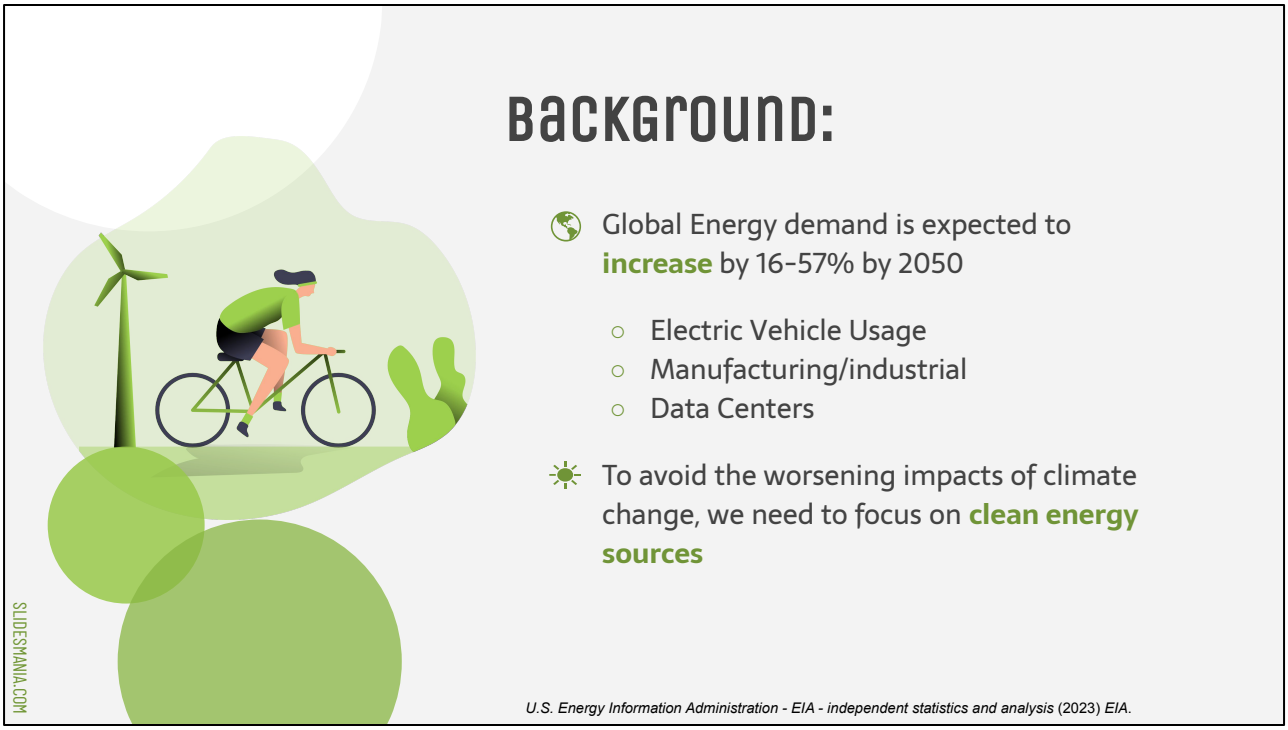
Chasing the Wind: Key Factors Driving Renewable Energy

Fall '24
DATASCI 207

Sonia Chakraborty
Jordan Lawhon
Tim Leong
Hunter Tonn
Mina Baghai



The U.S. energy information administration predicts that the global primary energy demand will increase anywhere from 16-57% by 2050. This is due to a rise in electric vehicle usage, data centers, increase industrial production, and a continuously globalizing planet.

To meet this increasing demand, while avoiding the worsening impacts of climate change, we must look into increasing the supply of renewable energy sources. For our project today we have chosen to focus on the possibility of a wind energy generation site using a kaggle data set from Germany.

Question

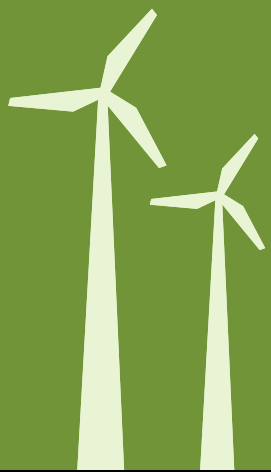
? How do atmospheric factors influence wind speed?

? How can we predict hourly wind speed to estimate energy production for utility providers throughout the year?

