Time series analysis to predict the evolutionary pattern of coal power production in a renewable energy integrated electric power grid at the electricity market in ERCOT region of Texas.

Kushal Bhalla

Introduction

With enormous growth in human population, the demand for electric power has increased significantly. Countries all over the world produce electric power from either fossil (non-renewable) or non-fossil (renewable/regenerative) sources. Over the years, the US government has taken vital steps to increase the share of sources with either zero or as minimal carbon footprint as possible. Consequently, driven by government incentives and inducements like investment tax credits, subsidies, production tax credits, renewable portfolio standards, and renewable energy credits, different non-fossil sources like wind, solar, biomass, geothermal, and landfill have become an integral part of the electric grid in different regions and states within the US. In the US state of Texas, huge wind farms had come up over the past decade or so. The wind power penetration into the electric grid had increased from 5% in 2012 to 25% in 2023. This growth is phenomenal. Not only wind, investments had come up in other renewable sources like solar and biomass. In this scenario, it becomes essential to gauge, evaluate, or empirically estimate the pattern of displacement of the traditional fossil fuel sources or greater integration of renewable sources with the electric grid. So, this analysis will be restricted to the electric grid within the ERCOT region of Texas.

Moving ahead, we know that in the power production landscape or within the generation fuel mix, coal-based power generators are held most responsible for releasing maximum amount of carbon emissions (greenhouse gases) into the atmosphere, therefore, this investigative endeavor shall dwell primarily upon understanding or modelling the evolution and growth of the coal resource. More specifically, time series analysis tools and techniques would be used to understand, monitor, and forecast the future growth of coal resource, and its pattern and speed of replacement, substitution, and integration with the electric grid. This information will be useful in planning an optimal and timely allocation among different sources and ensuring a cost-effective production of power from a diversified portfolio of different sources comprising of a mix of both renewable as well as nonrenewable sources. Time series analysis can help explore both the short-term trends (hourly patterns) as well as the long-term patterns (daily, weekly, and annual seasonal patterns and cycles) of power production from fossil and non-fossil sources. Based upon this premise, a variant of recurrent neural network technique called the Long Short-Term Memory (LSTM) neural network model will be used to capture the long-term time/temporal dependencies in the time series of coal, using gas, combined cycle gas, renewable, and nuclear as a set of external predictors.

Background and Related Work

Earlier, in different research scenarios, cumulative efforts had been made to examine, predict, and forecast the electric demand in different regions and countries. Different time series models starting right from exponential smoothing, moving averages, holt-winters method, distributed lag models, box Jenkins models, and moving ahead to unobserved components models, state space models, and neural networks models had been used to explore the trends, patterns, components, and latent/underlying structure of different series. Cross relationships and interdependencies of demand have been studied and evaluated with prices, alternative sources, futures, and forwards contract (financial instruments), financial derivatives (options), and power purchase agreements (long term contracts). Optimal economic dispatch models (linear/nonlinear programming models) provide cost effective solutions of producing power from a given source given the cost and output from other sources, system constraints, carbon tax, renewable energy credits, and other subsidies and tax incentives. Little work has been done as such which directly forecasts or predicts the power production from a fossil source within an electricity market given the information about other competing sources and using a time series analysis technique.

Data procurement and preparation (source: https://www.ercot.com/gridinfo/generation) / Methodology

In our analysis coal was the response variable or the dependent variable. The set of predictors included Gas, Gas-CC, Renewable, Nuclear. Original data had observations at a 15-minute interval of power produced from different sources. From this hourly data was created by data manipulation. Final data for analysis had 150192 records or observations ranging from 2012 to 2023. Values of wind, solar, and biomass were combined to create a new column named renewable (Ren). Columns for hydro and other sources were excluded since their numbers were too small and showed negligible correlation with the coal variable. Additionally, sets of sine and cosine functions were used to generate seasonal and periodic patterns based upon the datetime object. The data was normalized using the min max scaler for effective handling of numerical values.

