# DL in Applied Mathematics

Lecture 6: **Forecast Time Series Data**

**Marat Nurtas**
PhD in Mathematical and Computer Modeling
Department of Mathematical and Computer Modeling
International Information Technology University, Almaty, Kazakhstan

In this lecture delves into enhancing the process of forecasting daily energy consumption levels by transforming a time series dataset into a tabular format using open-source libraries.

We explore the application of a popular multiclass classification model and leverage AutoML with Cleanlab Studio to significantly boost our out-of-sample accuracy.

The key takeaway from this lecture is that we can use more general methods to model a time series data set into a tabular structure, and even find improvements in trying to forecast those time series data.

## Take a Snapshot

At a high level we will:

- Establish a baseline accuracy by fitting a Prophet forecasting model on our time series data
- Convert our time series data into a tabular format by using open-source featurization libraries and then will show that can outperform our Prophet model with a standard multiclass classification (Gradient Boosting) approach by a **67% reduction in prediction error** (increase by 38% raw percentage points in out-of-sample accuracy).

- Use an AutoML solution for multiclass classification **resulted in a 42% reduction in prediction error** (increase by 8% in raw percentage points in out-of-sample accuracy) compared to our Gradient Boosting model and **resulted in a 81% reduction in prediction error** (increase by 46% in raw percentage points in out-of-sample accuracy) compared to our Prophet forecasting model.

## Examine the Data

The data represents PJM hourly energy consumption (in megawatts) on an hourly basis. PJM Interconnection LLC (PJM) is a regional transmission organization (RTO) in the United States.

It is part of the Eastern Interconnection grid operating an electric transmission system serving many states.

The data includes one datetime column (object type), and the Megawatt Energy Consumption (float64) type) column we are trying to forecast as a discrete variable (corresponding to the quartile of hourly energy consumption levels).

Our aim is to train a time series forecasting model to be able to forecast the tomorrow’s daily energy consumption level falling into 1 of 4 levels: low , below average , above average or high (these levels were determined based on quartiles of the overall daily consumption distribution).

We first demonstrate how to apply time-series forecasting methods like Prophet to this problem, but these are restricted to certain types of ML models suitable for time-series data.

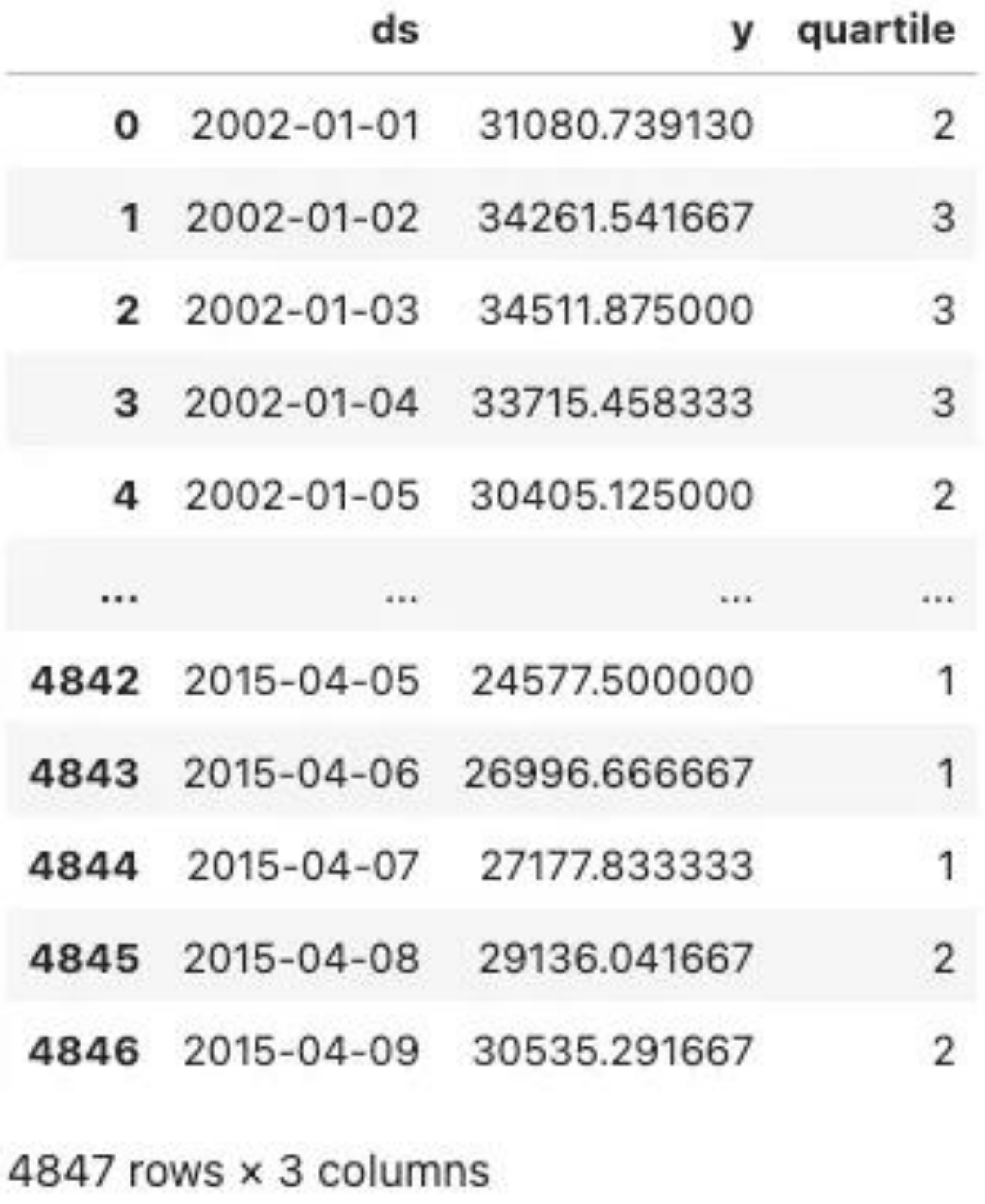
Next we demonstrate how to reframe this problem into a standard multiclass classification problem that we can apply any machine learning model to, and show how we can obtain superior forecasts by using powerful supervised ML.