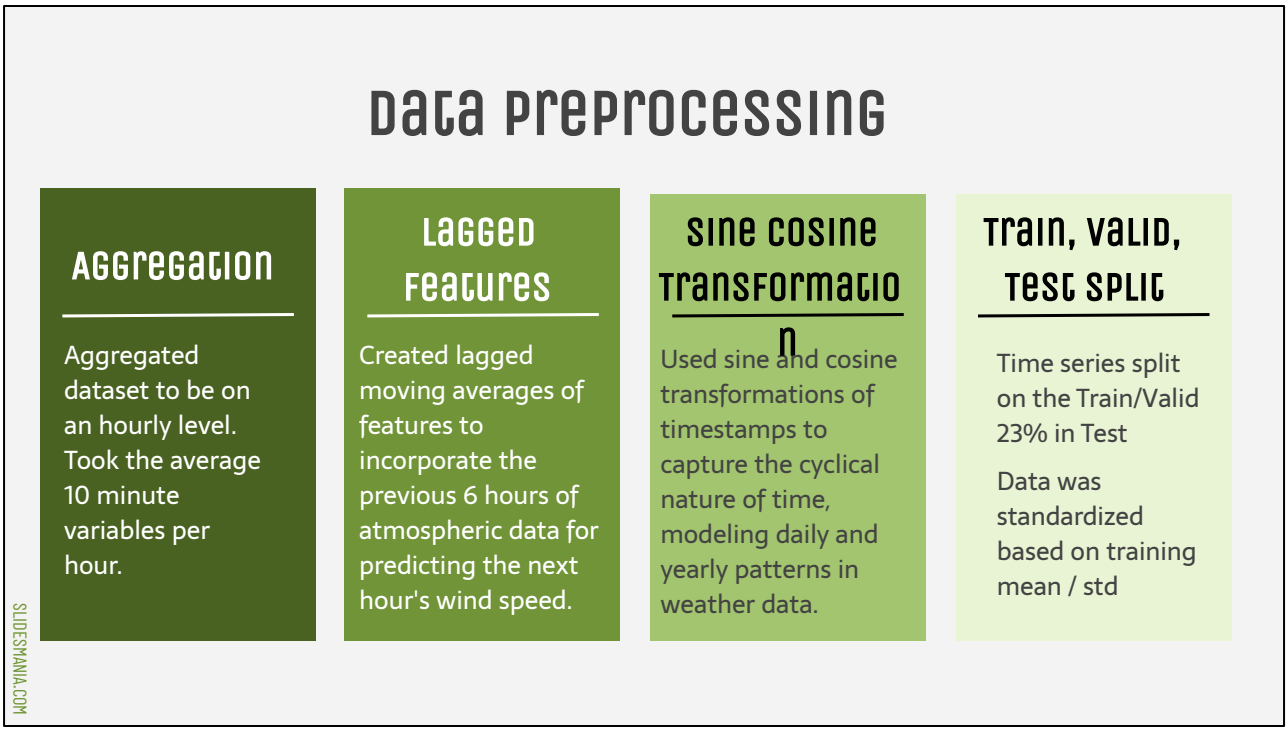
SUB QUESTION:
Can we forecast extreme weather conditions so that utility groups can take precautionary measures to avoid disruptions in energy production?

Data source

- *Kaggle's Weather Long-Term Time Series Forecasting Dataset*
  - 2020 meteorological indicators measured at a Max Planck Institute weather station (located in Germany)
  - Contains atmospheric variables including air temperature, humidity, wind patterns, radiation, and precipitation **recorded every 10 minutes**
  - 20 variables. 52,560 data points per variable (365 days × 24 hours × 6 measurements per hour)
- Target Variable: **average hourly wind speed**



Mina



Mina

To make the target variable of wind speed more actionable for energy decision-making, we aggregated the data to an hourly level by averaging the 10-minute values within each hour. This helps simplify the data and makes it more aligned with more conventional decision-making energy management, where decisions are often made on an hourly basis rather than every 10 minutes.

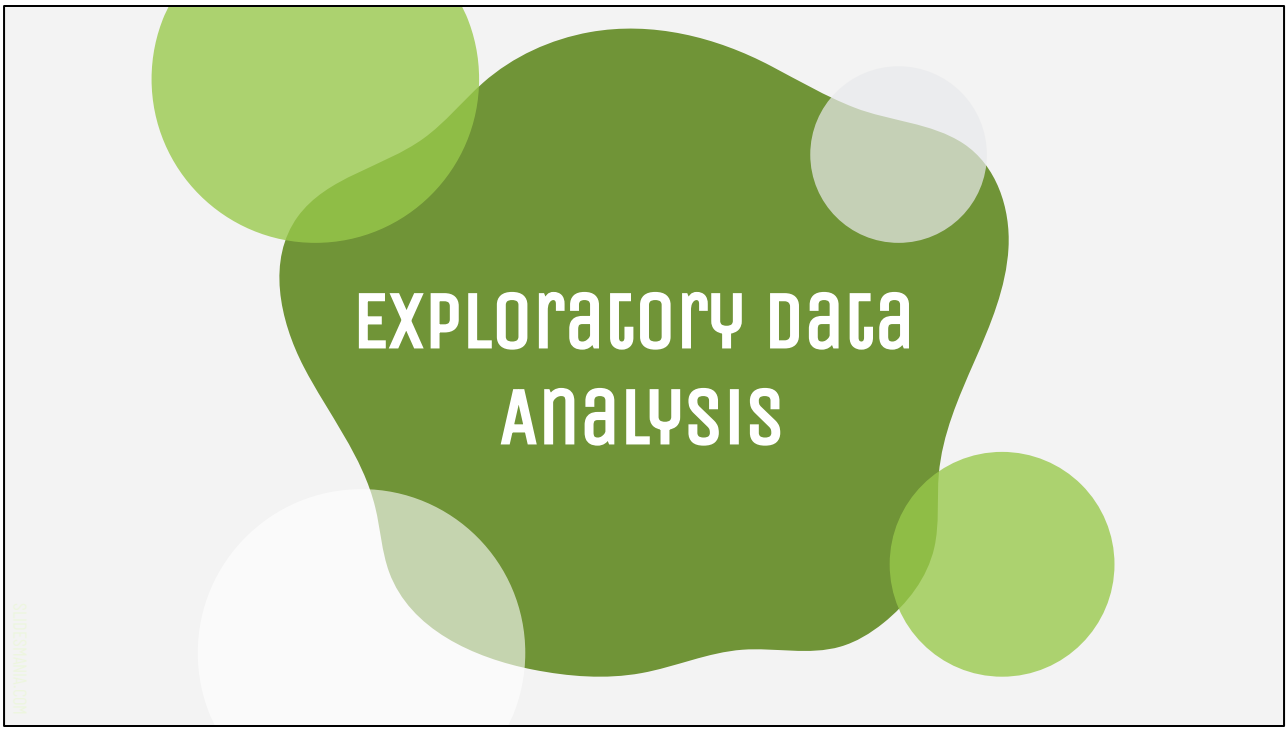
Next we created lagged moving averages of all our features, incorporating the previous 6 hours of atmospheric data to predict the next hour's wind speed. The purpose of creating these lagged features is to simulate a real-world scenario where we would use past data to predict future wind speed. Since wind speed can change rapidly, having a prediction with a slight time lag allows for more informed decision-making and gives time to react when it comes to energy management.

