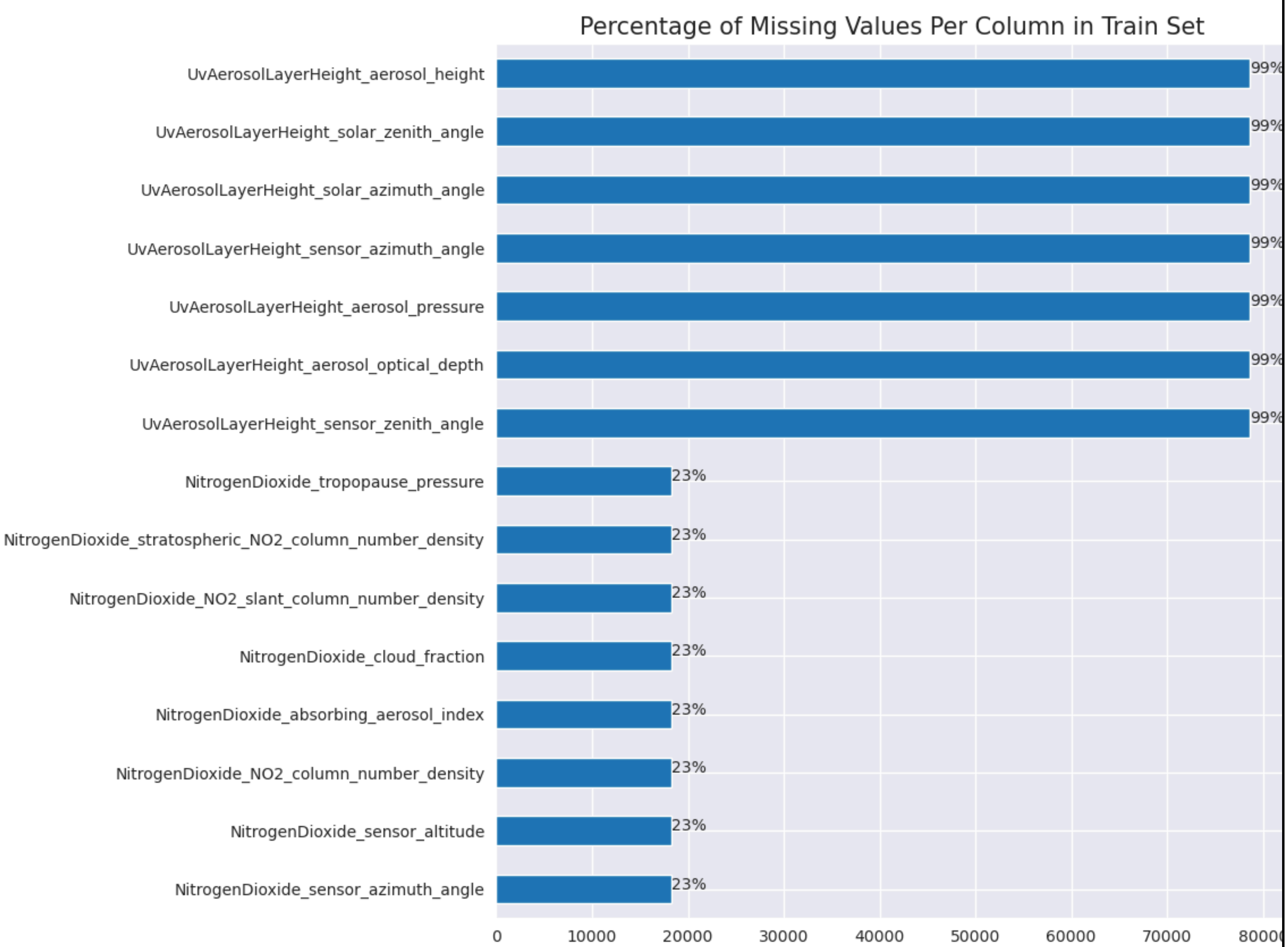
Out[14]:

```
(True, True)
```

In [15]:

```
# Plot missing values in train set
ax = train.isna().sum().sort_values(ascending =
False)[:15][::-1].plot(kind = 'barh', figsize = (9, 10))
```

```
plt.title('Percentage of Missing Values Per Column in Train
Set', fontdict={'size':15})
for p in ax.patches:
    percentage
='{:,.0f}%'.format((p.get_width()/train.shape[0])*100)
    width, height =p.get_width(),p.get_height()
    x=p.get_x()+width+0.02
    y=p.get_y()+height/2
    ax.annotate(percentage,(x,y))
```



