CO_2 Emissions Analysis and Forecasting Using Machine Learning :

A Comparative Study of ARIMA, SARIMA, LSTM, and Transformer Models

Table des matières

Chapitre 1

Report

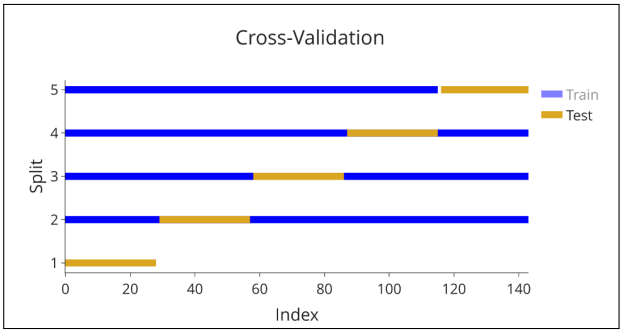
1 Introduction

In this project, we focus on time series data related to CO_2 emissions. We aim to apply, compare, and evaluate different time series forecasting models to better understand their strengths, limitations, and suitability for capturing and predicting emission patterns over time.

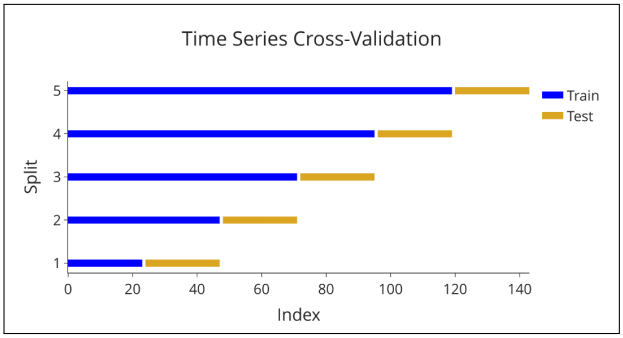
2 Cross-Validation

In time series analysis, traditional cross-validation methods like random shuffling or standard k-fold validation cannot be applied because time series data is inherently sequential. When forecasting time series, we only predict future values, not past ones.

Consider the example in Figure 1.1a, specifically split 3. If we train using this slice and test on the middle portion, we encounter a critical problem of **data leakage**. This occurs because the model effectively "sees the future" during training, which biases the results. The model learns patterns from data that chronologically follows the test set, leading to unrealistic performance estimates.



(a) Typical K-Fold Cross-Validation



(b) Time Series Rolling Window Validation

FIGURE 1.1 – Comparison of cross-validation approaches

The solution to this problem is **Rolling Cross-Validation** (also called time series cross-validation). As illustrated in Figure 1.1b, this method provides the best of both worlds :

- Maintains temporal ordering of observations
- Creates multiple training/validation splits
- Progressively expands the training window

In this approach, we typically :

- Reserve 20% of data for final testing
- Use 10% increments for validation
- Iteratively move the training window forward
- Evaluate performance at each step

This ensures our evaluation properly simulates real-world forecasting conditions where we only use historical data to predict future values.

And we can simply does this using the library sklearn :

```
from sklearn.model_selection import TimeSeriesSplit
```

3 Data Exploration And Preprocessing

3.1 EDA

In this part we will be primarily using the library **Seaborn** as **sns**.

The **time series graph** of our dataframe :

```
sns.lineplot(data=df, x="Year-Month", y="CO2 Emission", linestyle="-")
```

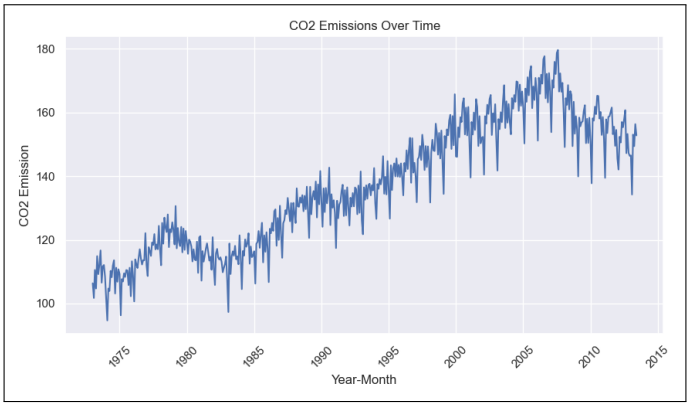


FIGURE 1.2 – Time series Graph

3.1.1 Distribution of CO_2 Emissions : Bar Plot and Histogram

In this section, we aim to analyze how CO_2 emissions vary across months and understand the overall distribution of emissions.