- Extra: Wind speed can fluctuate throughout the day, so using a 6-hour window helps smooth out immediate fluctuations while still capturing short-term trends in atmospheric conditions.

Weather data has clear daily and yearly periodicity. We want to create features using sine and cosine transformations to create "Time of day" and "Time of year" signals to capture these patterns.

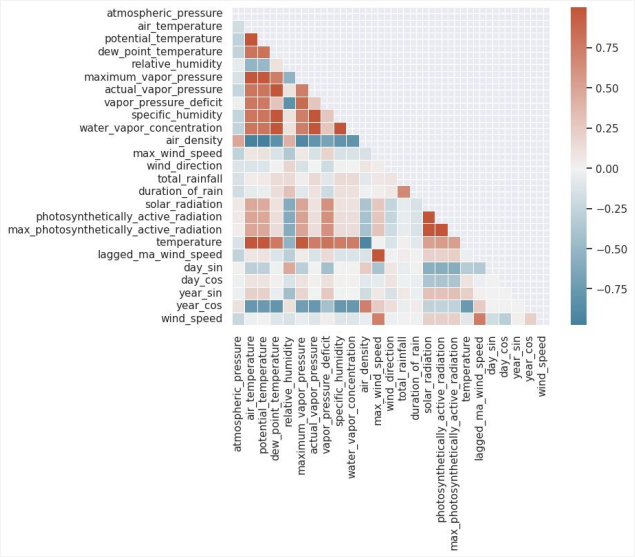
For the train valid, test, split we did a time series split. So for the testing set we took the last week of each month which was about 23% of the total data. This is technique

commonly done in weather forecasting modeling since weather data is sequential and we will always be using past data to predict future data. However we only have one years worth of data so rather this workaround was used rather than doing one year for training and the next year for testing for example.



After we did some data preprocessing, we wanted did some EDA to better understand our data.

correlation matrix



Positively correlated :

- Solar radiation,
- Lagged wind speed,
- Year cos

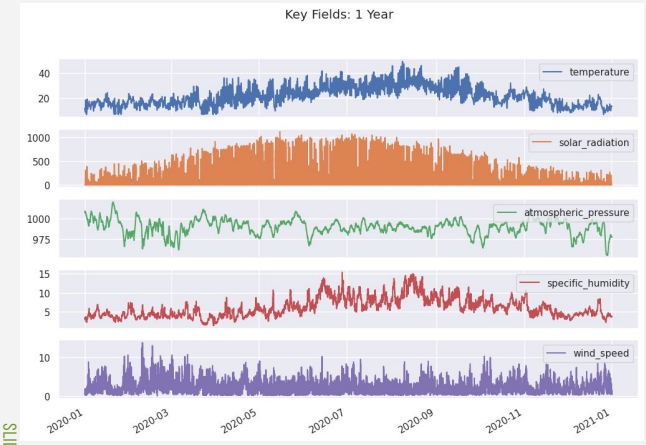
Negatively correlated:

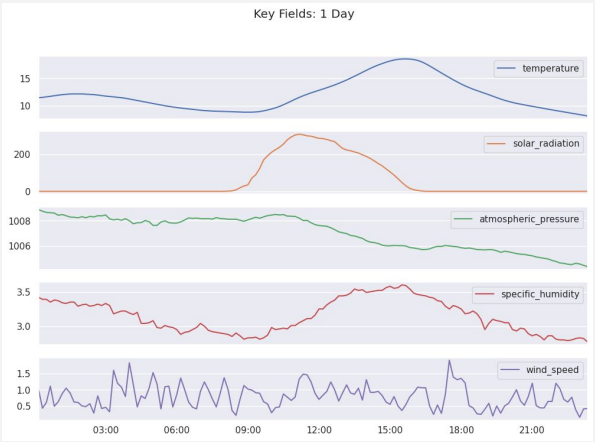
- Atmospheric pressure,
- Humidity,
- Dew point humidity,
- Day cos

For the first step in our EDA process, we took a look at the correlation matrix for our variables. When we looked at the matrix, we noticed there were a number of highly correlated variables which indicated multicollinearity among our variables. As an example, you can see that maximum vapor pressure has a high degree of correlation with air temperature and potential temperature. These multicollinear variables are problematic because they can cause our model to overfit.