Suggestions on how to handle missing values:

- Fill in missing values with mode, mean, median..
- Drop Missing datapoints with missing values
- Fill in with a large number e.g -999999

In [16]:

```
# Check for duplicates
train.duplicated().any(), test.duplicated().any()
```

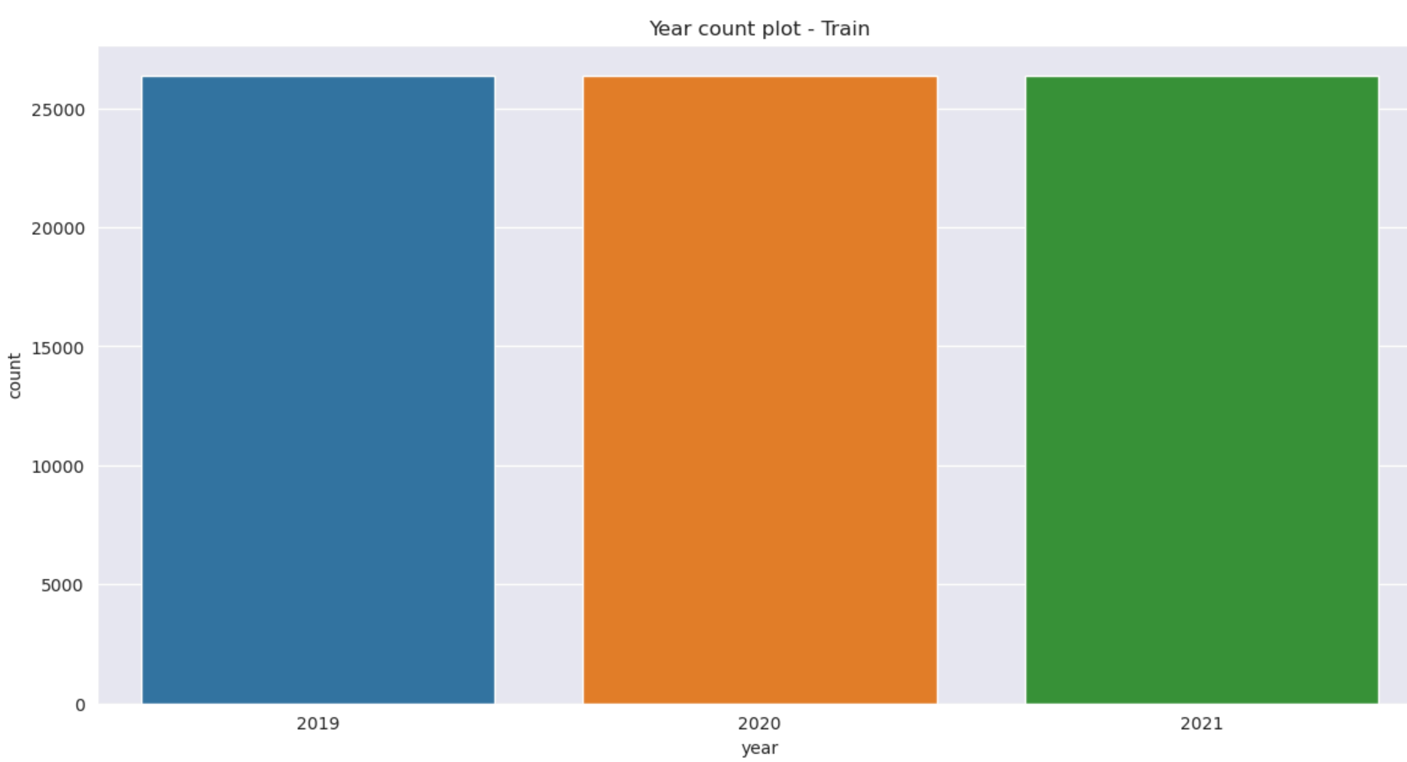
Out[16]:

```
(False, False)
```

