FORECASTING ANALYTICS

Project Report 3

Jason Sanghvi, Kamal Sunilkumar Patel

Contents

Team Name: Forecasting Experts

1. Jason Sanghvi
2. Kamal Patel

Contributions:

Kamal: The contribution to the project includes pre-processing data, exploratory data analysis and implementing models, and corresponding report write up.

Jason: The contribution to the project includes the implementation of the model and its analysis and the corresponding portion of the report write up.

1. Introduction

Electricity price forecasting is vital for energy market operations, guiding decisions for utilities, grid operators, policymakers, and market participants. Accurate forecasts optimize bidding strategies, enhance grid stability, support renewable energy integration, and enable cost-effective energy management. The aim of this project is to apply the methods and theory learned during the course to a real-world challenge: forecasting hourly Locational Marginal Prices (LMP) in a day-ahead electricity market.

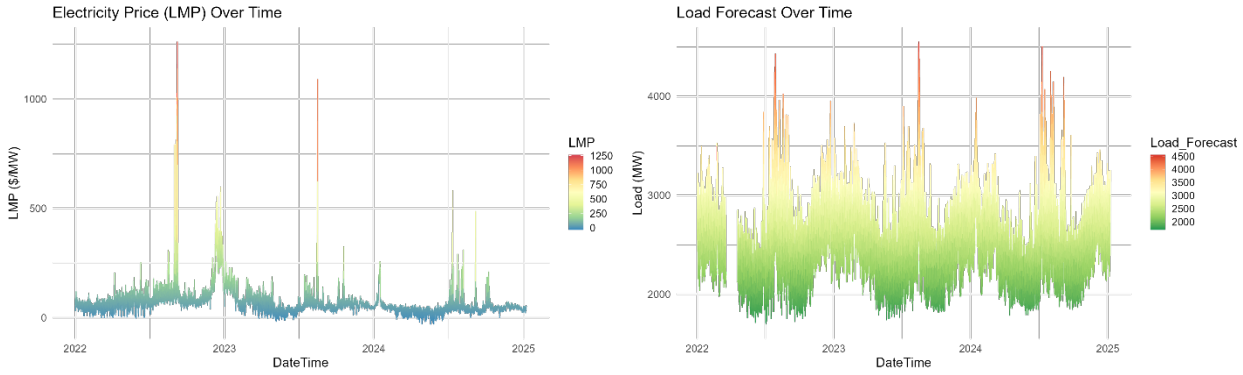
2. Problem Statement

The objective is to predict day-ahead Locational Marginal Prices (LMP) with an extensive set of data that contains previous LMP values and related exogenous variable values (Load Forecast, Solar Forecast, Wind Forecast, Fuel Price, and Air Temperature). Finding the relationship between these values and the day ahead hourly LMP will be crucial in capturing key features of the data, such as seasonality, that will overall lead to an accurate forecast.

3. Data Exploration

3.1 Visualizing the Data

Exploratory analysis focused on visualizing LMP patterns and relationships with exogenous variables. Time series plots (Figure 1) revealed key insights on each variable over time. We can clearly identify seasonality in LMP, with significant outlier points during late summer/early fall. Both the load forecast and air temperature also demonstrate a clear seasonality, even more explicitly, and they both follow somewhat similar patterns. Fuel price is shown to have a periodic nature, though it is relatively stable, excluding peaks in early 2023 and 2024. Solar and wind forecasts plots indicate that they are highly correlated, following similar structures of stability followed by massive spikes starting in the second quarter of 2024.