We also looked at how the explanatory variables were correlated with wind speed. We noticed that solar radiation, lagged wind speed, and year cosine were positively correlated with wind speed, while atmospheric pressure, humidity, and day cosine were negatively correlated with wind speed.

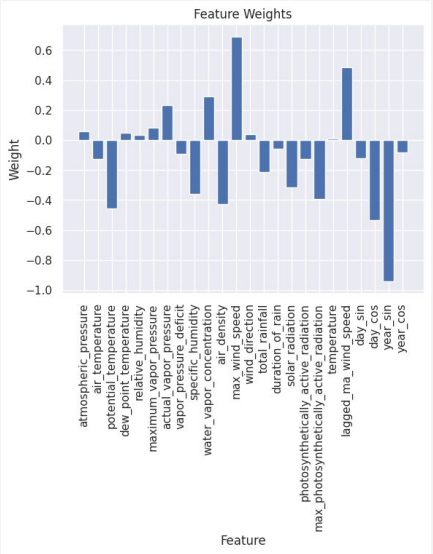
Time Series Analysis





Next we took a look at some of the variables that were correlated with our target variable across various time horizons. Looking at the 1 year time series, we can see that wind speed is higher in the winter and lower during spring and summer. This highlights why our year cosine variable was correlated with our target variable. When we look at the 1 day time series, we can see that wind speed is much more erratic than the other variables that have more of a cyclical nature to them. This is why we saw in the previous slide that our lagged wind speed variable had the highest degree of correlation with predicting future wind speed.

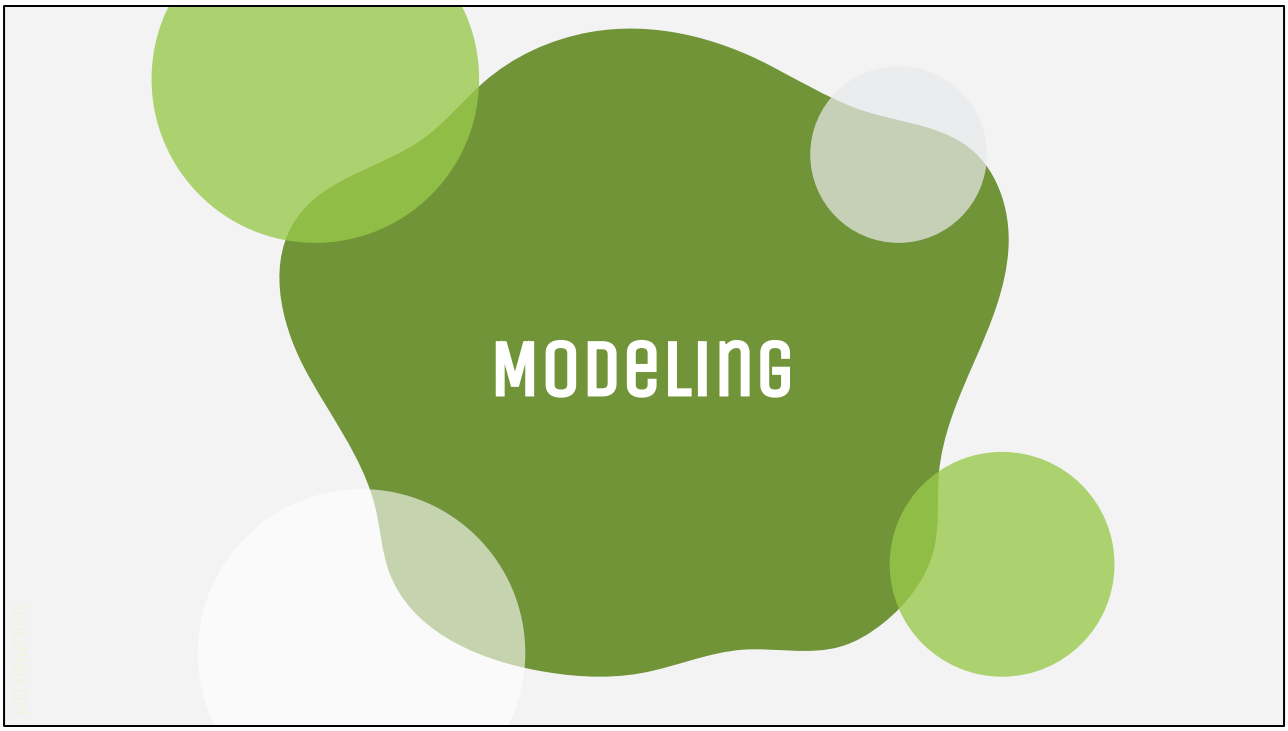
Feature Weights



Key Variables

- Potential temperature
- Actual vapor pressure
- Specific humidity
- Air density
- Max wind speed
- Solar radiation
- Max photosynthetically active radiation
- Lagged ma wind speed
- Day cosine
- Year sine

Finally we ran an initial linear regression to understand the various weights attributed to each variable. There are some differences between the feature weights identified here compared to the correlation matrix, but for the most part the majority of the key variables were similar across the two analyses. From our analysis, these variables were the most important to predicting wind speed.