We first convert this data into a average energy consumption at a daily level and rename the columns to the format that the Prophet forecasting model expects.

These real-valued daily energy consumption levels are converted into quartiles, which is the value we are trying to predict.

Our training data is shown below along with the quartile each daily energy consumption level falls into.

The quartiles are computed using training data to prevent data leakage.

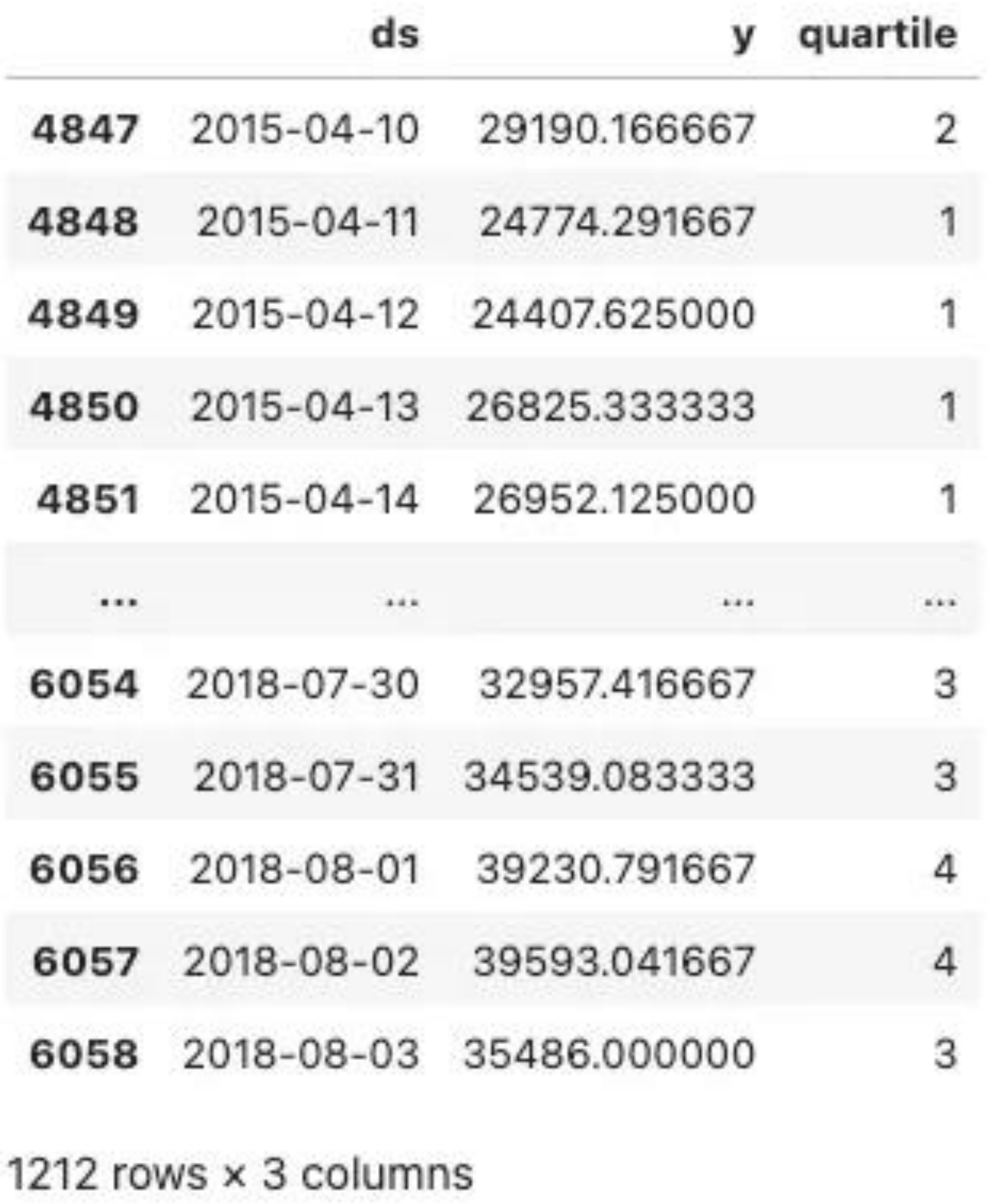


| | ds | y | quartile |
|---|---|---|---|
| 0 | 2002-01-01 | 31080.739130 | 2 |
| 1 | 2002-01-02 | 34261.541667 | 3 |
| 2 | 2002-01-03 | 34511.875000 | 3 |
| 3 | 2002-01-04 | 33715.458333 | 3 |
| 4 | 2002-01-05 | 30405.125000 | 2 |
| ... | ... | ... | ... |
| 4842 | 2015-04-05 | 24577.500000 | 1 |
| 4843 | 2015-04-06 | 26996.666667 | 1 |
| 4844 | 2015-04-07 | 27177.833333 | 1 |
| 4845 | 2015-04-08 | 29136.041667 | 2 |
| 4846 | 2015-04-09 | 30535.291667 | 2 |

4847 rows × 3 columns

We now show the training data we use to fit our prediction model.

Training data with quartile of daily energy consumption level included



| | ds | y | quartile |
|---|---|---|---|
| **4847** | 2015-04-10 | 29190.166667 | 2 |
| **4848** | 2015-04-11 | 24774.291667 | 1 |
| **4849** | 2015-04-12 | 24407.625000 | 1 |
| **4850** | 2015-04-13 | 26825.333333 | 1 |
| **4851** | 2015-04-14 | 26952.125000 | 1 |
| **...** | ... | ... | ... |
| **6054** | 2018-07-30 | 32957.416667 | 3 |
| **6055** | 2018-07-31 | 34539.083333 | 3 |
| **6056** | 2018-08-01 | 39230.791667 | 4 |
| **6057** | 2018-08-02 | 39593.041667 | 4 |
| **6058** | 2018-08-03 | 35486.000000 | 3 |

1212 rows × 3 columns

We then show the test data against which we evaluate our prediction results.

Test data with quartile of daily energy consumption level included

## Train and Evaluate Prophet Forecasting Model

As seen in the images above, we will use a date cutoff of 2015-04-09 to end the range of our training data and start our test data at 2015-04-10