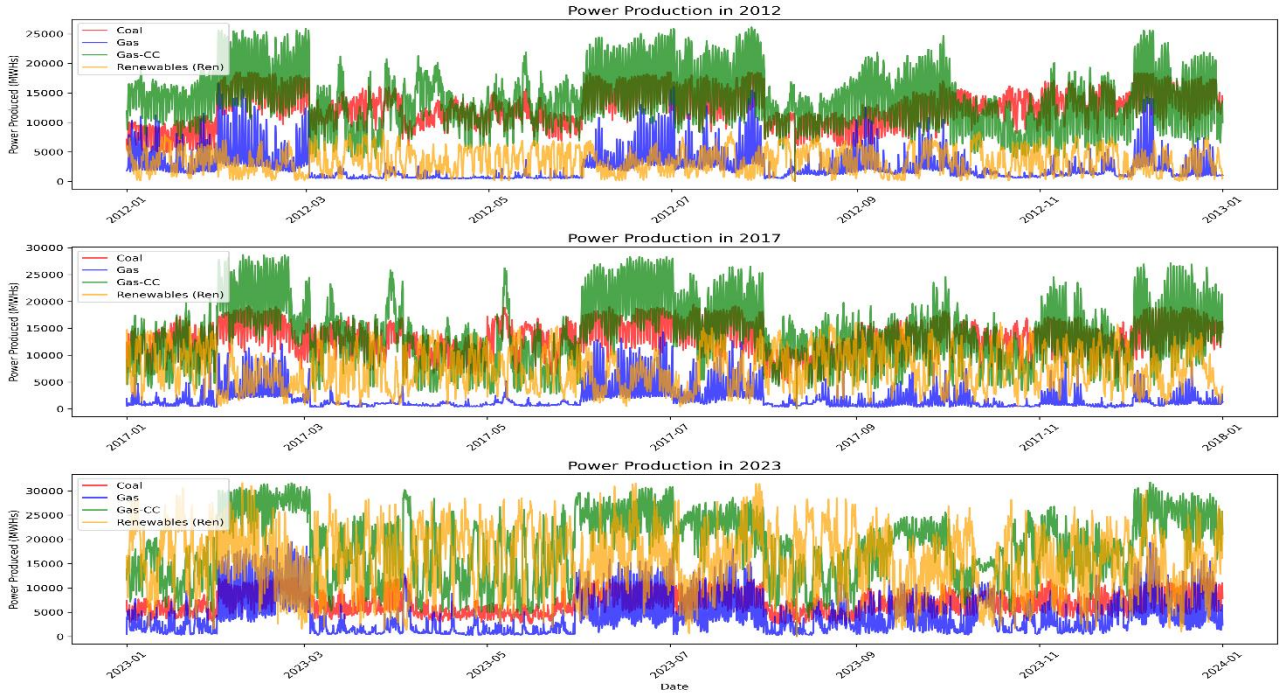
To generate reliable forecasts for the coal resource an LSTM neural network model was used. The long short term memory model (variant of the recurrent neural network model) was well equipped to handle inherently sequential data, long-term time/temporal dependencies, and recurrent connections. This model used layers and neurons to explore the hidden and deeply embedded patterns in the data relying upon techniques like back propagation through time. Each neuron learns about a feature or a pattern. With more neurons the model could learn more complex and intricate patterns. With more layers the network would have more depth. Information across the time steps was preserved, discarded (forgotten/erased), added, updated in the memory, and based upon this

information an output was produced. Consequently, both the short-term trends and the long-term patterns (prior sequential patterns/seasonal patterns with different periodicities) were adequately identified, learned, and captured for further forecasting.

Another interesting aspect of LSTM is, it is capable of handling nonlinear relationships of the response variable with other predictors. Batch size was adjusted such that both computational efficiency and convergence were assured. The select number of epochs assured model convergence and balance of training time and performance. Further, time steps = 24 was used for sequence creation by splitting the data. A sequential fivefold cross validation was used.

Data visualization

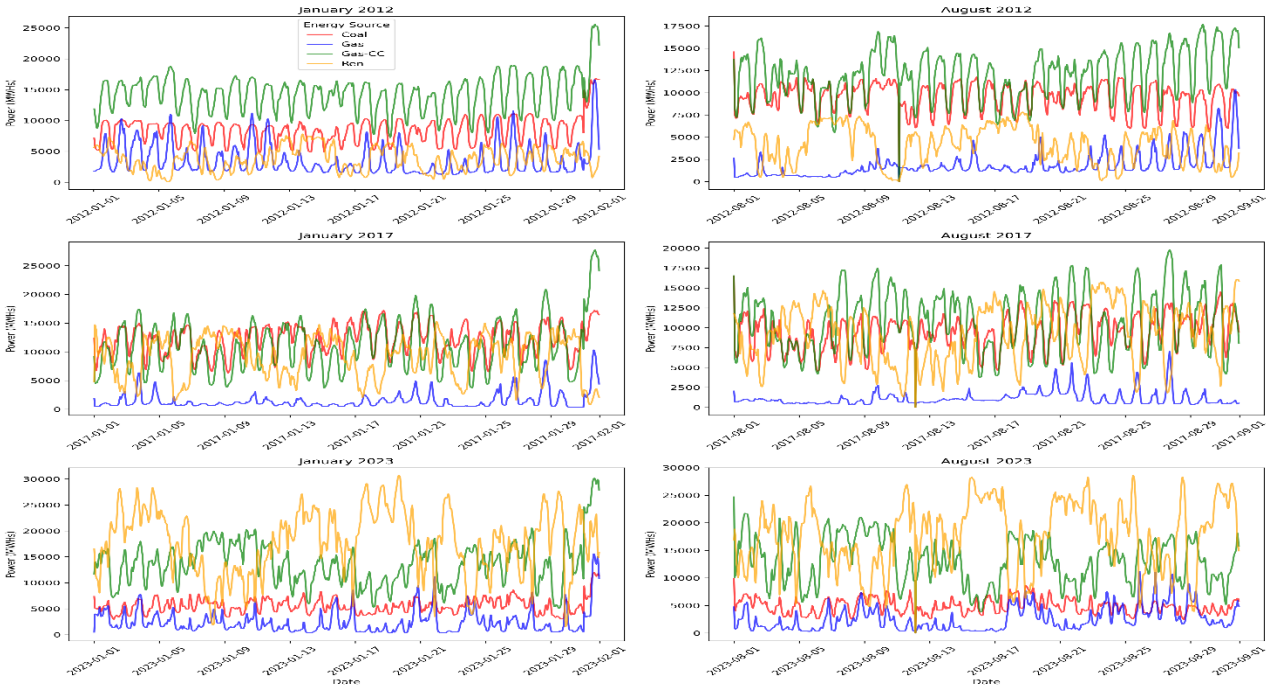
Fig 1.