7. Date features EDA

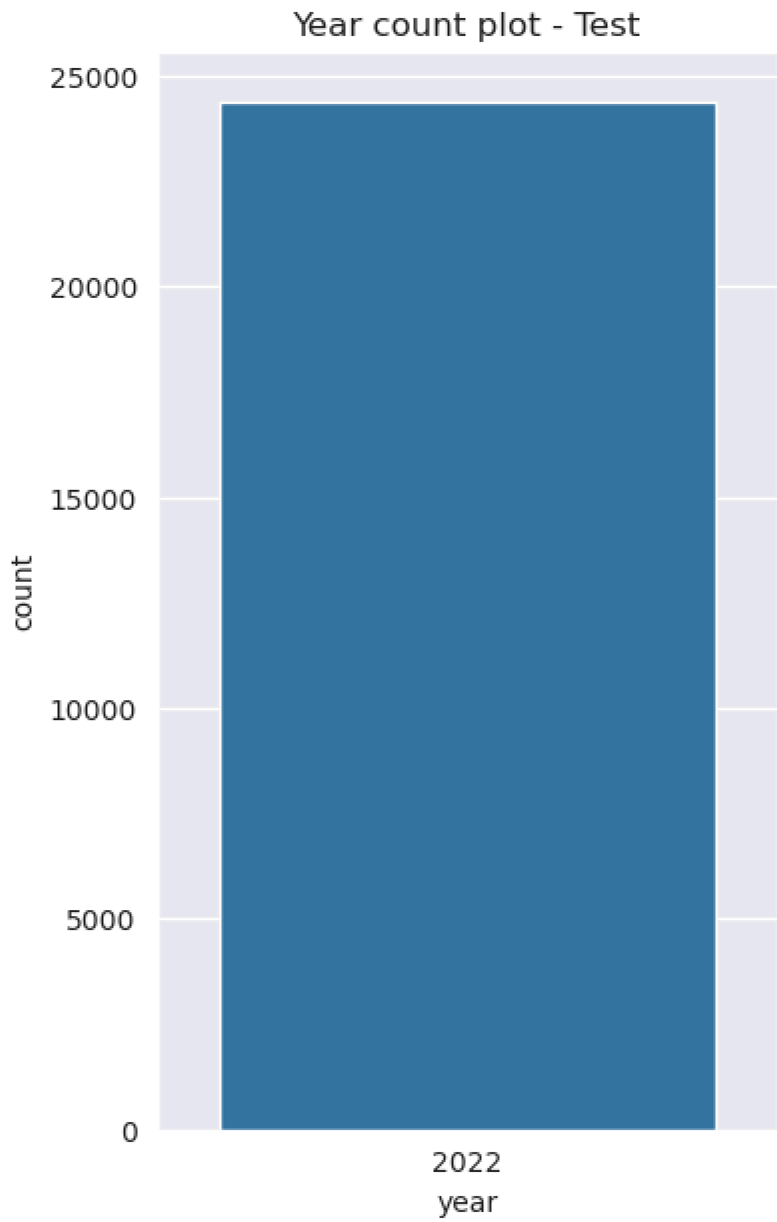
In [17]:

```
# Year countplot
plt.figure(figsize = (14, 7))
sns.countplot(x = 'year', data = train)
plt.title('Year count plot - Train')
plt.show()
```



In [18]:

```
# Year countplot
plt.figure(figsize = (4, 7))
sns.countplot(x = 'year', data = test)
plt.title('Year count plot - Test')
plt.show()
```