We compute quartile thresholds of our daily energy consumption using ONLY training data.

This avoids data leakage - using out-of-sample data that is available only in the future.

Next, we will forecast the daily PJME energy consumption level (in MW) for the duration of our test data and represent the forecasted values as a discrete variable.

This variable represents which quartile the daily energy consumption level falls into, represented categorically as 1 (low), 2 (below average), 3 (above average), or 4 (high). For evaluation, we are going to use the accuracy_score function from scikit-learn to evaluate the performance of our models.

Since we are formulating the problem this way, we are able to evaluate our model's next-day forecasts (and compare future models) using classification accuracy.

```python
import numpy as np
from prophet import Prophet
from sklearn.metrics import accuracy_score

# Initialize model and train it on training data
model = Prophet()
model.fit(train_df)


# Create a dataframe for future predictions covering the test period
future = model.make_future_dataframe(periods=len(test_df), freq='D')
forecast = model.predict(future)
```

```
# Categorize forecasted daily values into quartiles based on the
thresholds
forecast['quartile'] = pd.cut(forecast['yhat'], bins = [-np.inf] +
list(quartiles) + [np.inf], labels=[1, 2, 3, 4])

# Extract the forecasted quartiles for the test period
forecasted_quartiles = forecast.iloc[-len(test_df):]['quartile'].astype(int)


# Categorize actual daily values in the test set into quartiles
test_df['quartile'] = pd.cut(test_df['y'], bins=[-np.inf] + list(quartiles) +
[np.inf], labels=[1, 2, 3, 4])
actual_test_quartiles = test_df['quartile'].astype(int)
```

```
# Calculate the evaluation metrics
accuracy = accuracy_score(actual_test_quartiles, forecasted_quartiles)

# Print the evaluation metrics
print(f'Accuracy: {accuracy:.4f}')
>>> 0.4249
```

The out-of-sample accuracy is quite poor at 43%.