From the total range of hourly data which spanned across 12 years from 2012 – 2023, three specific years (2012, 2017, 2023) were selected to graphically examine the evolution of coal, gas, gas-cc, and renewables. It is quite apparent from Fig 1. that over the decade, the renewables have challenged the production dominance of gas and coal. In fact, the renewables have replaced and substituted the fossil fuel sources on many occasions. It is the relegation of coal to a much lower level of production which should have been quite instrumental in bringing about a decline in the greenhouse gas and carbon emissions over the years. The coal which had been hovering above 10000 MWHs in 2012, even crossing the 15000 level during some months, can be seen swaying around 5000 in 2023. The hourly production of power

from gas-cc has also shown a decline. Especially, the periods when there is more renewable energy penetration, the production from gas-cc is showing a remarkable decline. However, is does bear an explanation, as to why is it that gas-cc which had a peak production of 25000 MWHs in 2012 can be seen touching a high of 30000 MWHs in 2023. This phenomenon could be explained by the intermittency of wind, especially since gas-cc is the most economical and reliable substitute for the renewables, therefore, certain spinning reserves of gas-cc must be maintained.

Fig 2.



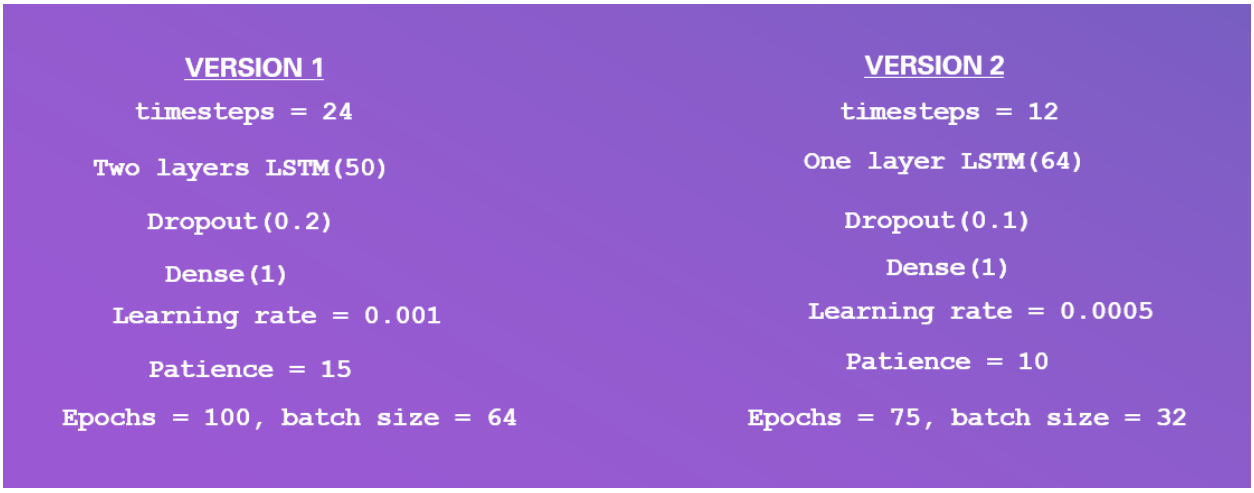
In Fig 2. the seasonal and periodic pattern of coal, gas, gas-cc, and renewables can be seen evolving and transforming for the months of January and August for the years 2012, 2017, and 2023. This plot sheds light on how an enhanced renewable energy penetration or integration with the electric grid had affected the periodic pattern of production of fossil sources along with their peak levels. For the month of January, a shift can be noticed in the daily seasonality of gas from 2012 to 2023. Wind had so intensely and so strongly intruded the gas production cycle, bringing in additional cyclic pattern to the gas-cc and a change in its level. Wind has poignant uncertainties in its periods of peak production and low production during different hours of the day and that had rendered more fluctuations in the production extremes for the gas-cc. A significant drop in coal and gas are also visible. January is the winter month and there is more electric demand due to heating requirements and a greater wind penetration during that time is inducing more fluctuations in the gas-cc production pattern. The same interplay between wind and gas-cc can be seen for the month of August starting from 2012. The variations are quite stark for August as in January since in August there is extreme heat due to summer season and power consumption significantly shoots up due to cooling requirements. And

during this time due to more wind penetration which has its own intermittency and unpredictability, the production pattern of gas-cc undergoes quite a drastic change, where the periodicity of the seasonal pattern is disturbed quite a bit. The pattern is a lot more irregular and the oscillations are wider and more intense. Not to mention, the coal has undoubtedly, registered a significant decline in its production during both January as well as August from 2012 to 2023. Coal which hovered periodically above 10000 MWHs in 2012 can be staying below 5000 MWHs in 2023.

Experiment Design and Results

First the LSTM model was run with the hyperparameter specification which allowed for timesteps = 24, two layers with 50 neurons, dropout = 0.2, learning rate = 0.001, patience = 15, epochs = 100, and the batch size = 64. However, since with this configuration there was a high RMSE, therefore, the parameters were changed in the second version. In the second version, there were 12-time steps and one layer of LSTM with 64 neurons. Here it can be noticed that the time steps and layers were reduced to half in comparison to version 1, although the number of neurons were increased to 64. Further, the dropout rate was set to 0.1, and learning rate was reduced to 0.0005. The patience was brought down to 10 and the number of epochs were changed to 75. Use of 75 epochs reflected a smaller number of iterations across the training data. Finally, the batch size was also reduced to half in comparison to the version 1, from 64 to 32. Now all these changes were introduced with the primary goal of reducing the RMSE, or in other words generating more reliable and accurate forecasts. The chart below shows the choice of hyperparameters for both the versions as discussed above.

Fig 3.