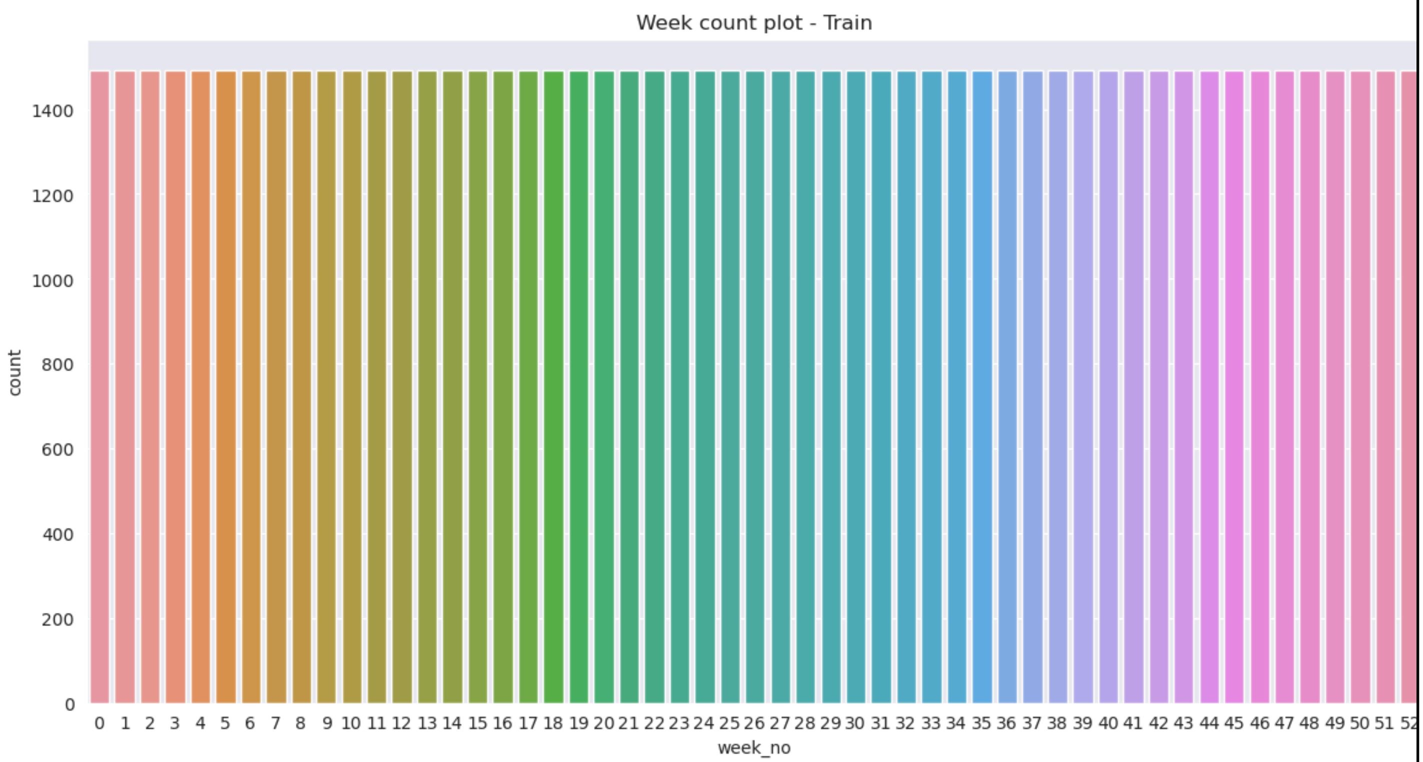


- The number of observations of CO2 emissions are the same across the years (2019, 2020, 2021)
- Year 2022 (in the test set) has fewer number of observations

In [19]:

```
# Week countplot
plt.figure(figsize = (14, 7))
sns.countplot(x = 'week_no', data = train)
```

```
plt.title('Week count plot - Train')
plt.show()
```



- The number of observations of CO2 emissions are relatively the same across the weeks

In [20]:

```
train.drop_duplicates(subset = ['year',
'week_no']).groupby(['year'])[['week_no']].count()
```

Out[20]:

	week_no
year	
2019	53
2020	53
2021	53

8. Correlations - EDA

In [21]:

```
# Top 20 correlated features to the target
top20_corrs =
abs(train.corr()['emission']).sort_values(ascending =
```

```
False).head(20)
top20_corrs
```

```
Out[21]:
emission
1.00000
longitude
0.10275
UvAerosolLayerHeight_aerosol_height
0.06901
UvAerosolLayerHeight_aerosol_pressure
0.06814
Cloud_surface_albedo
0.04659
CarbonMonoxide_H2O_column_number_density
0.04322
CarbonMonoxide_CO_column_number_density
0.04133
Formaldehyde_tropospheric_HCHO_column_number_density_amf
0.04026
UvAerosolLayerHeight_aerosol_optical_depth
0.04016
UvAerosolLayerHeight_sensor_azimuth_angle
0.03514
```