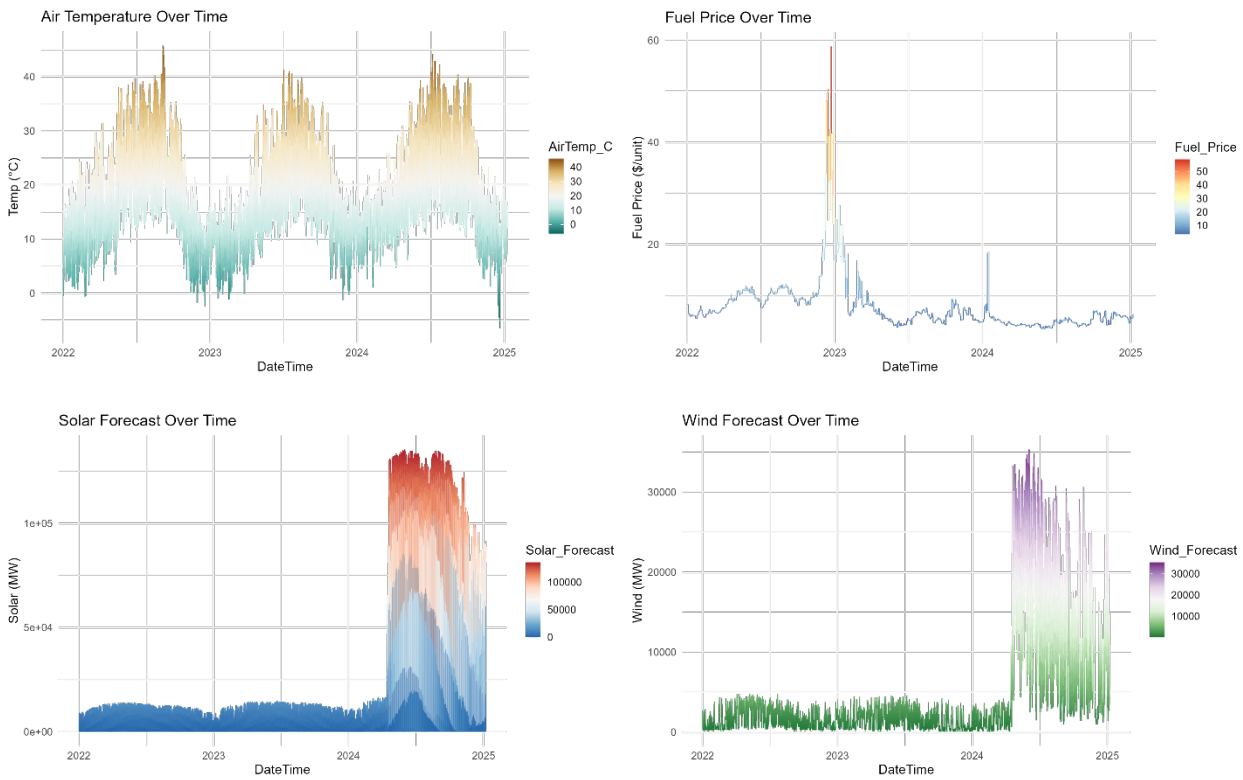
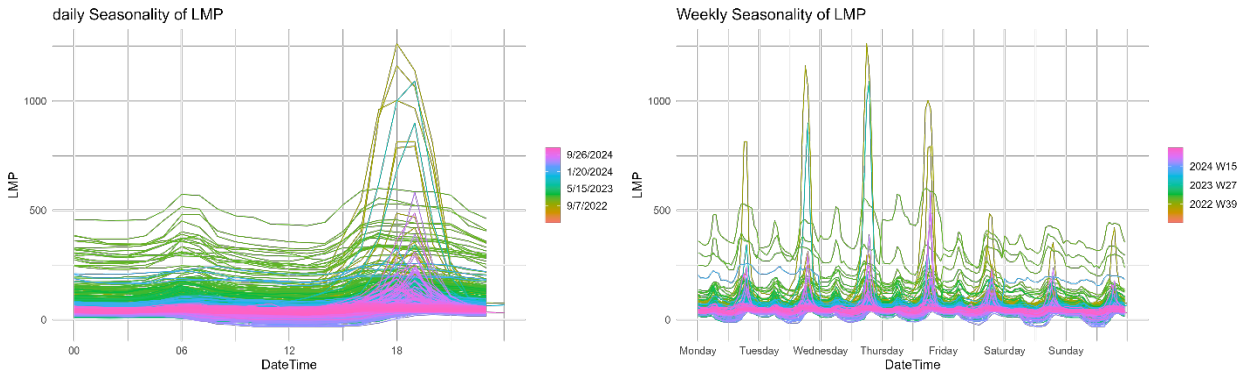


Figure 1: Time Plots

Seasonality Plots

From the below seasonal plots we can make distinct observations on the seasonality of LMP. The most noteworthy of these plots is the daily seasonal plot, as it demonstrates a clear and consistent pattern. This pattern indicates that LMP has a small peak around hour 6 and a major peak around the 18th hour. This indicates that hour might be a beneficial feature to consider in the model and the nonlinearity of the pattern indicates that a single discrete hour variable will likely not function effectively. The weekly plot also exhibits clear patterns, with there being a clear increase in LMP during weekdays compared to weekends, peaking between Tuesday-Thursday. The yearly plot demonstrates a noteworthy pattern that LMP peaks in the summer months, though the pattern is not as consistent as the daily and weekly plots. The monthly plot does not indicate significant enough patterns to warrant feature consideration.



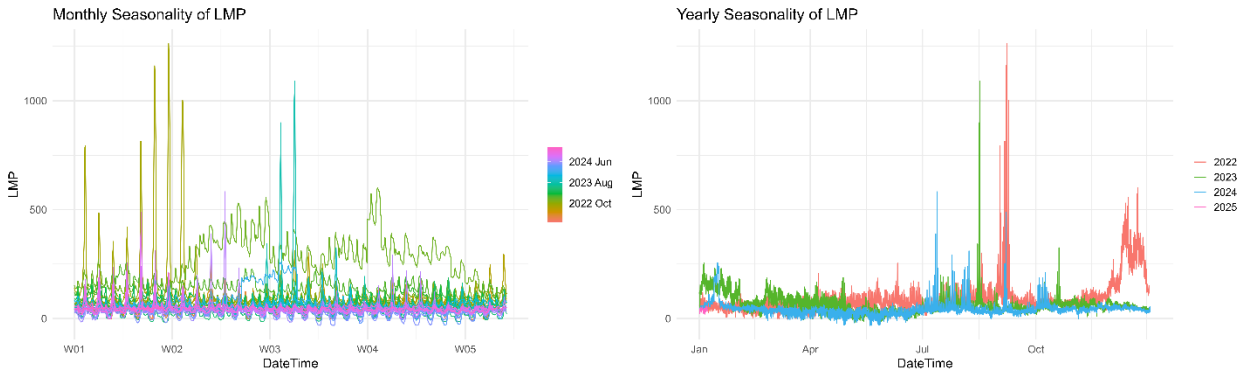


Figure 2: Daily (top left), Weekly (top right), Monthly (bottom left), and Yearly (bottom right) LMP Plots