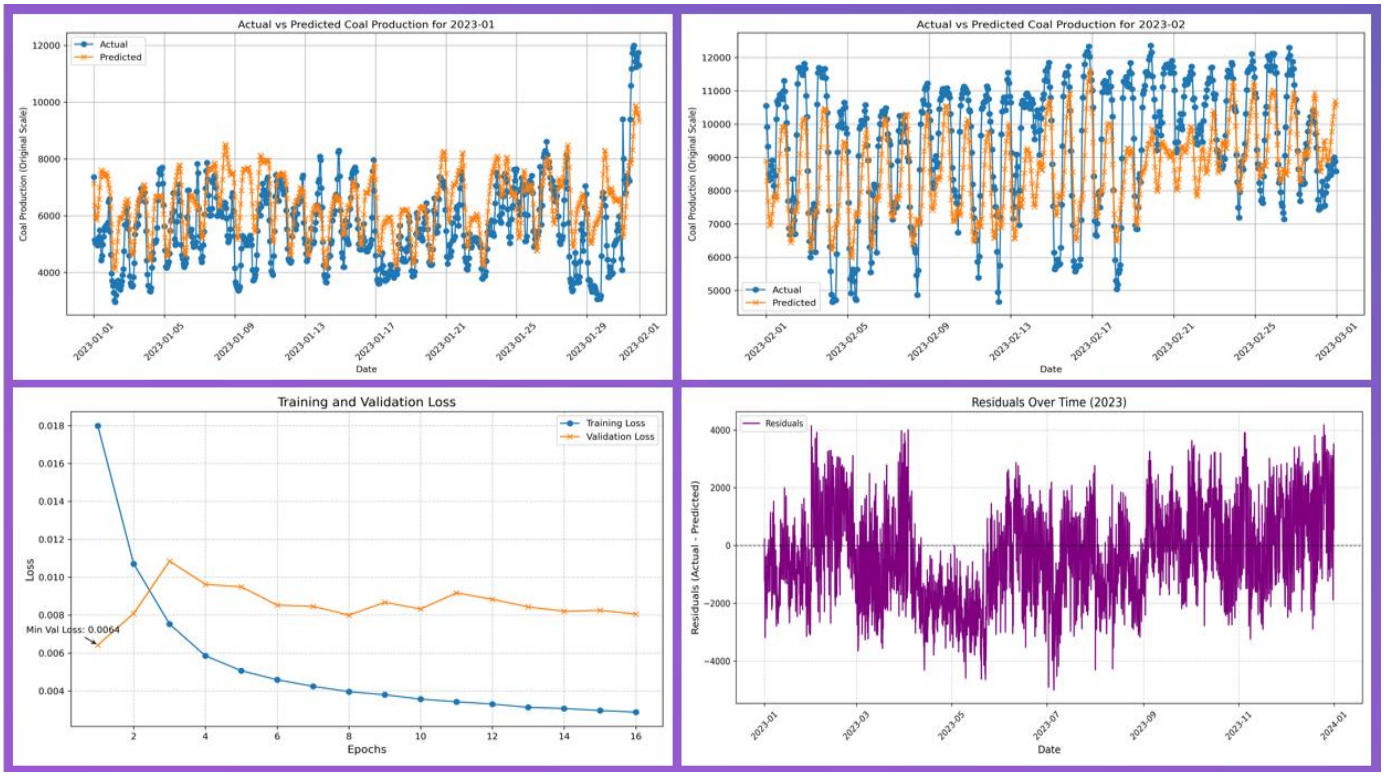


The hyperparameter tuning should be such, the model can be successfully generalized. The model should not memorize or learn too much from the training

data. The model should be able to perform well when it comes across unseen or unknown data. To that end, by reducing the number or epochs and the number of layers, the overfitting if any can be taken care of. In case of epochs, early stopping is helpful as it will stop the iteration when the validation loss stops improving. One epoch is one complete pass through the entire training set. 75 epochs would mean passing the entire data set through the model 75 times. In our case, decreasing the batch size from 64 to 32 is a step towards regularization. A slow learning rate is desirable for smooth model convergence and prevent overfitting. In our case the learning rate was dropped from 0.001 to 0.0005. The dropout rate was decreased from 0.2 to 0.1 which will ascertain, the model relies more on specific neurons. Additionally, in our case the batch size was dropped from 64 to 32 which would lead to better generalizations. Batch size is equivalent to the number of training samples used before the weights are updated. Finally, the time steps had been reduced from 24 to 12. This translates to the fact that as compared to version 1 where the past 24 hours (historical information) were used, in version 2 we used only the past 12 hours. In other words, the model is learning from 12 time points in each iteration. This drop in time steps will reduce the computation cost.