MODELING

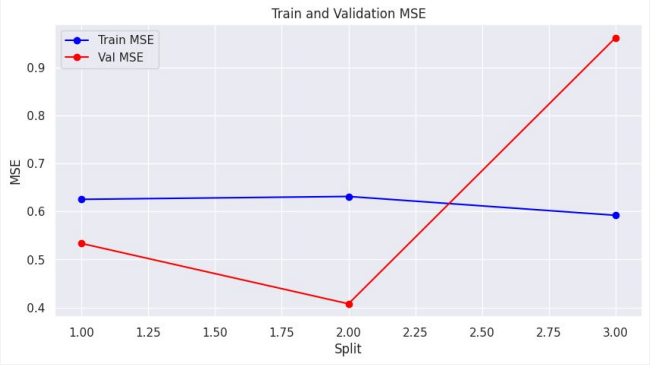
Baseline Model

- Predict wind speed as the average wind speed from the training dataset
- Train MSE: 2.32
- Valid MSE: 2.31

Mina

Linear Regression Model

- Ridge Regression Model
  - TimeSeriesSplit cross validation
- Begins to overfit the data




```
Average Train MSE: 0.6162
Average Validation MSE: 0.6343
```


Sonia
We implemented Ridge Regression model and utilized TimeSeriesSplit cross-validation to ensure training data precedes validation which protects the temporal order of the time series data we used.

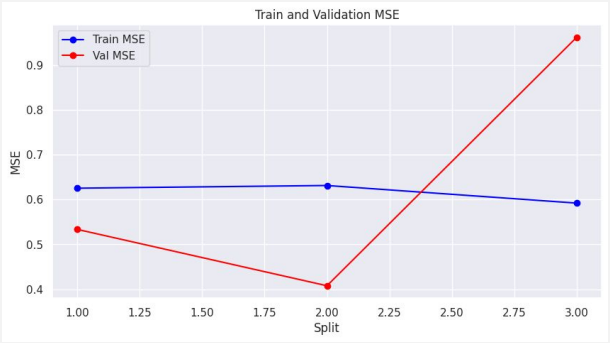
As you can see, the avg training MSE was 0.6162 and the avg validation MSE was 0.6343 showing that the model generalized rather well

However, looking at the graph and MSE splits in more detail, we see the train and validation lines start close, intersect, and then part ways. With the validation MSE jumping much higher. This indicates that the model begins overfitting and fits the training data well but is unable to generalize to unseen data.

This still happened even hyper parameter tuning*

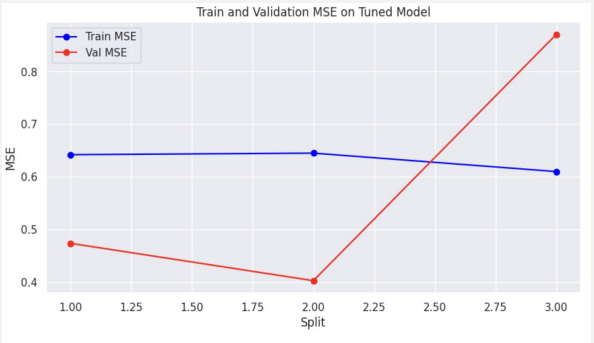
Linear Regression Model: Before and After Tuning

Alpha: 1.0




```
Average Train MSE: 0.6162
Average Validation MSE: 0.6343
```

Alpha: 21.2




```
Average Train MSE: 0.6324
Average Validation MSE: 0.5823
```


We used Keras Tuner to tune the alpha hyperparameter value.

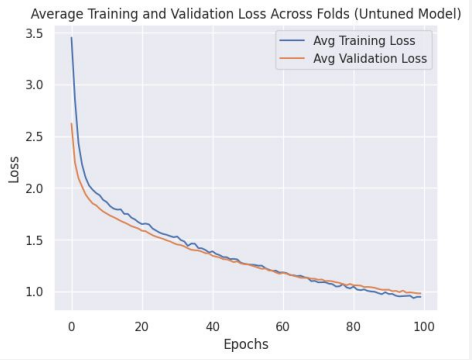
We did this by using a grid search to test 50 numbers evenly distributed on a logarithmic scale between 10^(-5) and 10^5.