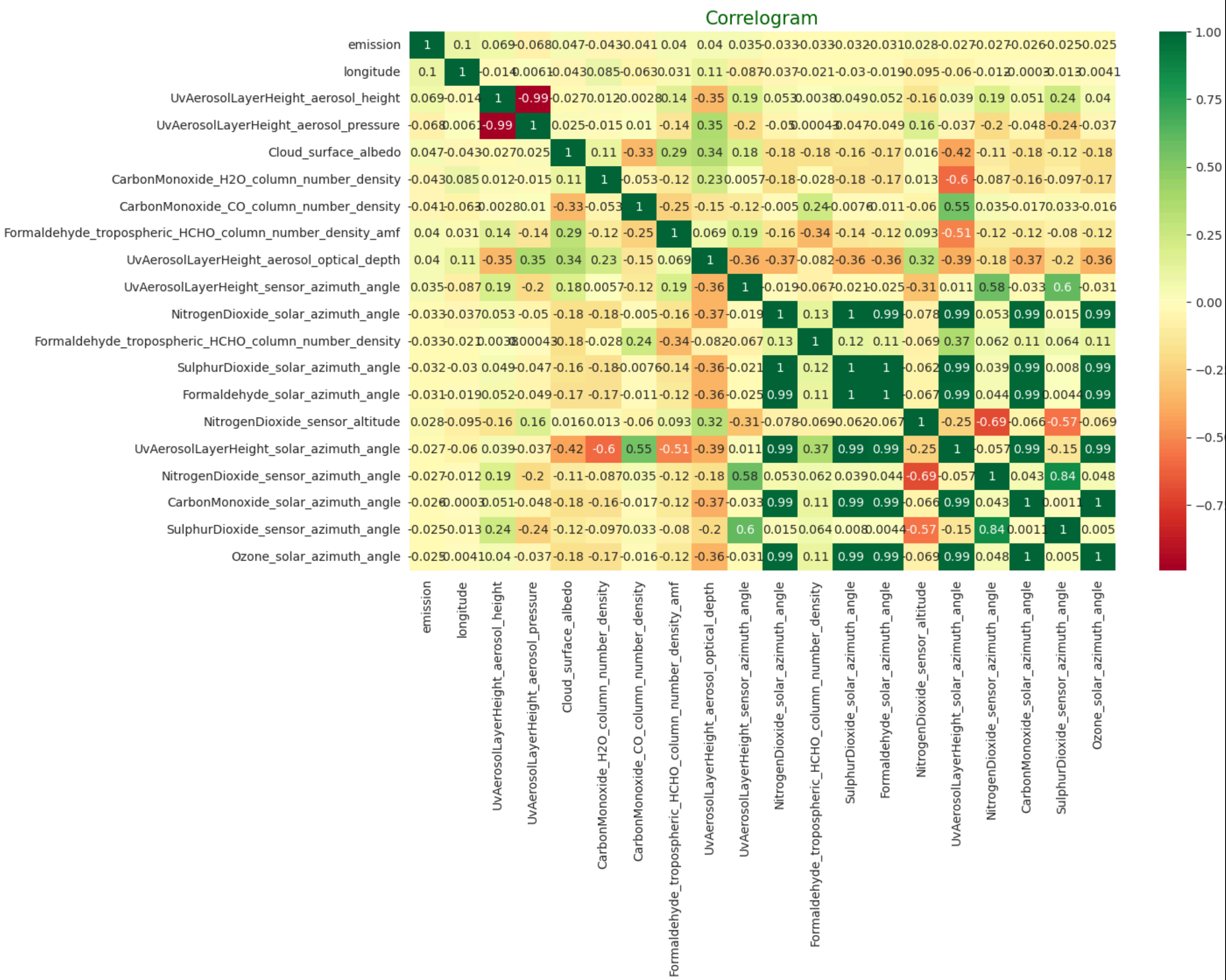
```
NitrogenDioxide_solar_azimuth_angle
0.03342
Formaldehyde_tropospheric_HCHO_column_number_density
0.03333
SulphurDioxide_solar_azimuth_angle
0.03234
Formaldehyde_solar_azimuth_angle
0.03081
NitrogenDioxide_sensor_altitude
0.02754
UvAerosolLayerHeight_solar_azimuth_angle
0.02721
NitrogenDioxide_sensor_azimuth_angle
0.02710
CarbonMonoxide_solar_azimuth_angle
0.02628
SulphurDioxide_sensor_azimuth_angle
0.02508
Ozone_solar_azimuth_angle
0.02485

Name: emission, dtype: float64
```

In [22]:

```
# Quantify correlations between features
corr = train[list(top20_corrs.index)].corr()
```

```python
plt.figure(figsize = (13, 8))
sns.heatmap(corr, cmap='RdYlGn', annot = True, center = 0)
plt.title('Correlogram', fontsize = 15, color = 'darkgreen')
plt.show()
```