3.2 Missing Regions of Data

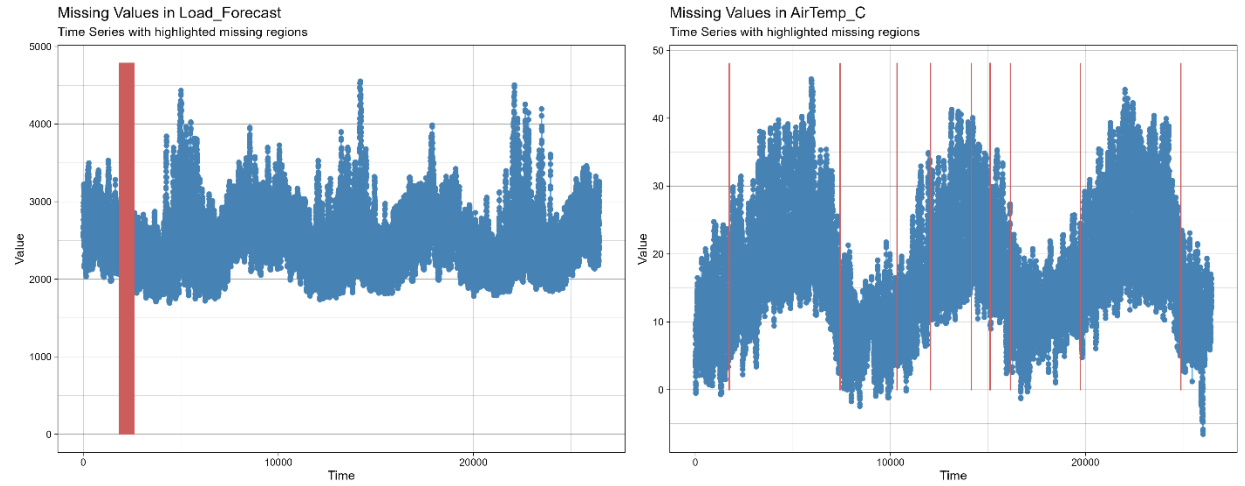


Figure 3: Missing Region in Data

After data analysis we observed that there were gaps in the data, in the Load Forecast and Air Temperature as shown in the figure above. There is large portion of missing data in Load Forecast, where in Air Temperature there are multiple separate regions where there is missing data. Filling in this missing data is important because it allows us to test models that consider all of the exogenous variables, without having to ignore a large portion of the testing data. Previously we tested mean imputation, however, while mean imputation preserves data volume, it also flattens temporal patterns. In order to mitigate this, we used linear interpolation to preserve time-series structure, while still avoiding production of excessive noise.

Fuel Price Forecasting

While not missing from the testing data, the Fuel Price is missing throughout the entirety of the test. While we could simply not consider Fuel Price in the model, correlation analysis provided valuable insights about the relationships between Fuel Price and other variables that indicated the necessity of its inclusion. The correlation heatmap (Figure 4) clearly demonstrates that fuel price exhibits the strongest positive correlation (0.8) with the Locational Marginal Price (LMP). This high correlation

confirms the critical role of fuel prices in predicting electricity market prices, validating the necessity of developing a reliable forecast for fuel price prior to the LMP forecasting stage.

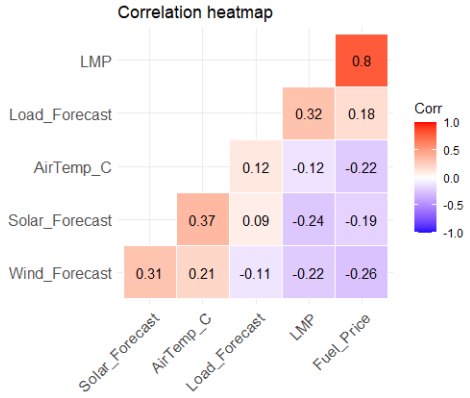


Figure 4: Correlation Heatmap

A univariate ARIMA model fit on the Fuel Price and residual diagnostics show that the residuals exhibit a relatively stable pattern (excluding extremities in early 2023 and early 2024), centered around zero with consistent variance (homoscedasticity) and no significant autocorrelation. The ACF is below significant values, indicating no significant autocorrelation remains in the residuals, implying that the model has adequately captured the time-dependent structure of the data.

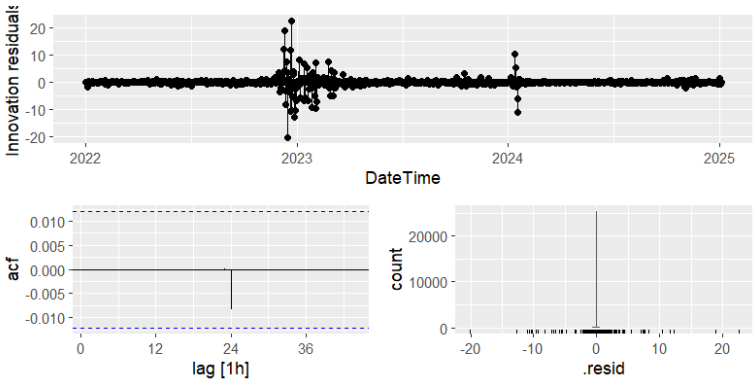


Figure 5: Time Plot, ACF, and Histogram of Residuals from ARIMA Modeled Fuel Price

3.3 Autocorrelation

Conducting the Augmented Dickey-Fuller (ADF) tests indicated that LMP was non-stationary ($p > 0.05$), requiring differencing. The ACF plot Figure 6 shows significant autocorrelation at lags 1 through 48, with notable peaks at lags 1, 24, and 48, indicating a strong daily seasonal pattern (24-

hour cycle or weekly seasonality). Even as the correlations gradually decay, they remain above the significance threshold, indicating that there is autocorrelation in the data.