Bar Plot : First, we visualize the total CO_2 emissions emitted each month over the years :

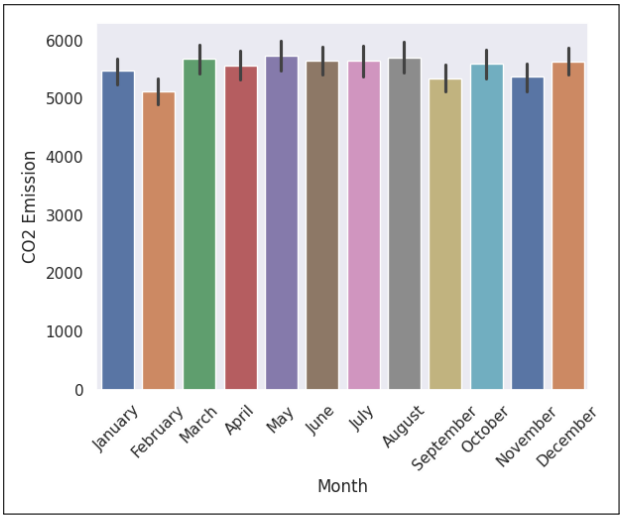
```
sns.barplot(data=df, x='Month', y='CO2 Emission', estimator=np.sum)
```

As we can see, the CO_2 emissions across the months are fairly uniformly distributed.

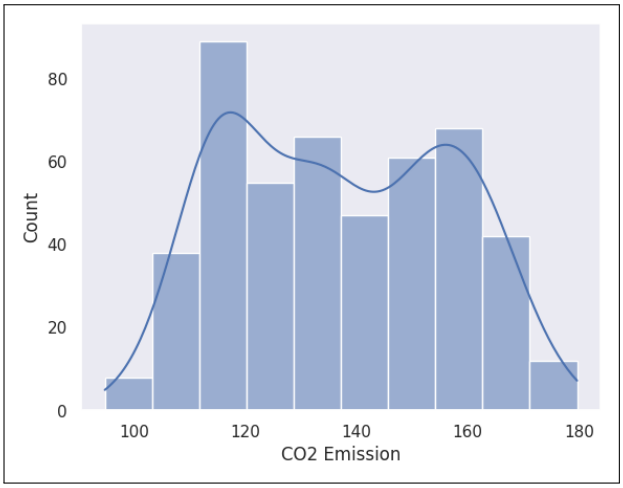
Histogram and Kernel Density Estimation (KDE) : Next, we examine the distribution of CO_2 emissions using a histogram combined with a Kernel Density Estimation (KDE) :

```
sns.histplot(data=df, x='CO2 Emission', kde=True)
```

We notice that the distribution is slightly skewed but remains close to a uniform distribution, consistent with our observation from the bar plot. This suggests that when applying scalers in models like **LSTMs** and **Transformers**, it is appropriate to use a **MinMax Scaler**.



(a) Bar plot



(b) Histogram and KDE

FIGURE 1.3 – (a) Bar plot and (b) Histogram with KDE

3.2 Preprocessing

In the DataFrame `Emission.csv`, we used the following Python code :

```
df.isna().sum()
```

This line checks if there are any missing values in the DataFrame. After executing the code, we found that there were **0 missing values**.

3.2.1 Outliers

We are going to use **Boxplot** to see if there's any **Outliers** :
From this figure 1.4 we can tell that there is no outliers so the preprocessing part is done.

