



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, CO2 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<sub>2</sub> 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 reproducibility
```

```
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]:

[illegible]

	A R - W E E K					ity	amf	sity	_fr ac tio n	ang le		o p - h ei g ht	ss ur e	b a s e - h ei g ht	pti ca l_ d e pt h	c e - a l b e d o	h_ an gl e	nit h - a n gl e	ut h - a n gl e
0	ID - 0. 51 0_ 29 .2 90 _2 01 9_ 00	- 0 . . 5 2 1 9 0 0 0 0 0 0	2 9 2 1 0 0 0			-0.0 0011	0.603 02	-0.00 007	0. 25 56 7	-98. 593 89	.	3 6 6 4. 4 3 6 2 2	61 08 5. 80 95 7	2 6 1 5. 5 1 2 8 5 4 8	1 5. 5 6 8 2 5 3 9	0 . -1 2. 62 89 9		3 5. 6 3 2 4 2	-1 3 8. 7 8 6 4 2
1	ID - 0. 0.	- 0 . . 0 9 . . 1	2 0 1	2 0 1		0.00 002	0.728 21	0.000 01	0. 13 09	16. 592 86	.	3 6 5	66 96 9.	3 1 7	8. 6 9	0 . 2 30 .3 59	3 9. 5	-1 4 5.	

	51 0_ 29 .2 90 _2 01 9_ 01	5 1 0 0 0 0 0 0	2 9 0 0 0 0 0 0	9				9			1. 1 9 0 3 1	47 87 3  4 2	4. 5 7 2 4 2	0 6 0  3	5 6 8 3	38	5 7 6 3 3	1 8 3 9 3	
2	ID _ 0. 51 0_ 29 .2 90 _2 01 9_ 02	- 0 . . 5 2 1 9 0 0 0 0 0 0	2 9 2 0 1 9 0 0 0 0 0	2	0.00 051	0.748 20	0.000 38	0. 11 00 2	72. 795 84	.	4 2 1 6. 9 8 6 4 9	60 06 8. 89 44 5	3 5 1 6. 2 8 2 6 7	2 1. 1 0 3 4 1 0	0 . . 2 5 1 1 0	15 .3 77 88	3 0. 4 0 1 8 2	-1 4 2. 5 1 9 5 4	
3	ID _ 0.	- 0 . . 2 9 0 0 1	2 9 0 0 1	3	NaN	NaN	NaN	N a N	Na N	.	5 2 2	51 06 4.	4 1 8	1 5. 3	0 . . 2	-1 1. 29	2 4. 3	-1 3 2.	

	51	5	2	9							8.	54	0.	8	6	34	8	6	
	0_	1	9								5	73	9	6	2	0	0	6	
	29	0	0								0	4	7	9	0		3	5	
	.2	0	0								7		3	0	4		6	8	
	90	0	0								7		3					3	
	_2										4		2						
	01																		
	9_																		
	03																		
4	ID																		
	-										3		3						
	0.	-	2								9		3		0		3		-1
	51	0	9								8	63	5				7.		4
	0_	.	.	2				0.			0.	75	5.	8.	.	38	3		1.
	29	5	2	0		-0.0	0.676	-0.00	12	4.1	.	1.	7	11	2	.5	9	5	
	.2	1	9	1	4	0008	30	005	11	212	.	12	7	4	3	32	2	0	
	90	0	0	9					6	7	.	57	1	6	5	26	9	9	
	_2	0	0								8	8	0	9	8		9	8	
	01	0	0								1		1		5		8		
	9_										2		1					1	
	04																		

5 rows × 76 columns

In [5]:

# Preview test dataset

ID	LAT	LONG	YEAR	WEEK	SulfurDioxide_SO2_column_number_density	SulfurDioxide_SO2_slant_column_number_density_amf	SulfurDioxide_sensor_azimuth_angle	Cloud_fraction	Cloud_optical_depth	Cloud_pressure_height	Cloud_base_pressure_height	Cloud_sulfate_aerosol_becdo	Cloud_sensor_azimuth_angle	Cloud_sensor zenith angle
----	-----	------	------	------	---	---	------------------------------------	----------------	---------------------	-----------------------	----------------------------	-----------------------------	----------------------------	---------------------------

0	ID - 0. 5 1 0 - 2 9. 2 9 0 - 2 0 2 2 - 0 0	- 0 . 5 1 0 0 0 0 0	2 9 . 2 9 0 0 0 0	2 0 . 2 2 0 0 0	0	NaN	NaN	NaN	NaN	NaN	NaN	3 6 0 2 2 0 2 7 3 4	8 4 7 2 . 3 1 3 4 8	4 1 0 4 7. 3 9 3 7 5 0	7 4 7 2. 3 1 3 4 8	7. 9 3 5 6 2	0 . 2 4 0 7 7	-1 00 .1 13 79	3 3. 6 9 7 0 4	-1 3 3. 0 4 7 5 5
	1	ID - 0. 5 1	- 0 . 5 1	2 9 . 2 2	2 0 . 2 2	1	0.00 046	0.69 116	0.000 32	0. 00 00 0	76. 239 20	4 8 5 3 9.	6 4 7 6 .	5 4 9 1 5.	5 4 7 6. 1	1 1. 4 4 8	0 . 2 9 3	-3 0. 51 03 2	4 2. 4 0 2	-1 3 8. 6 3