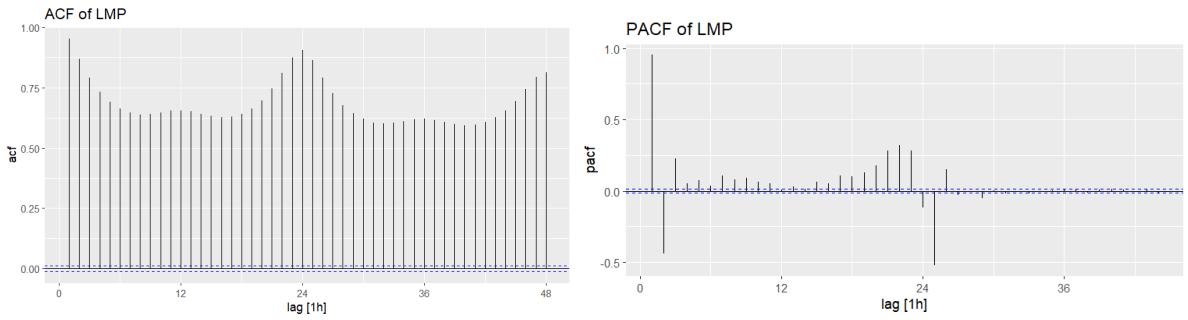


Figure 6: ACF and PACF of LMP

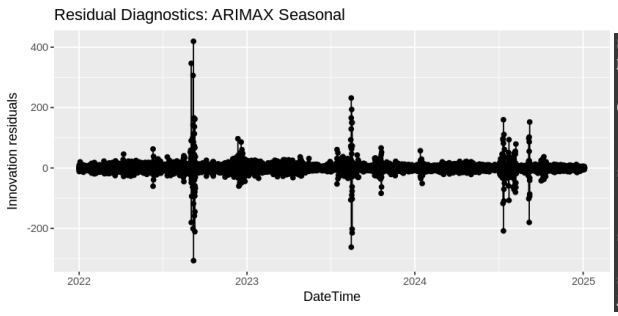
4. Final Methodology

4.1 Previous Models

Prior to this iteration of the forecast, we have tried a multitude of models: Naïve, Simple Exponential Smoothing, Seasonal ARIMA, Seasonal ARIMA-X, Dynamic Regression, and Dynamic Harmonic Regression. The model that consistently provided the best results was the Seasonal ARIMA-X model. We will analyze that model and its results to provide a baseline for our new model.

Seasonal ARIMA-X

To accurately capture seasonal influence along with correlation of the exogenous variables, we implemented a Seasonal ARIMA-X model. For this model, we found precise pdq and PDQ parameters for the model to optimize the model. The pdq parameters for the Seasonal ARIMA-X model were derived from the auto.arima function in r, which assesses the parameters by their resulting AIC values. The PDQ (seasonal) parameters were determined through manual evaluation techniques based on the resulting AIC values. The pdq parameters derived from the auto.arima function, (5,1,1), were fixed, so that the PDQ parameters could be altered. While this initially seems to be too computationally intensive, PDQ parameters of (1,1,1), (2,0,0), (0,2,0), and (0,0,2) produced either null values or over complexities that resulted in warnings from the R compiler. This means that only PDQ parameters that need to be evaluated are those that none of PDQ exceeds 1, and at least one of them is 0. Using these methods, we found that the optimal pdq is (5,1,1) and the optimal PDQ is (1,0,1), while the seasonal period was implicitly defined to be 24 in the model(ARIMA) function.




```
Series: LMP
Model: LM w/ ARIMA(5,1,1)(1,0,1)[24] errors

Coefficients:
         ar1      ar2     ar3      ar4     ar5      ma1     sar1    sma1
      1.1512  -0.4163  0.0991  -0.0729  0.0489  -0.9964  0.6947  0.0577
s.e.  0.0064   0.0095  0.0097   0.0094  0.0062   0.0007  0.0063  0.0087
      Load_Forecast  Solar_Forecast  Wind_Forecast  Fuel_Price  AirTemp_C
             0.0307          -4e-04         -5e-04      5.2451     0.1212
s.e.         0.0014           0e+00          1e-04      0.0839     0.0550

sigma^2 estimated as 97.61:  log likelihood=-97839.1
AIC=195706.2   AICc=195706.2   BIC=195820.7
```

Figure 7: Seasonal ARIMA-X model results

The Seasonal ARIMA-X model with optimized pdq and PDQ parameters, (5,1,1)(1,0,1)[24] produced a residual plot very similar to some of the other models, but was determined to be optimal, due to it providing the lowest MAE (4.0584), RMSE (9.8772), and AIC (195706.2), effectively balancing complexity and accuracy as shown in the Table 1 below.

Model	AICc	RMSE	MAE	MASE
Naive	NA	17.6	7.04	0.693
ETS	416362	16	6.16	0.606
ARIMA	201613	10.5	4.29	0.422
ARIMAX	199539	10.5	4.19	0.412
SARIMA	196751	9.94	4.10	0.403

Table 1: Model Comparison

4.2 Final Model

One type of model that we wanted to test is a neural network model, due to its ability to handle complex and nonlinear data and relationships. However, traditional neural networks process fixed sized vectors of data and do not contain the ability to retain previous information, making them relatively ineffective for time series data. One idea that we considered was explicitly adding autoregressive features to simulate an ARIMA type of learning, specifically adding lagged LMP values as features. However, for that model to provide any meaningful improvement over the Seasonal ARIMA-X model, it would likely need to be significantly more complex, which could lead to overfitting. We also considered a special type of neural network that more effectively handles time series data, recurrent neural networks. Recurrent neural networks function similarly to traditional neural networks, but they have a hidden state that is updated at each time step, which allows them to process sequences of data by retaining information at each step. However, this process of updating the hidden state at each time step uses a consistent set of weights, and because of this, updating the weights through backpropagation results in the gradient being multiplied many times. This means that when the weights are small, the gradient drastically decreases, or “vanishes”, and when the weights are large, the gradient drastically increases, or “explodes.” That means that RNNs are not very effective at learning long term patterns. This leads to the selection of the final model, an LSTM neural network, which is an alteration of a traditional RNN that solves the gradient issue. LSTM neural networks use LSTM units, or memory cells, that take in 3 states: input state (input value), hidden state (short term memory), and cell state (long term memory). The hidden state and cell state are passed from the previous cell, like a traditional RNN, however the way these values interact with the cell is what differentiates LSTMs from RNNs and minimizes the risk of gradient related issues. Within each cell, there are 3 gates: forget gate, input gate, and output gate. The forget gate takes the input and hidden state and transforms them with the use of weights and biases, along with the sigmoid activation function, to calculate a value that determines what percentage of the cell state should be remembered. The input gate does a similar transformation but uses the tanh activation function in