We started our initial untuned model with an alpha value of 1.0. After tuning, we landed on an alpha value of 21.2, which decreased our average validation MSE by 0.05 points landing at a final value 0.58

Neural Network

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 24)	0
dense (Dense)	(None, 256)	6,400
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

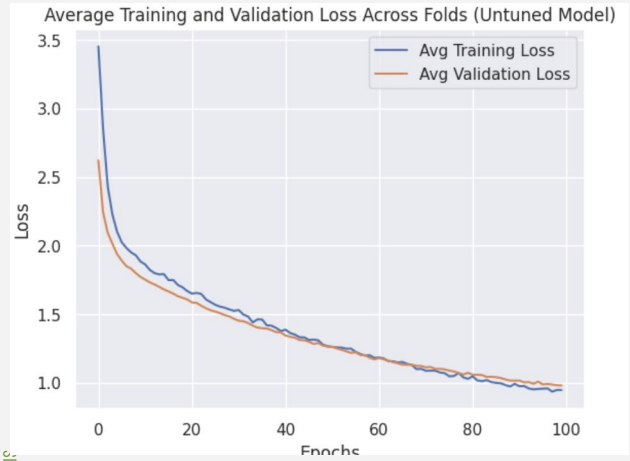
Total params: 47,617 (186.00 KB)
Trainable params: 47,617 (186.00 KB)
Non-trainable params: 0 (0.00 B)

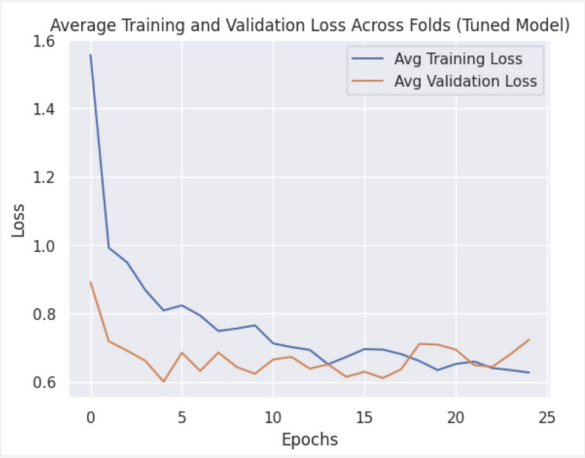


For the neural network, we initially developed a 3 layered model with a decreasing amount of units for each layer. We started off with a decreasing of units so that the model could refine its features through each subsequent layer. Additionally, we created dropout and regularization parameters to prevent overfitting.

The result of this model was a mean validation loss of .7496.

Neural Network: Before and After Tuning





`Mean Validation Loss: 0.7496 ± 0.2820`

`Mean Validation Loss: 0.5550 ± 0.1839`

We used Keras Tuner to tune the number of units, L2 regularization penalty, and dropout rate in each of our three hidden layers. We also tuned the learning rate and implemented early stopping with a patience of 10. We padded the model histories of each fold with their most recent values to ensure we could compare loss across each epoch between the different folds, regardless of when stopping occurred.

After tuning, the average validation loss dropped about 0.2 points. The tuned model completed training after just 25 epochs, which is much lower than the 100 epochs we started with.