	0 — 2 9. 2 9 0 — 2 0 2 2 — 0 1	0 0 0 0								7 3 7 2 4	1 4 7 3 2	7 0 8 5 8	4 7 1 6	4 4	1 2		5 9	2 8 2
2	ID — 0. 5 1 0 — 2 9. 2 9	- 0 . 5 1 0 — 2 9. 2 9	2 9 . 2 0 2 0 0 0	2 0 2	0.00 016	0.60 511	0.000 11	0. 07 98 7	-42 .05 534	. . . . .	3 4 1 3 3 0 8 0 4 7	8 9 8 4 . . 7 0 9 5 7 0	3 9 0 6 . 0 9 9 3 7 5	7 9 8 4. 7 5 3 1 8 0	1 0 . 2 6 7 1 3	39 .0 87 36	4 5. 9 3 6 4 8	-1 4 4. 7 8 4 9 9

[illegible]

	03																		
4	ID																		
	-																		
	0.																		
	5																		
	1																		
	0									4	6	5							
	-	-	2							6	8	2	5						
	2	0	9							5	4	8	8	1	0			3	-1
	9.	.	.	2						9	9	9	4	3.	.	-1	0.	3	5.
	2	5	2	0	-0.0				0.	4.	.	6.	9.	0	2	2.	1	5	5.
	9	1	9	2	003	0.58	-0.00	20	76.	.	.	5	2	6	8	90	2	0	5
	0	0	0	2	2	053	018	43	190	.	6	2	8	5	4	78	2	0	0
	-	0	0					5	86	.	8	8	4	0	3	5	6	0	1
	2	0	0								5	0	1	3	2		4	1	2
	0										4	8	7	9					
	2																		
	2																		
	-																		
	0																		
	4																		

5 rows × 75 columns

In [6]:

```
# Preview sample submission file
```

```
samplesubmission.head()
```

Out[6]:

	ID_LAT_LON_YE AR_WEEK	emiss ion
0	ID_-0.510_29.290 _2022_00	81.94 000
1	ID_-0.510_29.290 _2022_01	81.94 000
2	ID_-0.510_29.290 _2022_02	81.94 000
3	ID_-0.510_29.290 _2022_03	81.94 000

4	ID_-0.510_29.290 _2022_04	81.94 000
---	------------------------------	--------------

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')
```

[illegible]



9. 2 9 0 - 2 0 1 9 - 0 0																				
f r e q	1	N a N	N a N	N a N	N a N	NaN	NaN	NaN	N a N	Na N	. . .	N a N	N a N	N a N	N a N	N a N	N a N	N a N	N a N	
m e a n	NaN	- 1 8 9 1 0 7	2 9 8 0 1 5	2 0 2 0 0 0	2 6 0 0 0 0	0.00 005	0.83 485	0.00 004	0. 15 84 2	-7. 92 58 7	. . . 3 7 7	5 5 9 2 . 3 7 7	5 9 4 2 0. 2 9 7	4 6 7 0. 4 3 0 8	1 9. 1 3 9 2 4	0 . 2 7 1 4 6	-1 0. 78 48 3	4 0. 4 3 6 9 8	-8 6. 8 0 0 5 8	2 7. 9 2 5 9 8

				00						48	46	7						
standard	NN	69452	81038	81650	152	0.00027	0.18538	0.00021	0.07136	64.26337	.8150300	905125318	139.255158	1044943	30.37446	6.42822	37.83727	4.40384
mean	NN	-322920000000000	-281900000000000	201900000000000	000	-0.00100	0.24182	-0.00089	0.00000	-179.53706	.666178	2479.049682	1050.4453	001770	-102.739	2.99887	-153.46421	10.81829

25%	N a N	-24510000000	29139000000	20139000000	-0.00010	0.70582	-0.00008	0.11053	-56.78238	.	459540052	531754793	368085634	99745	024145	-3082991	-1259916	2468676
50%	N a N	-18820000000	19260000000	20260000000	0.00002	0.80912	0.00002	0.16185	-12.44173	.	55733285431	59332532515	462175517	1530695	02721	-126391	419635	-8464363
75%	N a	-1002	3009	239	0.00015	0.94279	0.00012	0.2118	72.0599	.	654	656	557	237	03	94022	444	-481

%	N	3	4	1	0				2	9	.	2	6	2.	8	0	0	4	3	9
		0	7	.	0						.	.	3.	9	5	2		6	2	9
		3	1	0	0						3	8	8	0	8			2	7	8
		0	0	0	0						0	4	3	3	9			7	0	8
		0	0	0	0						3	2	2							
				0							6	6	2							
				0							4	8								
m	N	-	3	0	5				0.	12	.	1	8	1						
a	a	0	1	2	2				30	2.0	.	2	9	1	2	0		6	-2	4
x	N	.	.	1	.				00		.	8	2	3	5	.	78	5.	2.	2.
		5	5	.	0	0.00	1.88	0.00	30	2.0	.	4	9	8	0.	7	78	9	6	0
		1	3	0	0	419	524	424	00	95	.	.	1.	4.	0	3	.2	5	5	6
		0	2	0	0				0	20	.	2	6	2	0	6	23	1	3	0
		0	0	0	0						3	1	3	0	0	5	04	2	1	4
		0	0	0	0						9	5	9	0	0	1		5	7	4
				0							4	5	4	0						
				0							6	8	6							

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 CO2 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
```