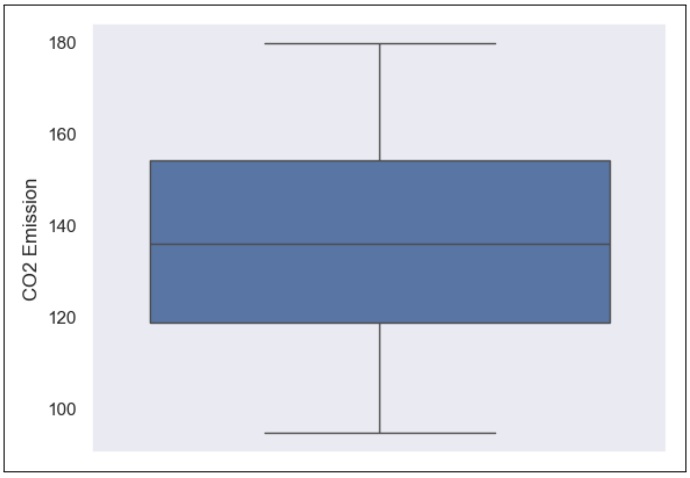


FIGURE 1.4 – Boxplot DB1

4 Application Of Classical Time Series Models(ARIMA/SARIMA)

In this part we can apply ARIMA/SARIMA,LSTMss and Transfromes and we compare results at the end.

4.1 Fondamental of Classical Time Series models

ARIMA (AutoRegressive Integrated Moving Average) and SARIMA (Seasonal ARIMA)(for more details check the section 1) are among the most widely used classical statistical models for time series forecasting. Despite their simplicity, they remain highly efficient and continue to be relevant in modern applications.

The basic foundation of classical time series analysis is that any time series can be decomposed into three components :

- **Trend** - Long-term progression (upward/downward movement)
- **Seasonality** - Regular periodic fluctuations
- **Noise** - Random irregular variations

4.1.1 Classical Model Types

In classical time series, there are two fundamental decomposition models :

- **Additive Model** :
$$y_t = T_t + S_t + \epsilon_t \tag{1.1}$$
where T_t is trend, S_t is seasonality, and ϵ_t is noise.
- **Multiplicative Model** :
$$y_t = T_t \times S_t \times \epsilon_t \tag{1.2}$$

4.1.2 Model Identification

We can distinguish between these models using various methods. In this report, we focus on :

- **Graphical Method** :
  - Plot the seasonal component across periods
  - If seasonal patterns are *parallel* → Additive model
  - If seasonal patterns *expand/contract* → Multiplicative model
- For multiplicative patterns, apply **Box-Cox transformation** to convert to additive.(for more detail check 1.1)

ARIMA/SARIMA models are linear model so we have to work with **additive model** and to ensure we don't need to use logarithmic transformation(**Box cox**) we are going to use **Graphical methode** :

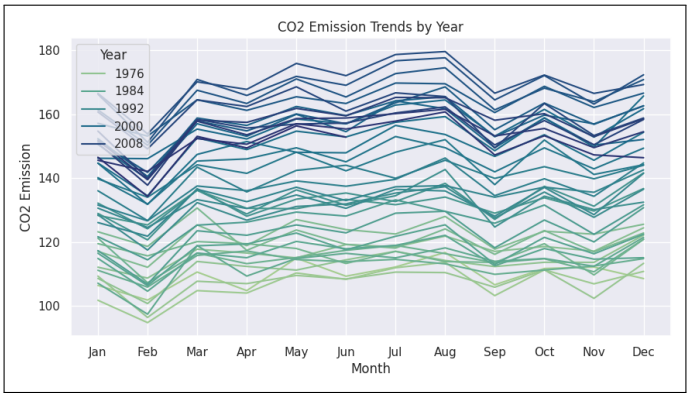


FIGURE 1.5 – Graphical methode to check the model of the Classical time series decomposition

As shown in Figure 1.5, we can observe a parallel between the seasonal components, so the model is **Additive**.

4.1.3 Analyzing the trend

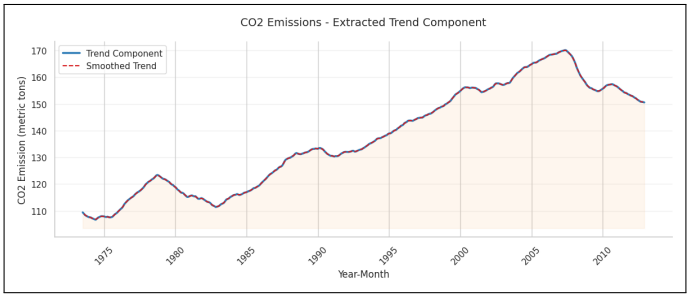


FIGURE 1.6 – trend of the time series

Important Observation

As evidenced in Figure 1.6, the apparent non-linear trend structure suggests that conventional linear time series models (ARIMA/SARIMA) fail to capture non linear relation ship such as the one we have here so our model will struggle.