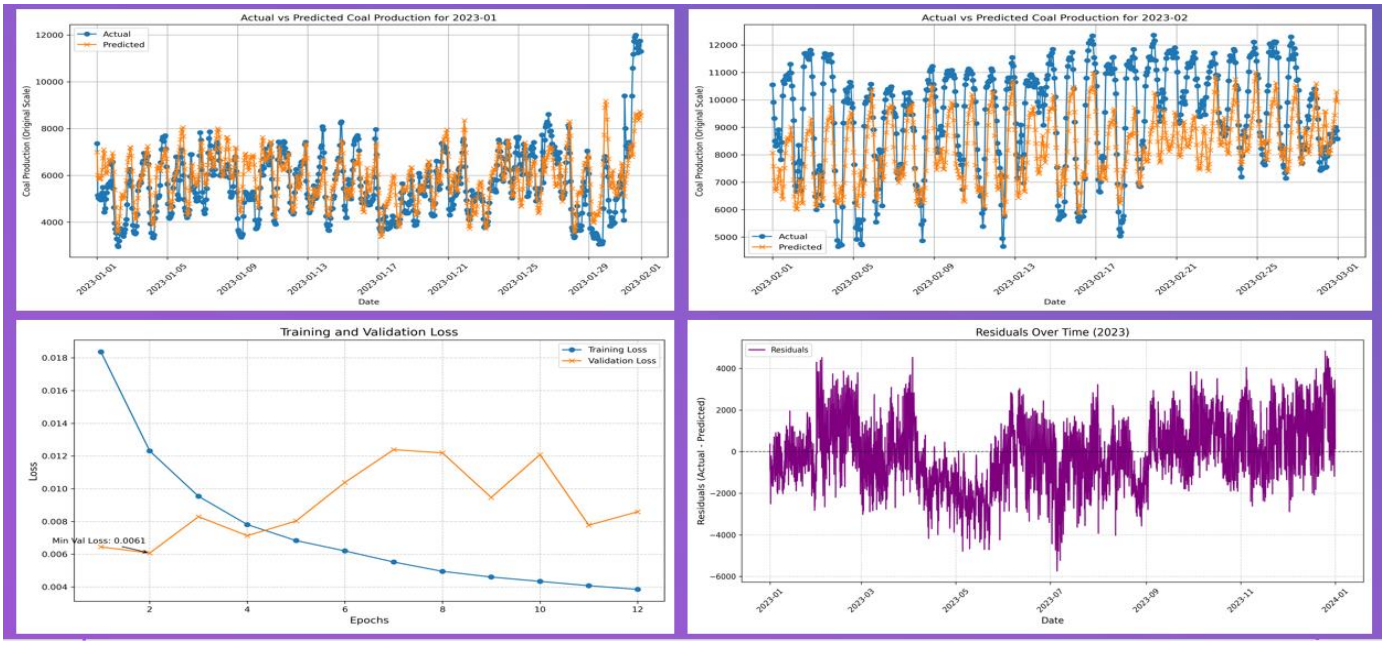
Presented next are the forecast results, training and validation loss plots, and residual plots as obtained from a successful run of LSTM version 1 and version 2.

Fig 4. Forecast/actual plots for January, February of 2023, LSTM loss function plot, and residual plot (version 1 of LSTM)



From figure 4, it can be deciphered, the LSTM model generated forecasts had been able to follow the pattern of power production of the actual values of coal resource. But there is still tremendous scope for improvement. In many instances the predicted values could not imitate the extremes in power production during the months of January and February of 2023. Even the validation loss plot appears to have started plateauing/stabilizing a lot earlier even when the training loss was still decreasing or declining. In an ideal plot for training and validation loss, both training loss and validation loss must decrease as the number of epochs increase. Moreover, there ought to be some convergence between the training loss and validation loss when it comes to minimizing the difference in their measure of root mean square error. In the residual plot, there are instances when the errors/residuals do not appear to be spread evenly across the 0 mark which is indicative of some systematic tendencies still present in the data which could not be captured by the model.

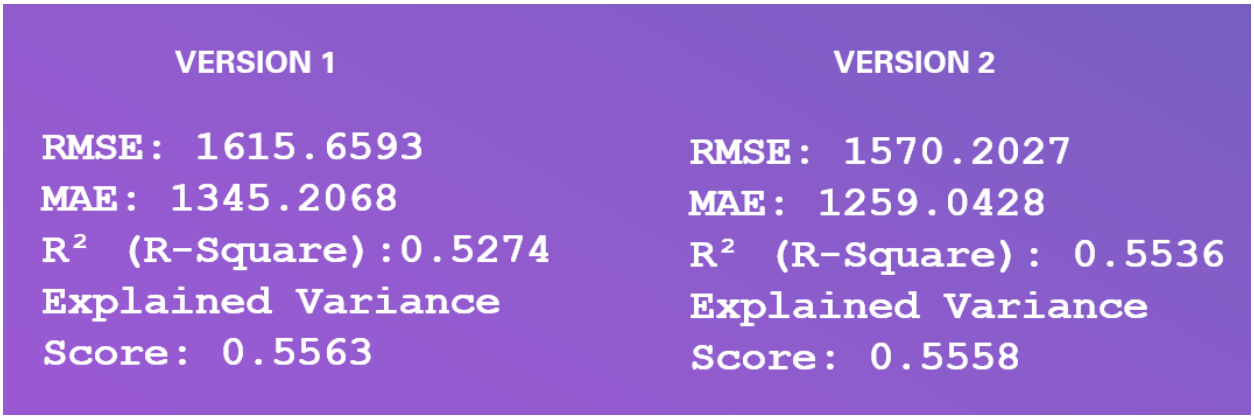
Fig 5. Forecast/actual plots for January, February of 2023, LSTM loss function plot, and residual plot (version 2 of LSTM)



The forecast/actual plot for January and February 2023 using LSTM version 2, depicts a slight improvement in terms of proximity/overlap of the actual and predicted values. The model has done a much better job for January by more accurately capturing the evolutionary pattern for the winter month. However, for February there is still more scope for betterment, as is apparent that the predicted values could not completely follow the extremes of power production. The loss function plot indicates more conformance for version 2 being a better model in

comparison to the version 1. Both training loss and validation loss are declining and during that decline the separation between the validation loss and training loss is narrowing or squeezing which reflects better convergence. The overfitting which we noticed in version 1 seems to have been addressed up to some level. There is still scope for enhancement in terms of obtaining better predictions by altering the hyperparameter values. Some systematic tendencies still appear apparent in the residua plot with residuals not evenly spread across the 0 mark.