addition to the sigmoid function, to calculate a value that determines what value should be added to the cell state after it passes through the forget gate. The cell state value after passing through the input gate is the new cell state that will be outputted from the cell. The output gate uses another set of weights and biases, along with both activation functions, to transform the input state, the hidden state, and the new cell state to calculate the new hidden state that will be output from the cell. The functionality of these gates prevents the hidden state from being multiplied repeatedly, which allows the gradient to remain stable over many time steps. The ability of LSTM neural networks to capture long term patterns on time series data makes it an optimal choice of model for this project.

4.3 Optimizing the Model

The results of the LSTM model in its initial iteration, which predicted LMP using the currently defined exogenous variables as features, produced mixed results. The model produced slightly improved accuracy compared to the Seasonal ARIMA-X model but also displayed a high autocorrelation in its residuals. This indicates that the LSTM model is likely a better model than the Seasonal ARIMA-X model but is currently not being implemented in an effective way, as it is missing clear temporal patterns in the data. Two main techniques were used to successfully overcome this: seasonally differencing the data and adding lagged LMP (and seasonally differenced LMP) as features. Seasonally differencing the data transformed the data to be closer to stationary, which led to reduction in the autocorrelation of residuals. Adding lagged values for the variable we are trying to predict adds an autoregressive term to the model, simulating the success of the Seasonal ARIMA-X model. The lagged value features were added in increments of 24 (daily) ranging between a 1-day lag and 7-day lag (weekly). This was chosen to provide the model with the most relevant data and to prevent the model from compounding its errors by relying on forecasted data (excluding forecasted Fuel Price data). Since the test set only includes 1 day that we need to make predictions for and the lagged values are at minimum lagged by 1 day, the features that are input into the model for every entry in the test data are definite and not forecasted. These two major changes made the LSTM model viable. After those changes were made, we used feature engineering to add relevant time-based features to improve the accuracy of the model. Dummy variables were implemented for hours, day of week, and month to effectively capture the observations made in the data exploration section of this paper. Lastly, we altered the loss function used during model training to be the Huber loss function. The Huber loss function is a hybrid loss function as it uses MSE for small errors and MAE for large errors. This reduces sensitivity to outliers, which is significant due to the presence of a few major outliers in LMP data, and thus seasonally differenced LMP data.

5 Validation and Experiments

5.1 Initial LSTM Results

As explained in the previous section, the initial LSTM model, which used Load Forecast, Solar Forecast, Wind Forecast, Fuel Price, and Air Temperature as features to predict LMP, displayed a high autocorrelation of residuals, despite sufficient accuracy. This was confirmed to be a definitive flaw of the model as we tested 1000 epochs (training iterations) and though slightly improved, the model

still produced insufficiently high autocorrelation of residuals. This is significant because it demonstrates that the high autocorrelation of residuals was not caused due to an underfit model and indicates that model will be overfit before it reaches a sufficiently low autocorrelation of residuals.

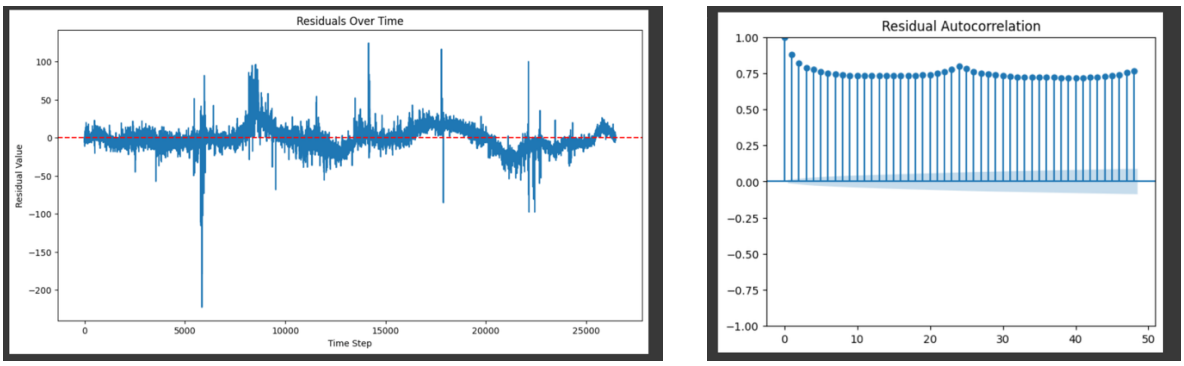


Figure 8: Time Plot and ACF of Residuals of Initial LSTM Model

5.2 Optimized LSTM Results

Making the improvements to the model drastically improved its performance as indicated by the residual plot and ACF plot below by adding featured variables like capturing peak hours and weekday where LMP price shows peak values.