4.1.4 Differencing the Time Series

Differencing transforms a time series to make it **stationary** - meaning it has a constant mean and variance over time, with consistent patterns. This is crucial because ARIMA/SARIMA models fundamentally require stationary data. When we difference (e.g., calculate $y_t - y_{t-1}$), we remove trends and stabilize the variance, allowing these models to work properly.

And we can check if our time series is stationary using the Ducky-fuller test :

```
from statsmodels.tsa.stattools import adfuller
```

And it was stationary after one differencing d=1

4.2 ARIMA

4.2.1 Tuning parameters

In this part we are going to see three methods, Graphical method using ACF/PACF ,Grid Search and rolling cross-validation

1-Graphical Method ACF/PACF

- Choose the **p** parameter by looking at the **PACF** plot :
  - Pick the lag where the PACF values drop to zero or fall within the blue confidence area
- Choose the **q** parameter by looking at the **ACF** plot :
  - Pick the lag where the ACF values drop to zero or fall within the blue confidence area

Note : This method should only be applied after making the time series stationary (if required).

We can see from the figure 1.7 both **ACF** and **PACF** goes to 0 at **lag 3** so the parameters **p=3 and q=3 ARIMA(3,1,3)**

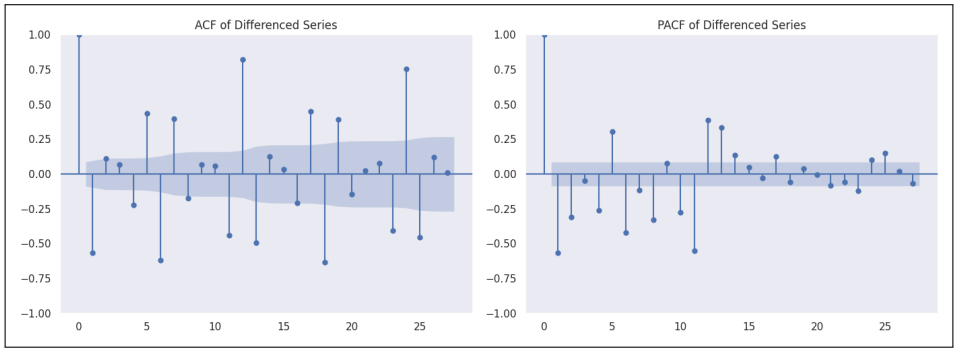


FIGURE 1.7 – ACF and PACF

2-Rolling Cross-Validation

Here we tried multiples parameters in different **Validation sets**.
We found the **ARIMA(3,1,3) as one of top 5**

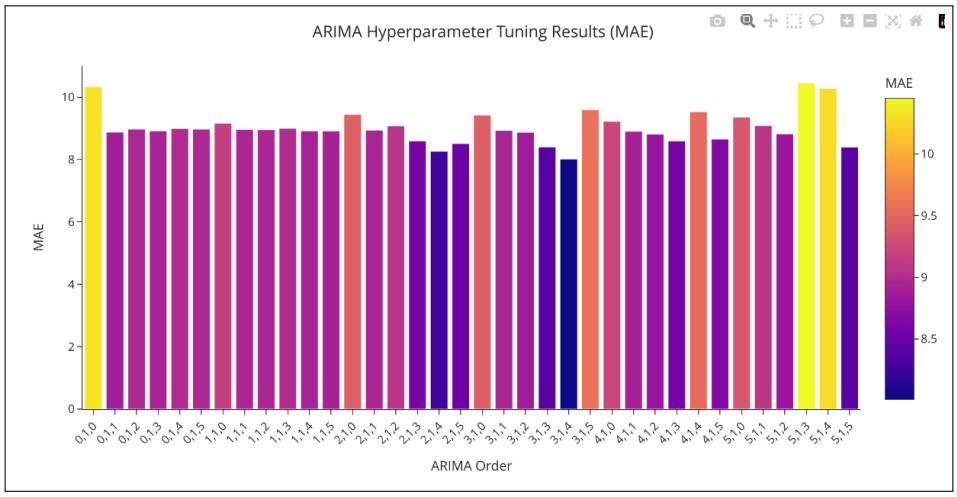


FIGURE 1.8 – Rolling Cross-validation

3-Grid Search

We performed a grid search over the space of $p \in [0, 5]$ and $q \in [0, 5]$ based only on the **AIC**.
And we found that best parameters are **ARIMA(2,1,4)**

Comparison between different models

As you can see in this figure 1.9 the **ARIMA** model didn't performer well.