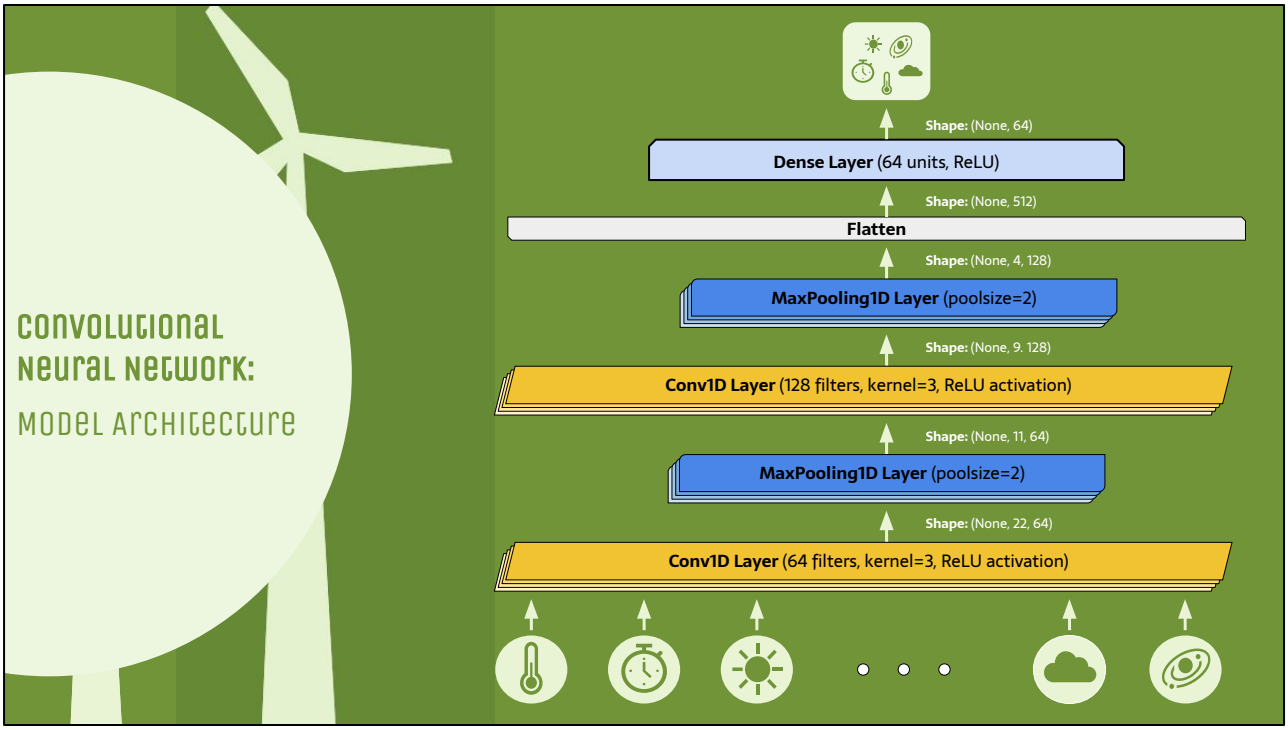
Neural Network: Before and After Tuning

Hyperparameter	Hidden Layer	Untuned Value	Tuned Value
Units	1	256	160
L2 Regularization Penalty	1	0.01	0.0002
Dropout Rate	1	0.2	0.25
Units	2	128	96
L2 Regularization Penalty	2	0.01	0.0002
Dropout Rate	2	0.2	0.3500
Units	3	64	64
L2 Regularization Penalty	3	0.01	0.0007
Dropout Rate	3	0.2	0.25
Learning Rate	-	0.0001	0.008
Epochs	-	100	25

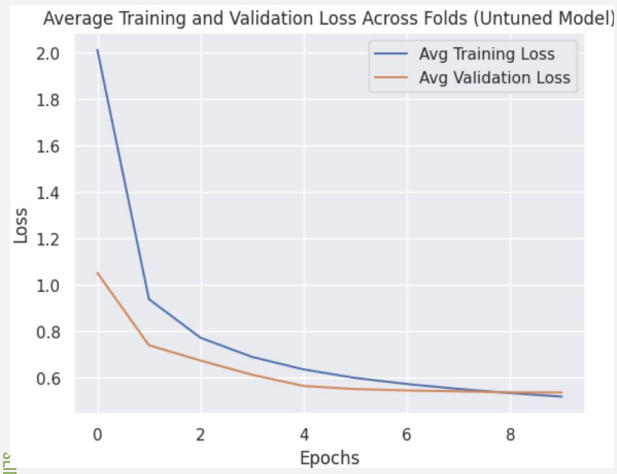
We used Random Search to test different combinations of hyperparameter values, where each hyperparameter value had a defined search space and step size, implemented with log sampling for the learning rate, number of units, and l2 regularization.

Notably, Our tuned learning rate of 0.008 was significantly higher than our original learning rate of 0.0001, which helped our validation loss to converge much faster.

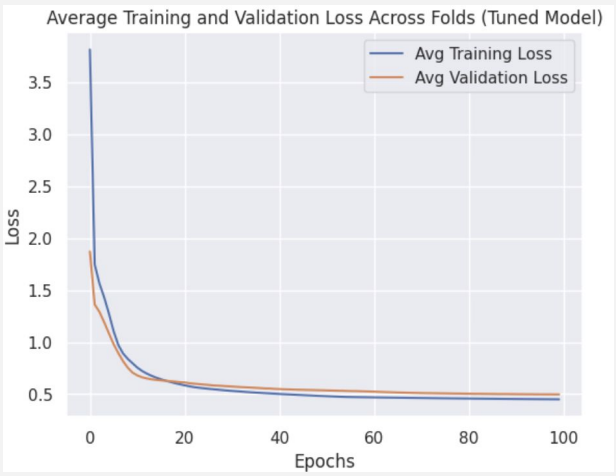
```
{'units_h1': 160, 'l2_h1': 0.00025430520924425877, 'dropout_h1':
0.25, 'units_h2': 96, 'l2_h2': 0.00018980757724363517,
'dropout_h2': 0.35000000000000003, 'units_h3': 64, 'l2_h3':
0.0007085245085911898, 'dropout_h3': 0.25, 'learning_rate':
0.008076262788470068}
```


Convolutional Neural Network:
Model Architecture
Shape: (None, 64)
Dense Layer (64 units, ReLU)
Shape: (None, 512)
Flatten
Shape: (None, 4, 128)
MaxPooling1D Layer (poolsize=2)
Shape: (None, 9. 128)
Conv1D Layer (128 filters, kernel=3, ReLU activation)
Shape: (None, 11, 64)
MaxPooling1D Layer (poolsize=2)
Shape: (None, 22, 64)
Conv1D Layer (64 filters, kernel=3, ReLU activation)

CNN: Before and After Tuning



Mean Validation Loss: 0.5340 ± 0.2528



Mean Validation Loss: 0.4867 ± 0.1695

We used Keras Tuner to tune the number of filters, kernel size, activation function, and pool size of each convolutional layer. We again implemented early stopping with a patience of 10 and padded the training histories to ensure accurate comparison between folds.