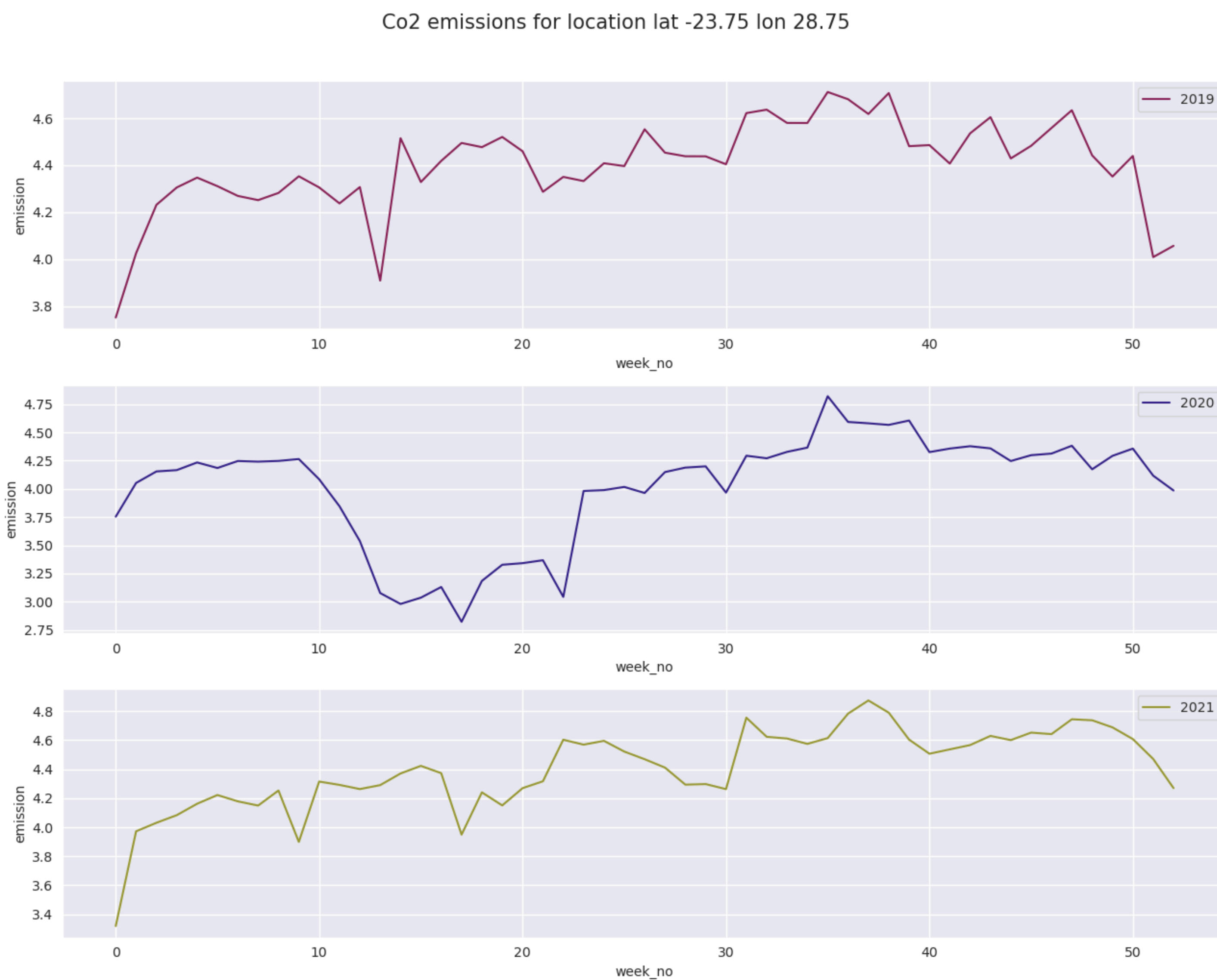
9. Timeseries visualization - EDA

In [23]:

```
linkcode
# Sample a unique location and visualize its emissions across the years
train.latitude, train.longitude = round(train.latitude, 2), round(train.longitude, 2)
sample_loc = train[(train.latitude == -0.510) & (train.longitude == 29.290)]

# Plot a line plot
sns.set_style('darkgrid')
fig, axes = plt.subplots(nrows = 3, ncols = 1, figsize = (13, 10))
fig.suptitle('Co2 emissions for location lat -23.75 lon 28.75', y=1.02, fontsize = 15)

for ax, data, year, color, in zip(axes.flatten(), sample_loc, sample_loc.year.unique(), ['#882255','#332288', '#999933' , 'orangered']):
  df = sample_loc[sample_loc.year == year]
  sns.lineplot(x=df.week_no,y= df.emission, ax = ax, label = year, color = color)
plt.legend()
plt.tight_layout()
```



[Reference code](#)