	Model	MSE	RMSE	MAE	R^2	AIC
0	Grid search ARIMA(2,1,4)	34.5642	5.8791	4.9793	0.1680	2253.015649
1	Rolling cross validation ARIMA(3,1,4)	36.5681	6.0472	5.2245	0.1197	2290.537053
2	Graphical Method ARIMA(3,1,3)	34.5960	5.8818	4.9358	0.1672	2269.781934

FIGURE 1.9 – Comparison of different arima models

4.2.2 Forecasting With ARIMA(3,1,3)

We choose the model **ARIMA(3,1,3)** he both the best in **rolling Cross validation** and in **Graphical method**, here a plot for different step size :

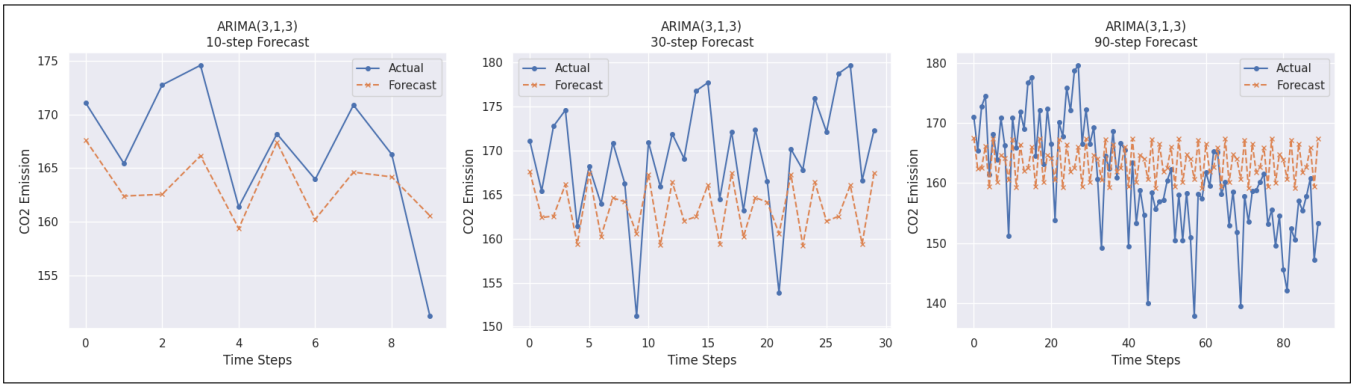


FIGURE 1.10 – Comparison of ARIMA model Forecasting with different steps

As we can see in this figure 1.10 the the longer the steps the worst the performance because **ARIMA model can't capture long dependencies and non linear relationships as we saw in the trend** 1.6.

4.2.3 Metrics for ARIMA(3,1,3)

Horizon (steps)	MSE	RMSE	MAE	R^2
10	20.3626	4.5125	3.0651	0.4948
30	33.4780	5.7860	4.4007	0.2085
90	116.6336	10.7997	8.8339	-0.3887

FIGURE 1.11 – Metrics of ARIMA Model

As we can see in the figure 1.11 that the model do medium in the 10 and 30 kinda bad in 30 and worst in 90 steps.

4.3 SARIMA

The Seasonal ARIMA (SARIMA) model with non-seasonal order $(p, d, q) = (3, 1, 3)$(perviously found) and seasonal order $(P, D, Q)_m = (?, ?, ?)_{12}$ that we will find.

4.3.1 Tuning parameters

in this part we will do one method as the two we saw them before.