Fig 6.



From the final training and testing run of the two versions of LSTM and as evident by the results of model evaluation metrics, it emerges quite clearly that version 2 of LSTM was more apt and efficient in terms of coming closer to capturing the evolutionary pattern of coal resource. This is indicated by a smaller RMSE of 1570 and a smaller MAE of 1259 in case of version 2 as compared to an RMSE of 1615 and an MAE of 1345 for version 1. Even the R square value for version 2 (0.55) is higher in comparison to version 1 (0.52).

Summary and Future Work

LSTM model was selected for this analysis based upon a set of reasons. Firstly, this model can accommodate multiple predictor variables and excel in handling non linear relationships of the response variable with the predictors. Secondly, unlike the box Jenkins models where stationarity is essential to make sure that the process is covariance stationary stochastic, the LSTM model does not require this precondition. Thirdly, the LSTM implicitly handles the periodicities like daily, weekly, and other seasonal and cyclical patterns, meaning the time series components do not need to be modelled explicitly. The LSTM model generates forecasts based upon the learned patterns.

Two successful model runs with different hyperparameter specifications provided an opportunity to enhance the model performance by controlling any underlying overfitting/underfitting. By altering the parameters, a better version of the LSTM model could be used to train and test the data for obtaining better and more

reliable forecasts of the coal resource. The RMSE, MAE could be reduced and even training and validation loss could be controlled.

Even a baseline hypothesis can be tested to detect any statistically significant difference in the RMSE values for version 1 and version 2. This will indicate whether the new parameter specifications were indeed helpful in improving the predicting power of the LSTM model. This will rule out the hand of chance in arriving at a smaller RMSE value in version 2. Not only this, even a large language model can be used to determine the most optimal vector of hyperparameters for the LSTM model (meaning trying many more model/vector configurations in a machine learning environment/framework using an LLM) which will most accurately represent the evolutionary pattern of coal resource by generating predictions which will far better than predictions yielded by any other vector combination, based upon the required statistical significance/evidence. Since, there had been significant time and resource constraints in preparing this document and completing analysis in a timely manner (given the GPU requirement) therefore, the important task of doing this hypothesis testing with a baseline model and determining the most optimal model will be a work in continuation.

Moreover, in future work, apart from further improving the LSTM model, an unobserved components models (UCM) can also be trained to compare the accuracy of the two models in getting more enhanced and precise predictions/forecasts for either coal or both coal and gas in a multivariate setting.

Appendix:

ChatGPT prompts for the LSTM model

Prompt 1:

There is a 15-minute interval data set of 9 variables with more than 4 lakh records, and then an hourly data set with more than 1 lakh records and data for specific hours of the day. Kindly advise on running a neural network time series model on the hourly data set of 9 variables which spans from 2012 to 2023. Earlier I thought about using an LSTM neural network model to predict the response variable of coal along with other explanatory variables and dummy indicator variables. Further unobserved components models can also be used for the same where we model the trend, including the level and the slope and different stochastic seasonal and cyclic components along with explanatory variables. Now my question is, given that I must run LSTM neural network and unobserved components model on the coal response variable, and I have more than 1 lakh records, then will it make any difference if I do the analysis in google colab (free tier) or databricks community edition? I mean whether both will be the same in terms of speed and performance and model convergence. Isn't it? Please let me know.

Prompt 2:

I must do a fivefold cross validation for my hourly time series data set from 2012 to 2023 and which has 13 explanatory variables and one dependent variable using the LSTM neural network model as discussed previously. In my explanatory variables I have the independent predictors of Gas, Gas-CC, Ren, Hydro, Other, the 4 nuclear dummies, and the 4 extracted numeric temporal features. First question is that in case of LSTM do we have to drop one nuclear dummy/indicator variable to get rid of the dummy variable trap? Further, suppose that I also convert the data to a numpy array. And based upon your description and given that I have a daily seasonality and a weekly seasonality, I choose timesteps = 24 to include the daily seasonality. Now when it comes to creating the sliding windows and the train test split, here I need help in the form of syntax of how to go about this in google colab. Another thing is I want to try with all the data set i.e. the entire hourly data from 2012 to 2023 which makes up 105192 records or observations. But here I want to include the fivefold cross validation. Initially 2012 to 2017 will be the training dataset and 2018 will be the validation set. Next 2012 to 2018 will be the training dataset and 2019 will be the validation set. Next 2012 to 2019 will be the training dataset and 2020 will be the validation set. Next 2012 to 2020 will be the training data set and 2021 will be the validation set. Finally, 2012 to 2021 will be the training data set and 2022 will be the validation set. This should be the hourly time series data set fivefold cross validation process. Could you please provide me a template of the syntax or an algorithm which could accomplish this fivefold cross validation in a sequential manner for an LSTM neural network model? After the fivefold cross validation is done then the model will be run on the testing data with 2012 to 2022 as the training data and 2023 as the testing data set. Please provide me a complete syntax that elaborates how to set the input shape for the training data for the LSTM at each step of the validation and testing. Kindly ascertain that the train test split must be done keeping in view the temporal nature/dependence/structure/ordering of this hourly time series data set. We will need the sequential train test split. In the end also kindly provide the syntax for how to check the model accuracy or model evaluation on the test set and make predictions. How to calculate evaluation metrics like the MSE, RMSE, training loss and validation loss over epochs (loss function during training), and residual analysis (residual plot over time).