By modelling our time series this way, we limit ourselves to only use time series forecasting models (a limited subset of possible ML models).

Once the time-series has been transformed into a standard tabular dataset, we’re able to employ any supervised ML model for forecasting this daily energy consumption data.

## Convert time series data to tabular data through featurization

Now we convert the time series data into a tabular format and featurize the data using the open source libraries sktime, tsfresh, and tsfel.

By employing libraries like these, we can extract a wide array of features that capture underlying patterns and characteristics of the time series data.

This includes statistical, temporal, and possibly spectral features, which provide a comprehensive snapshot of the data's behavior over time.

By breaking down time series into individual features, it becomes easier to understand how different aspects of the data influence the target variable.

TSFreshFeatureExtractor is a feature extraction tool from the sktime library that leverages the capabilities of tsfresh to extract relevant features from time series data.

tsfresh is designed to automatically calculate a vast number of time series characteristics, which can be highly beneficial for understanding complex temporal dynamics.

For our use case, we make use of the minimal and essential set of features from our TSFreshFeatureExtractor to featurize our data.

tsfel, or Time Series Feature Extraction Library, offers a comprehensive suite of tools for extracting features from time series data.

We make use of a predefined config that allows for a rich set of features (e.g., statistical, temporal, spectral) to be constructed from the energy consumption time series data, capturing a wide range of characteristics that might be relevant for our classification task.

```
import tsfel
from sktime.transformations.panel.tsfresh import
TSFreshFeatureExtractor

# Define tsfresh feature extractor
tsfresh_trafo =
TSFreshFeatureExtractor(default_fc_parameters="minimal")

# Transform the training data using the feature extractor
X_train_transformed = tsfresh_trafo.fit_transform(X_train)
```

```
# Transform the test data using the same feature extractor
X_test_transformed = tsfresh_trafo.transform(X_test)

# Retrieves a pre-defined feature configuration file to extract all
available features
cfg = tsfel.get_features_by_domain()

# Function to compute tsfel features per day
def compute_features(group):
    # TSFEL expects a DataFrame with the data in columns, so we
transpose the input group
    features = tsfel.time_series_features_extractor(cfg, group, fs=1,
verbose=0)
    return features
```

```
# Group by the 'day' level of the index and apply the feature
computation
train_features_per_day =
X_train.groupby(level='Date').apply(compute_features).reset_index(dr
op=True)
test_features_per_day =
X_test.groupby(level='Date').apply(compute_features).reset_index(dro
p=True)

# Combine each featurization into a set of combined features for our
train/test data
train_combined_df = pd.concat([X_train_transformed,
train_features_per_day], axis=1)
test_combined_df = pd.concat([X_test_transformed,
test_features_per_day], axis=1)
```

Next, we clean our dataset by removing features that showed a high correlation (above 0.8) with our target variable — average daily energy consumption levels — and those with null correlations.

High correlation features can lead to overfitting, where the model performs well on training data but poorly on unseen data.

Null-correlated features, on the other hand, provide no value as they lack a definable relationship with the target.

By excluding these features, we aim to improve model generalizability and ensure that our predictions are based on a balanced and meaningful set of data inputs.