Rolling Cross-Validation

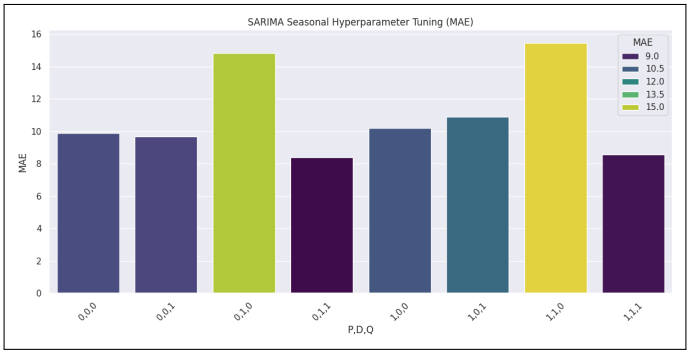


FIGURE 1.12 – Rolling Cross-Validation SARIMA

As we can see from the figure 1.12 we have mutiple options to choose from but we will choose the model yearly season :

$$\text{SARIMA}(3,1,3)(1,1,1)_{12}$$

4.3.2 Forecasting With SARIMA

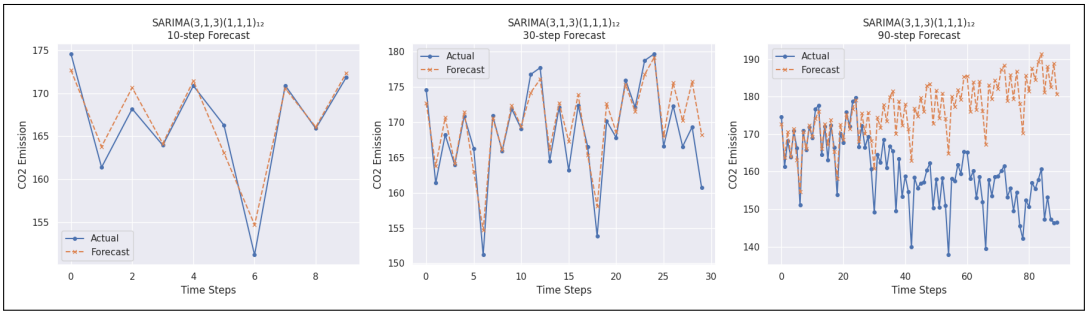


FIGURE 1.13 – Forecasting with SARIMA(3,1,3)(1,1,1,12)

asdasshhdfhehsaacjw

4.3.3 Comparing SARIMA Metrics With ARIMA Metrics

Horizon (steps)	MSE	RMSE	MAE	R^2
10	3.9119	1.9779	1.5342	0.9029
30	7.5280	2.7437	2.0823	0.8220
90	410.8616	20.2697	16.5345	-3.8918

(a) Metrics of SARIMA(3,1,3)$(1,1,1)_{12}$ model across different forecasting horizons

Horizon (steps)	MSE	RMSE	MAE	R^2
10	20.3626	4.5125	3.0651	0.4948
30	33.4780	5.7860	4.4007	0.2085
90	116.6336	10.7997	8.8339	-0.3887

(b) Comparison of ARIMA and SARIMA model performance

FIGURE 1.14 – Evaluation of time series forecasting models.

as we can see in the figure above the **SARIMA** outperformed the **ARIMA** model, And SARIMA did really well scoring $R^2 = 0.9$ and $R^2 = 0.82$ in 10 and 30 step.

4.3.4 Conclusion :

As we saw from the metrics of SARIMA and ARIMA, they can perform well in short-term forecasting. However, they struggle in long-term forecasting because they lack complexity. Since they are linear models, they cannot capture nonlinear relationships like we saw before in the trend 1.6.

5 Application Of Modern Time Series Models(LSTMs/Transformer)

In this part we are going to implement **LSTMs** and **Transformer** and see their advantages and disadvantages. And in this part we will be mainly using **TensorFlow**.