After tuning, the average validation loss dropped about 0.5 points. The tuned model trained for the full 100 epochs without implementing early stopping, compared to a fixed 10 epoch pre-tuning.

CNN: Before and After Tuning

Hyperparameter	Hidden Layer	Untuned Value	Tuned Value
Filters	1	64	40
Filters	2	128	10
Kernel Size	1	3	5
Kernel Size	2	3	4
Activation	1	relu	relu
Activation	2	relu	sigmoid
Pool Size	1	2	4
Pool Size	2	2	2
Epochs	-	10	100

We used RandomSearch to test different combinations of hyperparameter values, where each hyperparameter value had a defined search space and step size. Activation options included relu, sigmoid, and tanh.

Most notable change included reducing the number of filters helped reduce overfitting

Evaluation

CNN

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 2.6148
Test Loss: 2.4114246368408203
```

*Neural Network

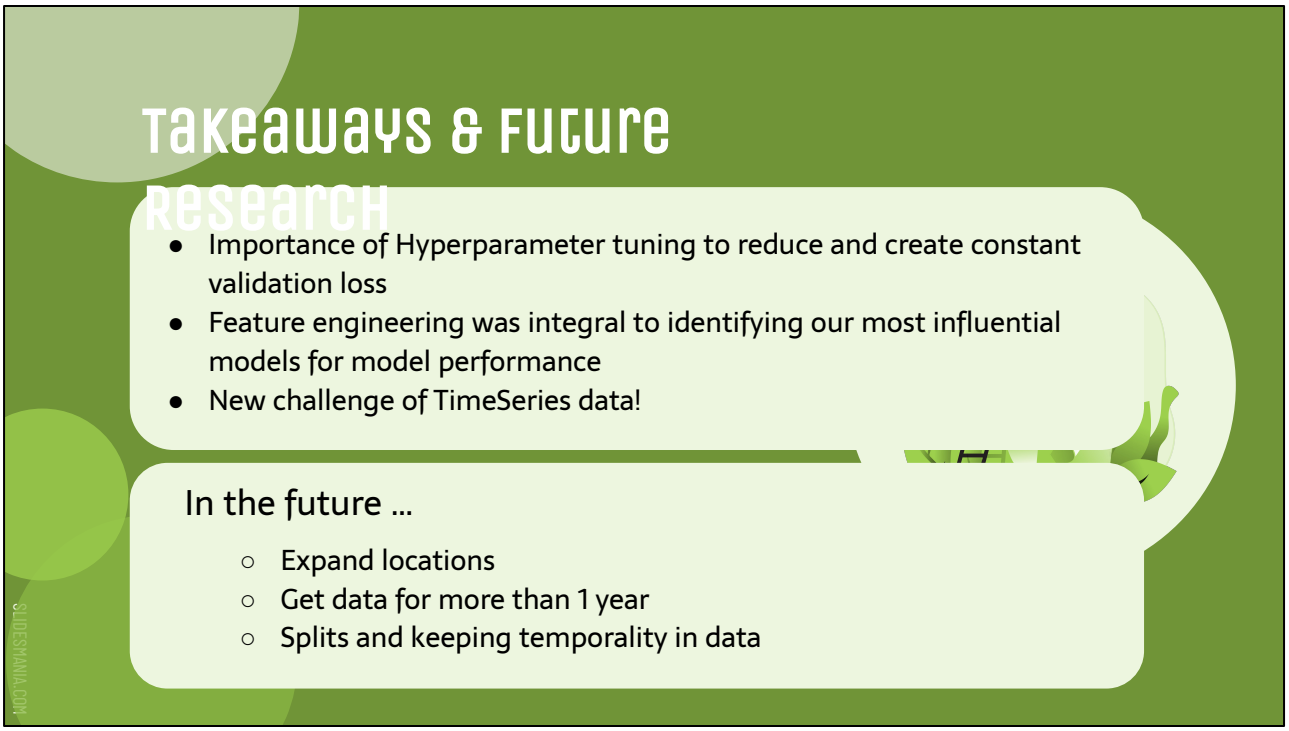
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 1.5572 - mean_squared_error: 1.4956
Test Loss: [1.2756891250610352, 1.2140346765518188]
```


Evaluated model on the entire datasets
Here loss refers to the mean squared error
We think both our model generalize well - training and test loss are similar, with less than 0.1 difference between the two
However the neural network outperformed the CNN

PERFORMANCE COMPARISON

	Baseline Model	Linear Regression	Neural Network	CNN
Untuned Validation MSE	2.31	0.63	0.75	0.53
Tuned Validation MSE	N/A	0.58	0.55	0.49

Hunter



Hunter
Main takeaways: Importance of hyperparamater tuning and feature enginnering
For the test/train data, test data is last week of every month, skipping/jumps of data could be breaking temporality

THANKS!

Team contributions

- **Mina Baghai:** data aggregation, feature engineering, baseline model, EDA
- **Tim Leong:** EDA, feature engineering, neural network model
- **Sonia Chakraborty:** Linear regression model, presentation slides
- **Hunter Tonn:** CNN model, presentation slides
- **Jordan Lawhon:** Model tuning, model evaluation, created GitHub repo