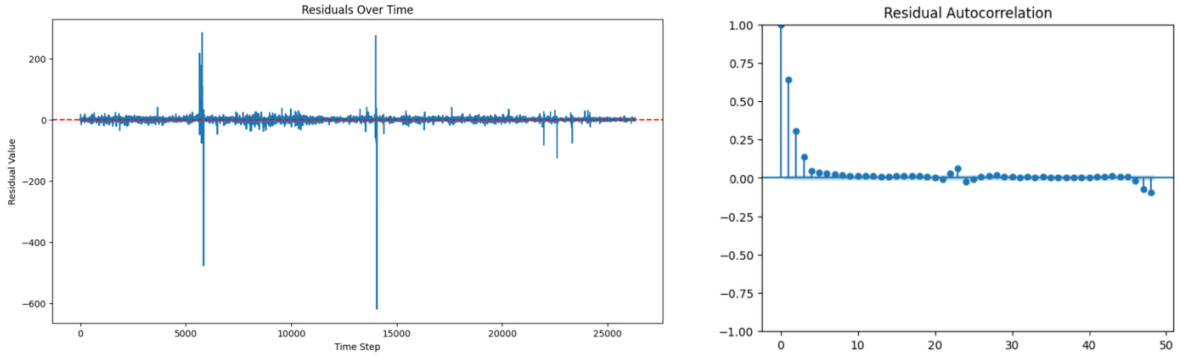


Figure 9: Time Plot and ACF of Residuals of Optimized LSTM Model

The residual ACF plot does indicate that the model has not completely captured all the temporal patterns of the data, as the first two lagged values show a meaningful residual autocorrelation. However, it is sufficient enough to demonstrate a good capturing of the temporal pattern of the data, and adding 1 and 2 hour lagged features to combat this would not only be difficult to implement as the features would need to update each run with forecasted values, but it would likely lead to a compounding of errors that would reduce the overall forecasting accuracy in a 24 hour window.

One noteworthy observation from the residual plot, is that despite the clear improvement over initial LSTM model, the residual extremities are greater in absolute value than those from the initial model. This, however, makes sense, as these residual peaks directly align with the outliers in the LMP data, and seemingly match the expected magnitude given the level of extremity shown in the LMP time

plot. The fact that the residuals in the initial LSTM model have such a low magnitude indicates that it is likely that the model is overfit and memorizing values rather than fitting patterns. This is supported by the clearly observable seasonal patterns shown in its residual plot.

5.3 Accuracy

While the residual plots seem to indicate the improved accuracy of the optimized LSTM model over the initial LSTM model and the Seasonal ARIMA-X model, the histogram of residuals provides definitive conclusions.

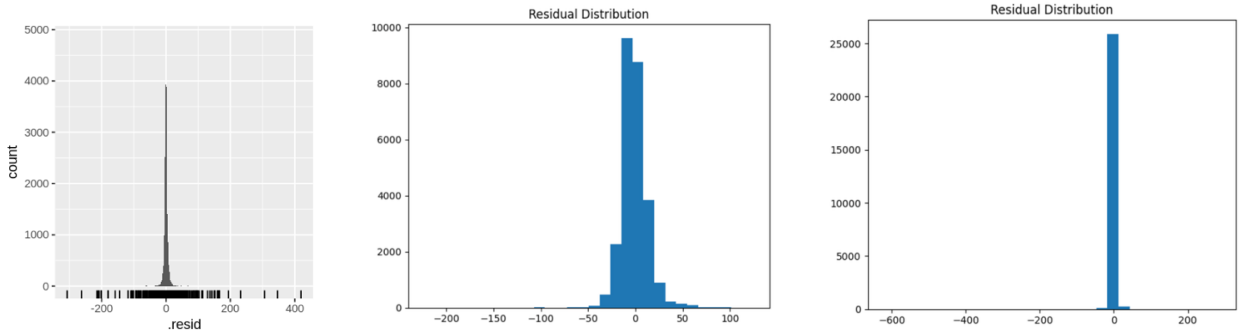


Figure 10: Histogram of Residuals of Seasonal ARIMA-X (left), Initial LSTM (middle), and Optimized LSTM (right) Models

These histograms clearly demonstrate that the optimized LSTM model produces residuals that are the most closely centered around 0, indicating great accuracy. In order to confirm that this is not the result of overfitting, we ran the previous deliverables' train and test sets through both the Seasonal ARIMA-X and optimized LSTM models and compared the resulting forecasts to the released data (simulating use of models for each deliverable), and the optimized LSTM model produced more accurate results (MAE) for 6 of the 7 deliverables, further validating it as the final model selection.

6 Conclusions and Future Direction

Throught visual and statistical diagnostics, this final iteration of models, an LSTM neural network, is a substantial improvement over the previously optimal model, a Seasonal ARIMA-X model. The accuracy is meaningfully improved as evidenced by visual analysis on the corresponding residual plots and relative performance of the models on previous deliverable train and test sets. However, it is also notable that the LSTM model requires significant more optimization to be a sufficient model. While providing sufficient levels of accuracy, the initial LSTM model had extremely high autocorrelation of residuals, indicating a very poor forecast. Seasonally differencing LMP and adding lagged LMP features greatly improved the model, turning a very poor forecast into a good forecast. After that, small improvements were made by adding time-based features along with optimizing the loss function with respect to the data. These improvements emphasize the importance of data exploration, as these improvements were made based on analysis from data exploration.

While this final model performs well, the high autocorrelation of residuals for time lags of 1 and 2 indicate the further improvements can be made to the model. Further exploration would be required

to determine what methods can be applied to resolve this issue without including forecasted values (excluding Fuel Price) in the model features, to remain resistant to compounding forecast errors.