5.1 LSTMs

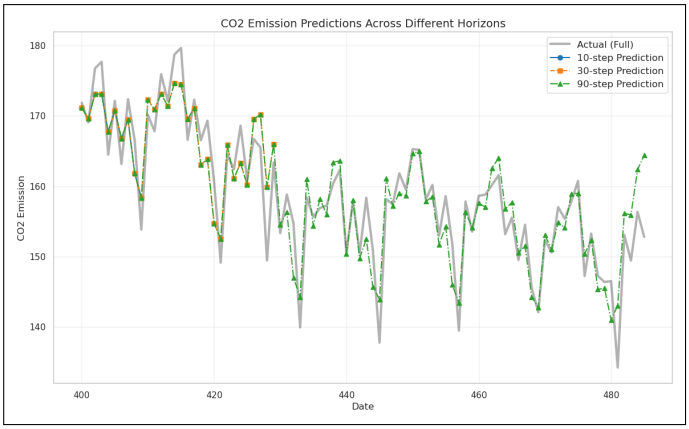
LSTMs is abbreviation for Long Short Term Memory and it's build on top of **RNN**(Recurrent Neural Network), **LSTMs** they are basically RNN but they have the ability to capture long term relationships and they solve the problem of **RNN** (Exploding and Vanishing of Gradient).

5.1.1 Training LSTMs

- **Data Preparation :** Decomposed the CO_2 series into trend, seasonality, and residuals ; engineered lag features to capture yearly patterns ; and scaled data to $[0, 1]$ using MinMax scaling.
- **Model Setup :** Built a deep Bidirectional LSTM with dropout regularization, using sequences of 24 months.
- **Training :** Trained with Adam optimizer, early stopping, and learning rate scheduling to ensure stable convergence.

5.1.2 Plotting And Metrics

We are going to plot LSTM in different steps :



(a) LSTMs Forecasting Vs Actual Data

	Steps	MSE	RMSE	MAE	R^2
0	10	11.1427	3.3381	2.9828	0.7548
1	30	15.1865	3.8970	3.3217	0.7287
2	90	13.4803	3.6715	2.8654	0.8475

(b) Metrics of LSTMs

FIGURE 1.15 – LSTM Model Results

As we can see from the figure 1.15 the LSTM model did pretty well but in short term Forecasting **SARIMA**1.14a did so much better than **LSTMs**, but in long term forecasting **LSTMs** wins and **SARIMA**1.14a performed terribly.

5.2 Transformers

Transformers for time series are built on top of LSTMs, and they typically require a large amount of data. However, in our case, we only have around 500 data points, which isn't enough. As a result, this model is not effective, as demonstrated below :

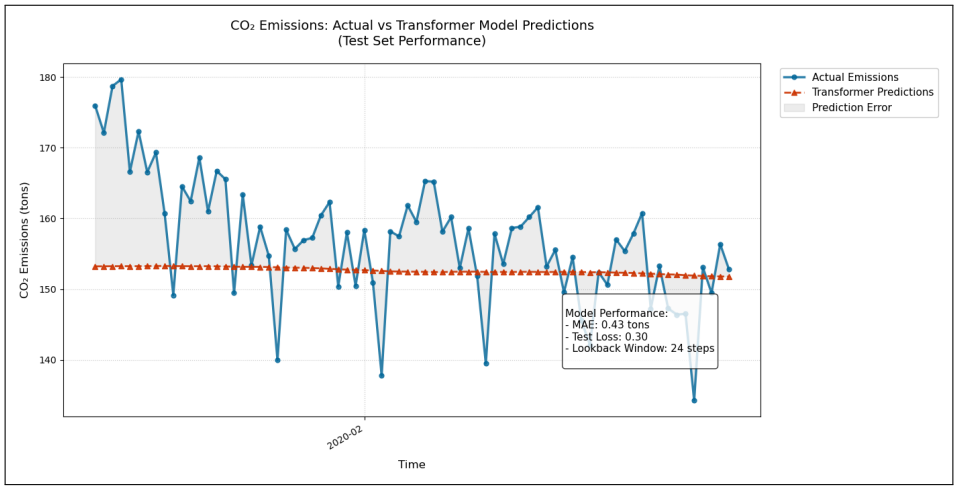
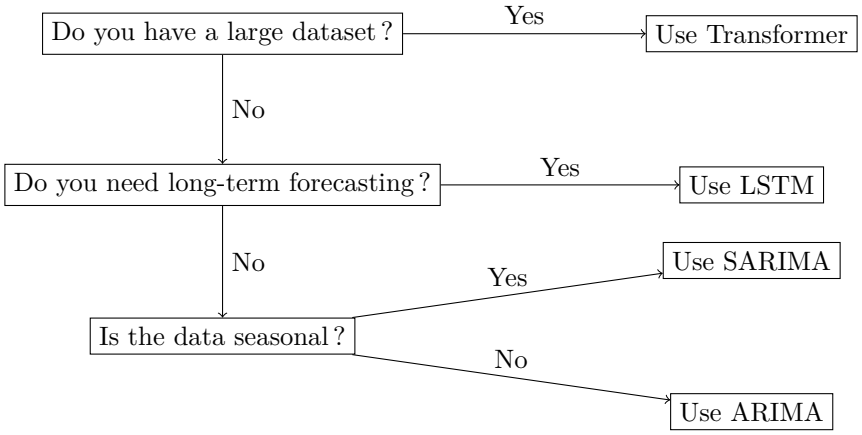