Prompt 3:

Can we have a smaller number of epochs and smaller batch size? I wonder why did we use 50 epochs and 32 batch size? It will take so much time. Its already five minutes and it is still epoch 9/50 of fold 1. This will take forever. Any way to reduce the time?

Prompt 4:

This fivefold cross validation will take how much time given that its already 3 minutes and I have only reached epoch 7/20 of fold 1?

Prompt 5:

I am using google colab for this fivefold cross validation. How will I determine whether a CPU is being used or a GPU?

Prompt 6:

I have changed it to T4 GPU in google colab, and which was the only option available. Earlier it was set to CPU. I had to stop the 5-fold cross validation.

Prompt 7:

I have ran the following in a separate cell. Meaning not in the one where 5 fold cross validation is running. Is it ok? from tensorflow.keras import mixed_precision

```
# Enable mixed precision training

mixed_precision.set_global_policy('mixed_float16')
```

Prompt 8:

I have to do the Sliding Windows with Sequential Splitting as a part of running the LSTM neural network model. As suggested the sin and cos variables have been incorporated as follows 'hour_sin', 'hour_cos', 'day_of_week_sin', 'day_of_week_cos', 'day_of_year_sin', 'day_of_year_cos' and minmaxscaler was applied to the following variables only 'Gas', 'Gas-CC', 'Ren', 'Nuclear', 'Hydro', 'Other', 'year' and date which was a datetime object was left as such.

Do you think this set up is fine for further analysis ? I will be using the following code for sliding windows with sequential splitting

```
# Setup for Sliding Windows with Sequential Splitting
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping


# Define explanatory variables (features) and target variable
features = df1[['Gas', 'Gas-CC', 'Ren', 'Nuclear', 'Hydro', 'Other', 'year',
           'hour_sin', 'hour_cos', 'day_of_week_sin', 'day_of_week_cos',
           'day_of_year_sin', 'day_of_year_cos']].values

target = df1['Coal'].values

timesteps = 24  # Number of time steps for daily seasonality
```

```
# Create sliding windows
def create_sliding_windows(features, target, timesteps):
    X, y = [], []
    for i in range(len(features) - timesteps):
        X.append(features[i:i+timesteps])  # Create a sequence of `timesteps`
        y.append(target[i+timesteps])      # Target is the value after the sequence
    return np.array(X), np.array(y)

X, y = create_sliding_windows(features, target, timesteps)

if np.isnan(X).sum() > 0 or np.isnan(y).sum() > 0:
    raise ValueError("NaN values detected in sliding window data.")
```

Finally, just a quick reminder, I am doing this analysis for an hourly time series data from 2012 to 2023. Coal will be the dependent variable and all other variables excluding the date will be the predictors. I am interested in doing a 5-fold cross validation starting from training with data from 2012 to 2017 and validating with 2018. Further using 2012 to 2018 and validating with 2019 and this sequentially increases till 2022 is the final validation set for training data from 2012 to 2021. And then lastly 2023 will be used as the testing data set and 2012 to 2022 will be the training data. Unfortunately google colab is not providing me the GPU today, so I will have to do this analysis using the CPU.

Prompt 9:

Would you recommend any additional changes in the following syntax given that I am using the google colab free tier CPU and the GPU?

```
# Sequential 5-Fold Cross-Validation

# Define yearly split ranges
fold_ranges = [
    (2012, 2017, 2018),
    (2012, 2018, 2019),
    (2012, 2019, 2020),
    (2012, 2020, 2021),
    (2012, 2021, 2022)
]

# Convert years to indices
def get_fold_indices(start_year, end_year, validation_year, df1):
    train_indices = df1[(df1['date'].dt.year >= start_year) & (df1['date'].dt.year <= end_year)].index
    val_indices = df1[df1['date'].dt.year == validation_year].index
    return train_indices, val_indices
```

```
# Initialize metrics storage
fold_metrics = []

for fold, (start_year, end_year, validation_year) in enumerate(fold_ranges):
    print(f"Fold {fold + 1}: Training on {start_year}-{end_year}, Validating on {validation_year}")

    # Get train and validation indices
    train_indices, val_indices = get_fold_indices(start_year, end_year, validation_year, df1)

    # Split data
    X_train, y_train = X[train_indices], y[train_indices]
    X_val, y_val = X[val_indices], y[val_indices]

    # Define LSTM model
    model = Sequential([
        LSTM(50, activation='relu', input_shape=(timesteps, X_train.shape[2])),
        Dense(1)  # Output layer
    ])
    model.compile(optimizer='adam', loss='mse')

    # Early stopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Train the model
    history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
                        epochs=20, batch_size=64, callbacks=[early_stopping], verbose=1)

    # Evaluate on validation set
    val_predictions = model.predict(X_val)
    val_rmse = np.sqrt(mean_squared_error(y_val, val_predictions))
    print(f"Validation RMSE for Fold {fold + 1}: {val_rmse}")

    # Store fold metrics
    fold_metrics.append({
        'fold': fold + 1,
        'val_rmse': val_rmse,
        'history': history
    })
```

Prompt 10:

In the following code for an LSTM neural network model in google colab I am getting a high RMSE and MAE. What can I do to get more accurate and precise forecasts? I will not be able to get more informative features or explanatory variables at this time. Whatever must be done will be within the limits of tweaking the hyperparameters of the LSTM model.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam  # Import Adam optimizer
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd

# Sliding window function
def create_sliding_windows(features, target, timesteps):
    X, y = [], []
    for i in range(len(features) - timesteps):
        X.append(features[i:i+timesteps])
        y.append(target[i+timesteps])
    return np.array(X), np.array(y)

# Create sliding windows
timesteps = 24
features = df1[['Gas', 'Gas-CC', 'Ren', 'Nuclear', 'Hydro', 'Other', 'year',
                'hour_sin', 'hour_cos', 'day_of_week_sin', 'day_of_week_cos',
                'day_of_year_sin', 'day_of_year_cos']].values
target = df1['Coal'].values

X, y = create_sliding_windows(features, target, timesteps)

# Adjust df1 to match X and y
df1 = df1.iloc[timesteps:].reset_index(drop=True)

# Define get_fold_indices
def get_fold_indices(start_year, end_year, validation_year, df1):
    train_indices = df1[(df1['date'].dt.year >= start_year) & (df1['date'].dt.year <=
end_year)].index
    val_indices = df1[df1['date'].dt.year == validation_year].index
    return train_indices.tolist(), val_indices.tolist()

# Cross-validation setup
fold_ranges = [
    (2012, 2017, 2018),
    (2012, 2018, 2019),
```

```
    (2012, 2019, 2020),
    (2012, 2020, 2021),
    (2012, 2021, 2022)
]

# Initialize metrics
fold_metrics = []

for fold, (start_year, end_year, validation_year) in enumerate(fold_ranges):
    print(f"Fold {fold + 1}: Training on {start_year}-{end_year}, Validating on
{validation_year}")

    # Get train and validation indices
    train_indices, val_indices = get_fold_indices(start_year, end_year,
validation_year, df1)

    # Use pre-scaled X and y
    X_train, y_train = X[train_indices], y[train_indices]
    X_val, y_val = X[val_indices], y[val_indices]

    print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")
    print(f"X_val: {X_val.shape}, y_val: {y_val.shape}")

    # Define model
    model = Sequential([
        LSTM(50, activation='tanh', return_sequences=True, input_shape=(timesteps,
X_train.shape[2])),
        Dropout(0.2),
        LSTM(50, activation='tanh', return_sequences=False),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

    # Early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=15,
restore_best_weights=True)

    # Train model
    history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
                        epochs=100, batch_size=64, callbacks=[early_stopping],
verbose=1)

    # Validate
    val_predictions = model.predict(X_val)

    # Evaluate in original scale (if applicable)
    val_rmse = np.sqrt(mean_squared_error(y_val, val_predictions))
```

```
    print(f"Validation RMSE for Fold {fold + 1}: {val_rmse}")

    # Append metrics
    fold_metrics.append({
        'fold': fold + 1,
        'val_rmse': val_rmse,
        'history': history
    })
```

Prompt 11:

I read all the information above. It seems extremely helpful. But I have computational resource constraints. I am doing this analysis using the google colab free resource and the GPU platform. So kindly suggest only those changes or modifications which will not significantly increase the computational work load. The GPU platform is available only for some time.

Prompt 12:

I want three layers and 128 neurons as per the following suggestion provided earlier by you:

Adjust LSTM Layers and Neurons
Increase or decrease the number of LSTM layers or neurons. For example:
Use 1 or 3 LSTM layers instead of 2.
Try configurations like 64 or 128 neurons per layer for more expressive capacity.
Also want the following: Add L2 regularization to LSTM layers to prevent overfitting. As per this code: from tensorflow.keras.regularizers import L2
model.add(LSTM(50, activation='tanh', return_sequences=True,
kernel_regularizer=L2(0.01), input_shape=(timesteps, X_train.shape[2]))) ALSO want to try this: Change the Activation Function
Replace tanh with relu in the LSTM layers to test if it captures trends better.
Finally I want the following: 7. Increase Epochs and Patience
Increase the number of epochs (e.g., 200) while adjusting the patience for EarlyStopping to allow the model to learn longer.
8. Sequence Length (Timesteps)
Adjust the sliding window size (timesteps) to capture more context from the data.
Try lengths like 48 or 12 (instead of 24).

Could you please include all these changes in the most recent fivefold cross validation code that you had provided me and furnish a new fresh syntax for the fivefold cross validation. I want to see whether the google colab GPU can handle the computational load.

Prompt 13:

Yesterday, you had provided information about the LSTM neural network mentioning its use for inherently sequential data with long term time dependencies. You had also shed light on the loss function plot from where overfitting and underfitting can be detected. Today can you please shed some light on what happens if we increase or decrease the hyperparameters of the LSTM. In fact, what are all these hyperparameters and how does the model behave if their value is set to a higher, medium, or a lower level. My hourly time series data has 105192 records or observations.

Prompt 14:

While using the LSTM model I had done 5-fold cross validation of the hourly time series data. For this, sequential folds were created. The model was run five times on incremental time series data and in the end a loss function was plotted. This function showed the training loss and validation loss. How to interpret this loss function plot. What is a good sign and what is a bad sign in this plot or in other words how do we know whether the plot of the loss function stands in favor of the model or against it? Also, what is the meaning of training loss and validation loss? What exactly they represent and mean? I need your help in explaining the loss function plot. What should the plot look like for a good model?

Prompt 15:

What hyperparameters should be decreased in case of overfitting? also kindly provide their precise values? also explain the difference between a batch size, units, sequence length, epoch, and timesteps with regards to LSTM?

Prompt 16:

With regards to the above discussion what will differ for patience = 15 and patience = 10