7 Lessons Learnt

- Thorough EDA and visualization of trends, seasonality, and missing data guide effective preprocessing and feature selection.
- Traditional statistical models like SARIMA work well, but neural network LSTMs handle nonlinearity better with proper tuning to avoid overfitting.
- Understanding of non-stationarity & seasonality helped in analyzing ACF/PACF plots to identify optimal lag structures, feature Engineering Lagged values and adding time-based variables (hour, weekday, month) to capture temporal patterns.
- We learnt to compare other simple and autoregressive models and their residuals diagnostics.
- Understanding the concepts of time series analysis was as crucial as coding and trying ideas to build models that are accurate and useful. So, we learnt great amount of R language.
- Overall, we learned to build a workflow from data preprocessing, diagnostics analysis to building complex machine learning models.
- More complex models require finer tuning to properly fit data without overfitting.

8 Appendix – Codes

README

*Attached code in Jupyter Notebook file with outputs and plots shown from last submitted deliverable. To run code, use attached train and test excel files (modified to add all given values to train set and only unknown entries in test set). Appendix code contains training lines due to its necessity for accuracy as model initializes with unoptimized weights without training.

CODE

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# import data from excel files
train_df = pd.read_excel("Train_Round8.xlsx", parse_dates=['DateTime'])
```

```
test_df = pd.read_excel("Test_Round8.xlsx", parse_dates=['DateTime'])

# convert DateTime column to datetime type
train_df['DateTime'] = pd.to_datetime(train_df['DateTime'], format='%Y-%m-%dT%H:%M:%S-08:00', utc=True)
train_df['DateTime'] = train_df['DateTime'].dt.tz_convert('US/Pacific')
test_df['DateTime'] = pd.to_datetime(test_df['DateTime'], format='%Y-%m-%dT%H:%M:%S-08:00', utc=True)
test_df['DateTime'] = test_df['DateTime'].dt.tz_convert('US/Pacific')

# set DateTime as the key index
train_df.set_index("DateTime", inplace=True)
test_df.set_index("DateTime", inplace=True)

# interpolate missing entries in training data
train_df.interpolate(method='time', inplace=True)

# use arima to forecast fuel price for testing data
fuel_model = SARIMAX(train_df['Fuel_Price'], order=(0,0,0), seasonal_order=(1,0,1,24))
fuel_model_fit = fuel_model.fit()
test_df['Fuel_Price'] = fuel_model_fit.forecast(len(test_df))
print(test_df['Fuel_Price'])

# make copy of train and test data to preserve values before scaling
unscaled_test_df = test_df.copy()
unscaled_train_df = train_df.copy()

# feature engineering time based features (month, hour, and day of week)
def add_time_features(df):

  df['Month'] = df.index.month
  df['Hour'] = df.index.hour
  df['Dayofweek'] = df.index.dayofweek

  # Add dummy variables corresponding to hour
  hour_vars = pd.get_dummies(df['Hour'], prefix='Hour')

  # Add dummy variables corresponding to month
  month_names = {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June', 7: 'July', 8: 'August', 9:
'September', 10: 'October', 11: 'November', 12: 'December'}
  df['Month_Name'] = df['Month'].map(month_names)
  month_vars = pd.get_dummies(df['Month_Name'])

  # Add 0 columns for months not in df
  for month in month_names.values():
   if month not in month_vars.columns:
    month_vars[month] = 0
```

```
    # Sort months
    month_vars = month_vars[[month_names[i] for i in range(1,13)]]

    # Add dummy variables corresponding to day
    day_names = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
    df['Day_Name'] = df['Dayofweek'].map(day_names)
    day_vars = pd.get_dummies(df['Day_Name'])

    # Add 0 columns for days not in df
    for day in day_names.values():
      if day not in day_vars.columns:
        day_vars[day] = 0

    # Sort days
    day_vars = day_vars[[day_names[i] for i in range(7)]]

    # convert to float
    hour_vars = hour_vars.astype(float)
    month_vars = month_vars.astype(float)
    day_vars = day_vars.astype(float)

    # add onlt selected features to df
    df = pd.concat([df, hour_vars, month_vars, day_vars], axis=1)
    df.drop(['Month', 'Hour', 'Dayofweek', 'Month_Name', 'Day_Name'], axis=1, inplace=True)

    return df

# add time based features to train and test df
train_df = add_time_features(train_df)
test_df = add_time_features(test_df)

# Adding day lagged LMP columns
train_df['LMP_Lag24'] = train_df['LMP'].shift(24)
train_df['LMP_Lag48'] = train_df['LMP'].shift(48)
train_df['LMP_Lag72'] = train_df['LMP'].shift(72)
train_df['LMP_Lag96'] = train_df['LMP'].shift(96)
train_df['LMP_Lag120'] = train_df['LMP'].shift(120)
train_df['LMP_Lag144'] = train_df['LMP'].shift(144)
train_df['LMP_Lag168'] = train_df['LMP'].shift(168)

# retrieve lagged LMP values for test set
test_lag = train_df['LMP'].tail(168)

# Adding day lagged seasonal differenced LMP columns
train_df['LMP_Seasonal_Differenced_Lag24'] = train_df['LMP_Lag24'] - train_df['LMP_Lag48']
train_df['LMP_Seasonal_Differenced_Lag48'] = train_df['LMP_Lag48'] - train_df['LMP_Lag72']
```

```
train_df['LMP_Seasonal_Differenced_Lag72'] = train_df['LMP_Lag72'] - train_df['LMP_Lag96']
train_df['LMP_Seasonal_Differenced_Lag96'] = train_df['LMP_Lag96'] - train_df['LMP_Lag120']
train_df['LMP_Seasonal_Differenced_Lag120'] = train_df['LMP_Lag120'] - train_df['LMP_Lag144']
train_df['LMP_Seasonal_Differenced_Lag144'] = train_df['LMP_Lag144'] - train_df['LMP_Lag168']