FIGURE 1.16 – Transformers Forecasting

6 Conclusion

In conclusion, simple models like **ARIMA/SARIMA** work really well when we have a small dataset and need quick, **short-term forecasting**. If we want **long-term forecasting** and have a small dataset, **LSTMs** are the go-to choice. For large datasets, **Transformers** take the lead.

We will show you in the graph below when to use each model in order to take advantage of their strengths :


Do you have a large dataset ?
Yes
Use Transformer
No
Do you need long-term forecasting ?
Yes
Use LSTM
No
Use SARIMA
Yes
Is the data seasonal ?
No
Use ARIMA

Chapitre 2

Annex(Math part no need to read it ! ! !)

1 ARIMA/SARIMA

ARIMA stands for :

- **AutoRegressive (AR)** : Uses past values to predict future ones.
- **Integrated (I)** : Makes the series stationary by differencing.
- **Moving Average (MA)** : Uses past errors to improve predictions.

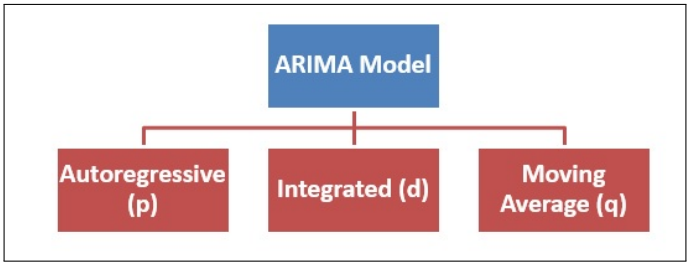


FIGURE 2.1 – ARIMA

Differencing (Integration) To make the time series stationary, we take the first difference :

$$Y_t' = Y_t - Y_{t-1} \tag{2.1}$$

For ARIMA(1,1,1), the model is :

$$Y_t' = \phi_1 Y_{t-1}' + \theta_1 \epsilon_{t-1} + \epsilon_t \tag{2.2}$$

SARIMA extends ARIMA by adding seasonal components :

- **Seasonal AR (SAR)** : Seasonal autoregressive terms
- **Seasonal MA (SMA)** : Seasonal moving average terms
- **Seasonal Differencing** : Removes seasonal patterns

For SARIMA$(1,1,1)(1,1,1)_s$, the model is :

$$\Phi(B^s)\phi(B)\nabla_s^D \nabla^d Y_t = \Theta(B^s)\theta(B)\epsilon_t \tag{2.3}$$

where s is the seasonal period.

1.1 Box-Cox Transformation

- Converts multiplicative models to additive form :

$$y_t^{(\lambda)} = \begin{cases} \frac{y_t^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y_t) & \text{if } \lambda = 0 \end{cases} \tag{2.4}$$

- Key properties :
  - $\lambda = 1$: No transformation needed (already additive)
  - $\lambda = 0$: Logarithmic transformation
  - $\lambda = 0.5$: Square root transformation
- Optimization : λ is typically found via maximum likelihood :

$$\lambda^* = \arg\max_\lambda \mathcal{L}(\lambda; \mathbf{y}) \tag{2.5}$$

  where $\mathcal{L}$ is the profile likelihood function.
- After transformation, the model becomes additive :

$$y_t^{(\lambda)} \approx T_t^{(\lambda)} + S_t^{(\lambda)} + \epsilon_t^{(\lambda)} \tag{2.6}$$