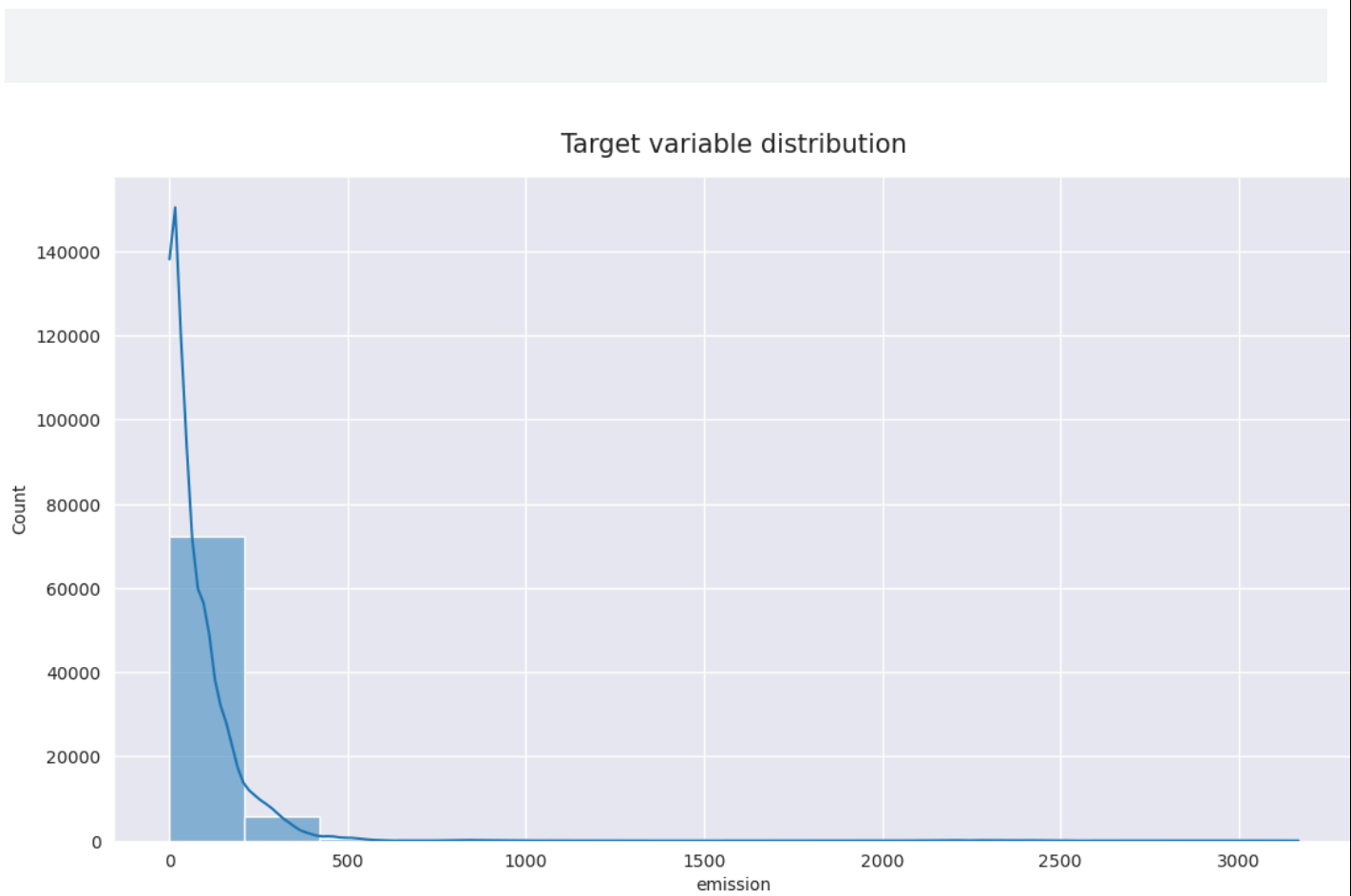
```
sns.set_style('darkgrid')
```

```
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 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]:

```
# Plotting boxplot for the CO2 emissions
```

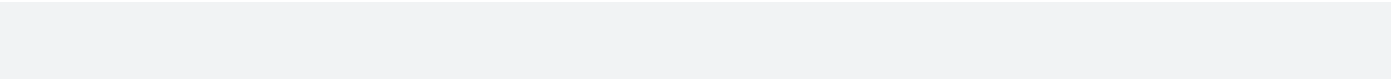
```
sns.set_style('darkgrid')
```

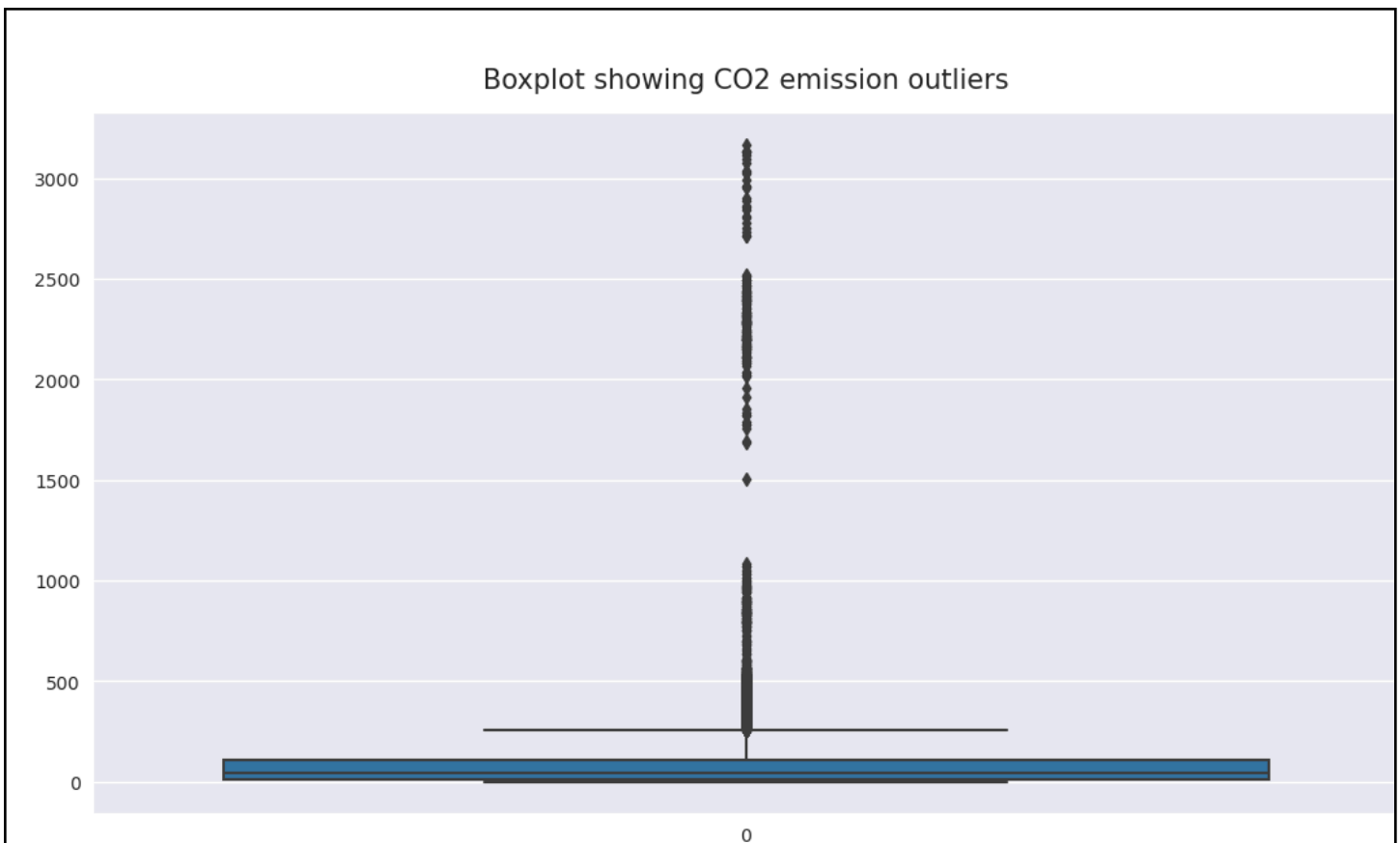
```
plt.figure(figsize = (13, 7))
```

```
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]:

	latitu de	longit ude	set_t ype	geometry
0	-0.51 000	29.29 000	train	POINT (29.29000 -0.51000)
1	-0.52 800	29.47 200	train	POINT (29.47200 -0.52800)
2	-0.54 700	29.65 300	train	POINT (29.65300 -0.54700)
3	-0.56 900	30.03 100	train	POINT (30.03100 -0.56900)
4	-0.59 800	29.10 200	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]:

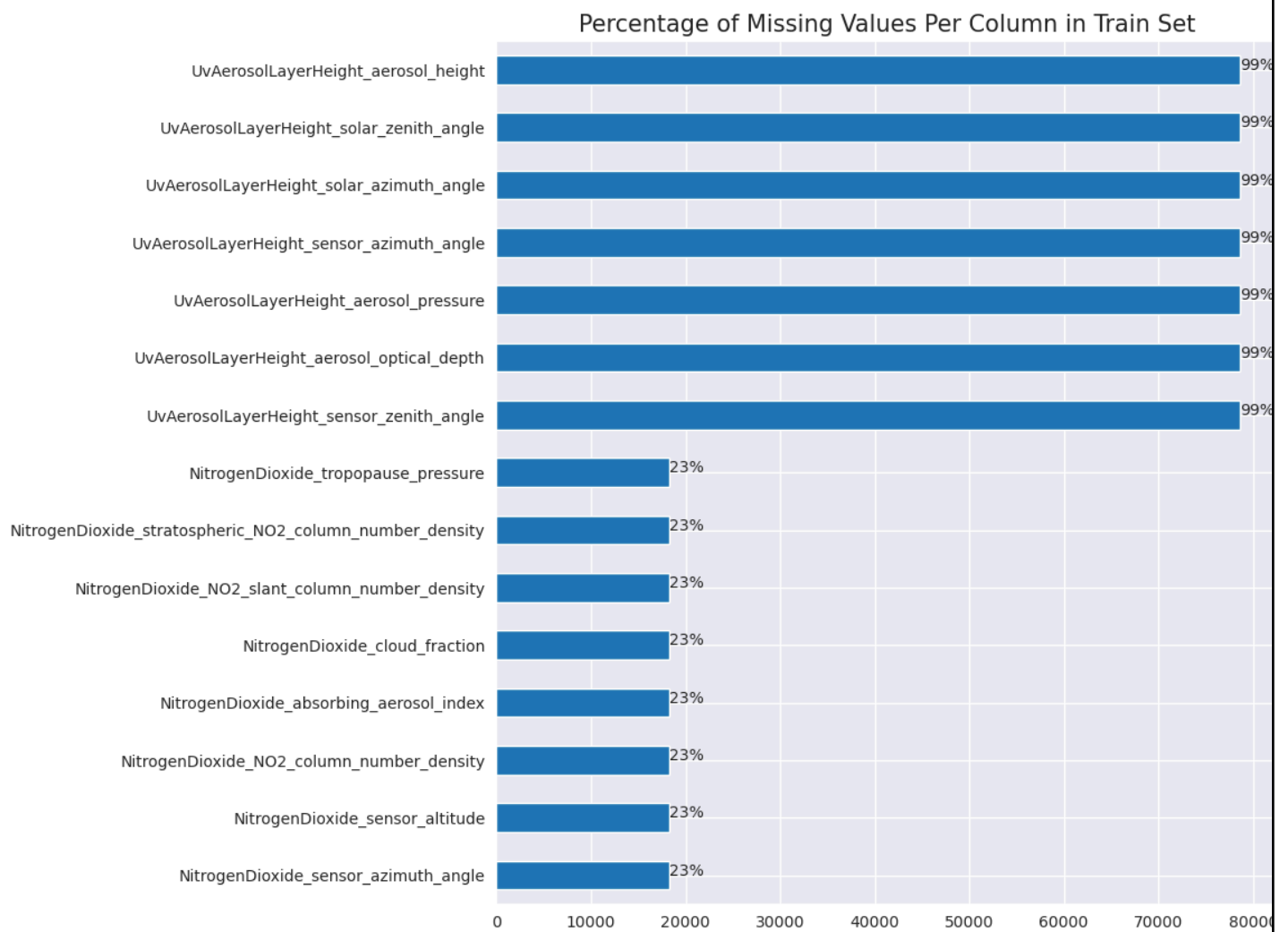
```
(True, True)
```

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]:

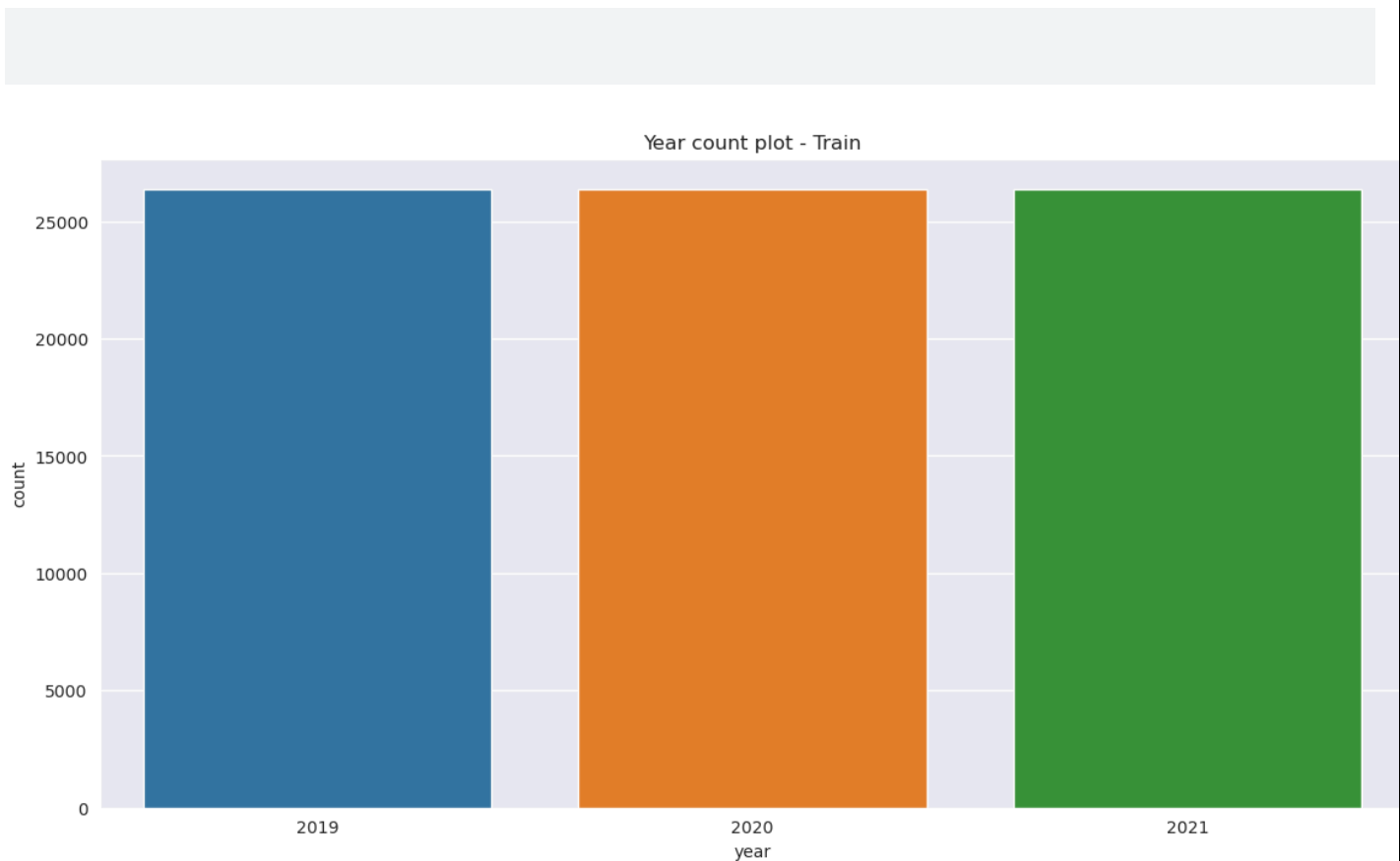
```
# Year countplot
```

```
plt.figure(figsize = (14, 7))
```

```
sns.countplot(x = 'year', data = train)
```

```
plt.title('Year count plot - Train')
```

```
plt.show()
```



In [18]:

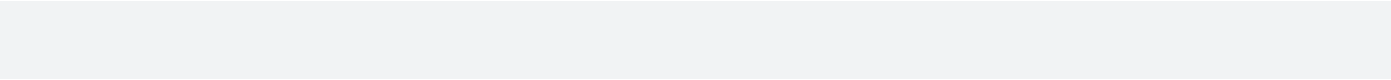
```
# Year countplot
```

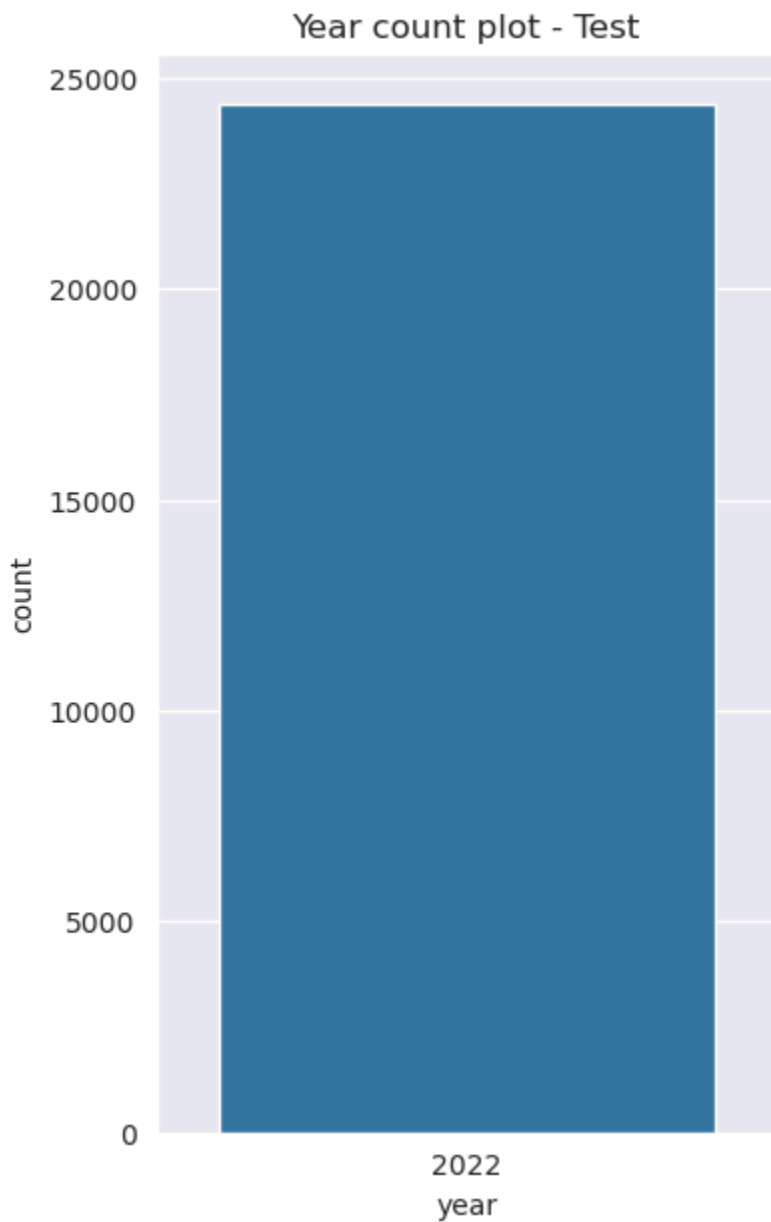
```
plt.figure(figsize = (4, 7))
```

```
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

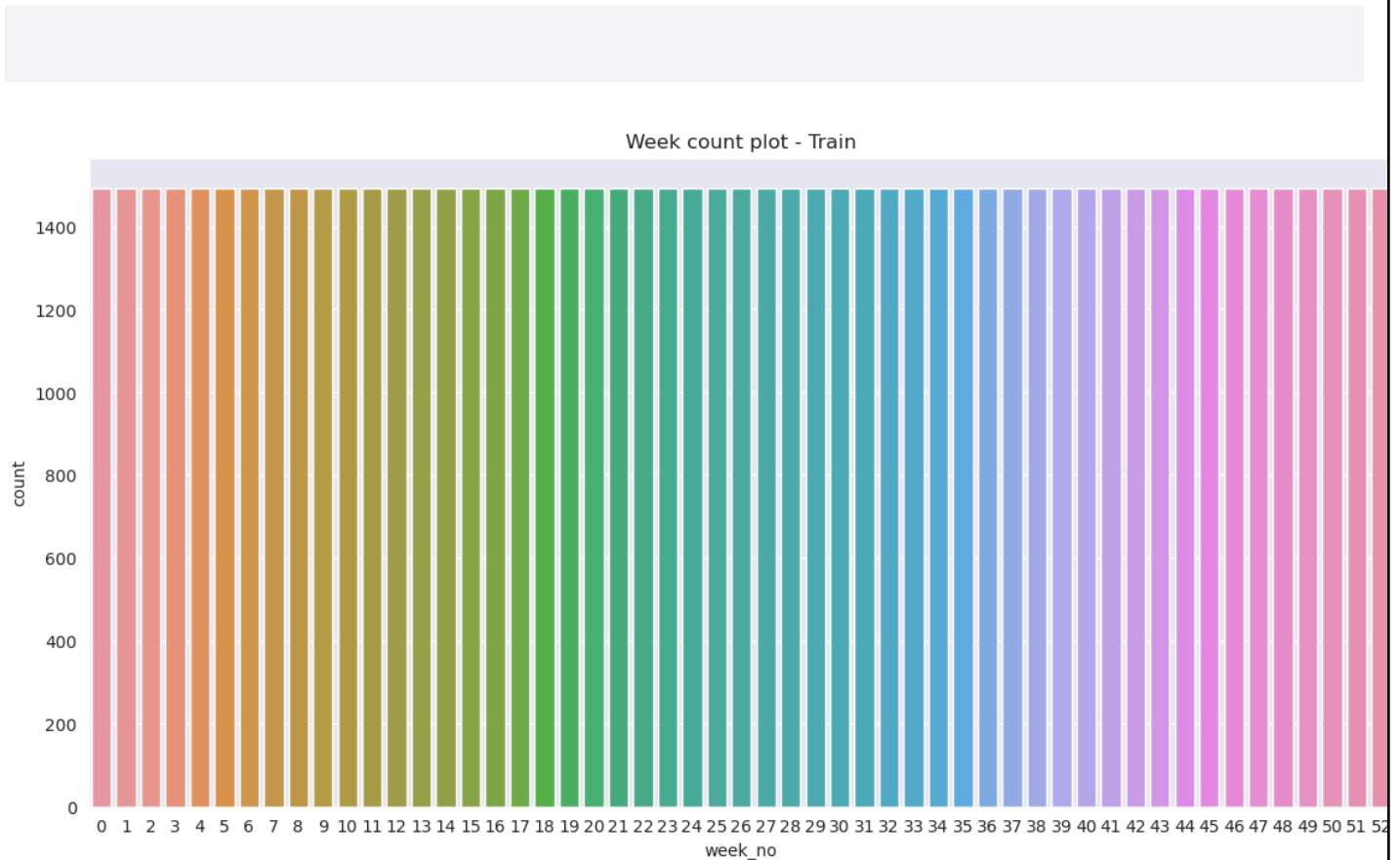
In [19]:

```
# 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
ye ar	
20 19	53
20 20	53
20 21	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()
```

```
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']):
```

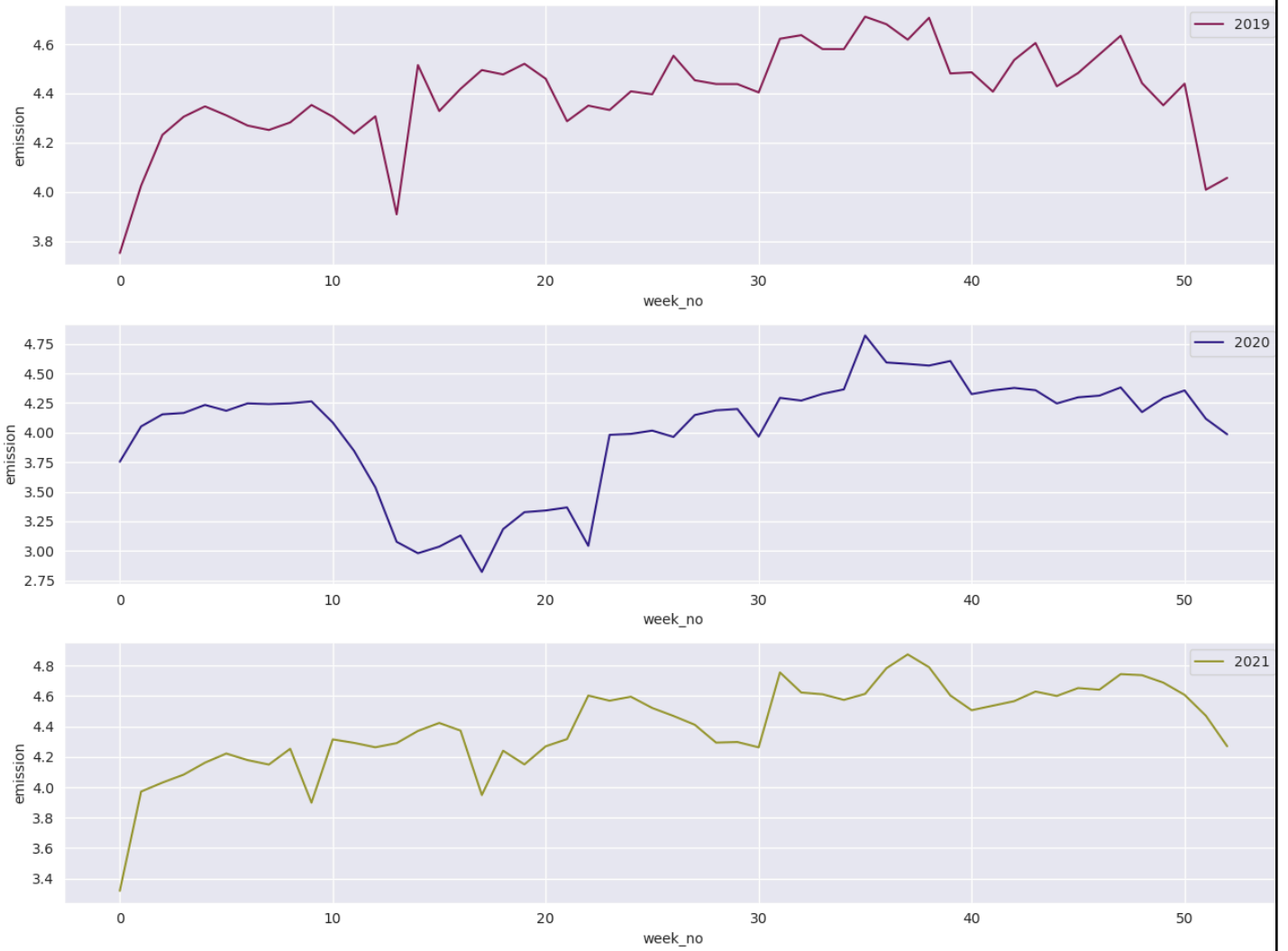
```
    df = sample_loc[sample_loc.year == year]
```

```
    sns.lineplot(x=df.week_no, y= df.emission, ax = ax, label =  
year, color = color)
```

```
plt.legend()
```

```
plt.tight_layout()
```

Co2 emissions for location lat -23.75 lon 28.75



[Reference code](#)