# Adding day lagged LMP columns
test_df['LMP_Lag24'] = test_lag[144:].values
test_df['LMP_Lag48'] = test_lag[120:144].values
test_df['LMP_Lag72'] = test_lag[96:120].values
test_df['LMP_Lag96'] = test_lag[72:96].values
test_df['LMP_Lag120'] = test_lag[48:72].values
test_df['LMP_Lag144'] = test_lag[24:48].values
test_df['LMP_Lag168'] = test_lag[:24].values

# Adding day lagged seasonal differenced LMP columns
test_df['LMP_Seasonal_Differenced_Lag24'] = test_df['LMP_Lag24'] - test_df['LMP_Lag48']
test_df['LMP_Seasonal_Differenced_Lag48'] = test_df['LMP_Lag48'] - test_df['LMP_Lag72']
test_df['LMP_Seasonal_Differenced_Lag72'] = test_df['LMP_Lag72'] - test_df['LMP_Lag96']
test_df['LMP_Seasonal_Differenced_Lag96'] = test_df['LMP_Lag96'] - test_df['LMP_Lag120']
test_df['LMP_Seasonal_Differenced_Lag120'] = test_df['LMP_Lag120'] - test_df['LMP_Lag144']
test_df['LMP_Seasonal_Differenced_Lag144'] = test_df['LMP_Lag144'] - test_df['LMP_Lag168']

# Adding seasonal differenced LMP column
train_df['LMP_Seasonal_Differenced'] = train_df['LMP'] - train_df['LMP_Lag24']
test_df['LMP_Seasonal_Differenced'] = test_df['LMP'] - test_df['LMP_Lag24']

# Remove rows with null values that were produced from adding lag columns
train_df = train_df.dropna(subset = ['LMP_Lag24', 'LMP_Lag48', 'LMP_Lag72', 'LMP_Lag96', 'LMP_Lag120',
'LMP_Lag144', 'LMP_Lag168'])
# Define features and target of model
columns = train_df.columns.tolist()
features = columns[1:-1]
target = 'LMP_Seasonal_Differenced'

# scale the features
scaler_X = MinMaxScaler()

train_df[features] = scaler_X.fit_transform(train_df[features])
test_df[features] = scaler_X.transform(test_df[features])

# Create dataset class that will be used in pytorch LSTM model
# input_window - number of previous data points that will be used to make prediction
# output_window - number of data points that model will predict at each iteration.
class LMPDataset(Dataset):
 def __init__(self, df, features, target, input_window=24, output_window=1):
   self.X = []
```

```
    self.y = []

    for i in range(len(df) - input_window - output_window + 1):
      x_seq = df[features].iloc[i:i+input_window+output_window].values
      y_seq = df[target].iloc[i+input_window:i+input_window+output_window].values
      self.X.append(x_seq)
      self.y.append(y_seq)

    self.X = torch.tensor(np.array(self.X), dtype=torch.float32)
    self.y = torch.tensor(np.array(self.y), dtype=torch.float32)

  def __len__(self):
    return len(self.X)

  def __getitem__(self, idx):
    return self.X[idx], self.y[idx]
# Create model class that will be used in pytorch LSTM model
# hidden_size - number of hidden neurons
# num_layers - number of stacked layers
# dropout - percentage of deactivated neurons (helps reduce overfitting)
class LSTMModel(nn.Module):
  def __init__(self, input_size, hidden_size=64, num_layers=2, dropout=.2):
    super(LSTMModel, self).__init__()
    self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size, num_layers=num_layers,
batch_first=True, dropout=dropout)
    self.fc = nn.Linear(hidden_size, 1)

  def forward(self, x):
    lstm_out, _ = self.lstm(x)
    out = self.fc(lstm_out[:, -1, :])
    return out
input_window = 24
batch_size = 32 # number of entries ran through the model before optimize
epochs = 250   # number of full training iterations
learning_rate = .001 # rate in which model parameters are adjusted
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# define objects that will facilitate use of pytorcch
train_dataset = LMPDataset(train_df, features, target, input_window=input_window)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)

# define the initial model
model = LSTMModel(input_size=len(features)).to(device)
criterion = nn.SmoothL1Loss() # Huber Loss function
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
# model training
for epoch in range(epochs):
 model.train()
 total_error = 0

 for batch_x, batch_y in train_loader:
  batch_x, batch_y = batch_x.to(device), batch_y.to(device)
  optimizer.zero_grad()
  output = model(batch_x).squeeze()
  loss = criterion(output, batch_y.squeeze())
  loss.backward()
  optimizer.step()
  total_error += loss.item()

# attach end of training set to test set for input window purpose
prev_input = train_df.tail(input_window)
test_input_df = pd.concat([prev_input, test_df], ignore_index=True)

# define objects that will facilitate use of pytorch
test_dataset = LMPDataset(test_input_df, features, target, input_window=input_window)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)

# using model on test set to make predictions
model.eval()
predictions = []

with torch.no_grad():
 for x, _ in test_loader:
  x = x.to(device)
  pred = model(x).cpu().numpy()
  predictions.append(pred)

predictions = np.array(predictions).reshape(-1, 1)

# add forecasted seasonally differenced LMP values to lagged LMP values to get LMP values
unscaled_test_df['LMP'] = predictions + unscaled_train_df['LMP'].tail(input_window).values.reshape(-1, 1)

# export to csv
unscaled_test_df.to_csv('Results_8.csv', index=True)
```