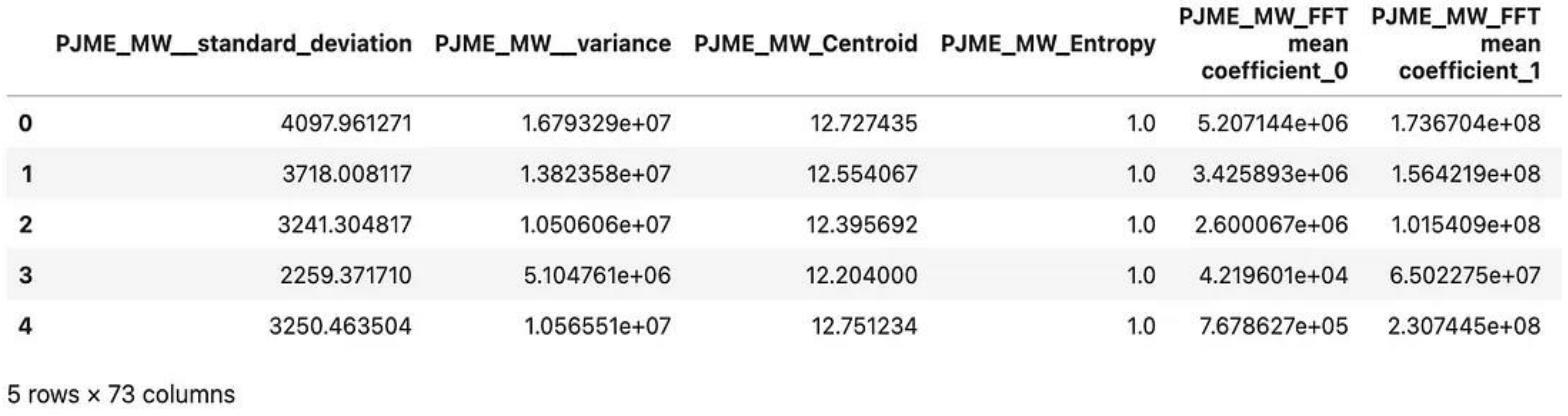
```
# Filter out features that are highly correlated with our target variable
column_of_interest = "PJME_MW__mean"
train_corr_matrix = train_combined_df.corr()
train_corr_with_interest = train_corr_matrix[column_of_interest]
null_corrs = pd.Series(train_corr_with_interest.isnull())
false_features = null_corrs[null_corrs].index.tolist()

columns_to_exclude =
list(set(train_corr_with_interest[abs(train_corr_with_interest) >
0.8].index.tolist() + false_features))
columns_to_exclude.remove(column_of_interest)
```

```
# Filtered DataFrame excluding columns with high correlation to the column of interest
X_train_transformed = train_combined_df.drop(columns=columns_to_exclude)
X_test_transformed = test_combined_df.drop(columns=columns_to_exclude)
```

If we look at the first several rows of the training data now, this is a snapshot of what it looks like.

We now have 73 features that were added from the time series featurization libraries we used.

The label we are going to predict based on these features is the next day's energy consumption level.



| | PJME_MW__standard_deviation | PJME_MW__variance | PJME_MW_Centroid | PJME_MW_Entropy | PJME_MW_FFT mean coefficient_0 | PJME_MW_FFT mean coefficient_1 |
|---|---|---|---|---|---|---|
| **0** | 4097.961271 | 1.679329e+07 | 12.727435 | 1.0 | 5.207144e+06 | 1.736704e+08 |
| **1** | 3718.008117 | 1.382358e+07 | 12.554067 | 1.0 | 3.425893e+06 | 1.564219e+08 |
| **2** | 3241.304817 | 1.050606e+07 | 12.395692 | 1.0 | 2.600067e+06 | 1.015409e+08 |
| **3** | 2259.371710 | 5.104761e+06 | 12.204000 | 1.0 | 4.219601e+04 | 6.502275e+07 |
| **4** | 3250.463504 | 1.056551e+07 | 12.751234 | 1.0 | 7.678627e+05 | 2.307445e+08 |

5 rows × 73 columns

First 5 rows of training data which is newly featurized and in a tabular format

It’s important to note that we used a best practice of applying the featurization process separately for training and test data to avoid data leakage (and the held-out test data are our most recent observations).

Also, we compute our discrete quartile value (using the quartiles we originally defined) using the following code to obtain our train/test energy labels, which is what our y_labels are.

```
# Define a function to classify each value into a quartile
def classify_into_quartile(value):
    if value < quartiles[0]:
        return 1
    elif value < quartiles[1]:
        return 2
    elif value < quartiles[2]:
        return 3
    else:
        return 4
```

```
y_train =
X_train_transformed["PJME_MW__mean"].rename("daily_energy_level")
X_train_transformed.drop("PJME_MW__mean", inplace=True, axis=1)

y_test =
X_test_transformed["PJME_MW__mean"].rename("daily_energy_level")
X_test_transformed.drop("PJME_MW__mean", inplace=True, axis=1)

energy_levels_train = y_train.apply(classify_into_quartile)
energy_levels_test = y_test.apply(classify_into_quartile)
```

## Train and Evaluate GradientBoostingClassifier Model on featurized tabular data

Using our featurized tabular dataset, we can apply any supervised ML model to predict future energy consumption levels.

Here we'll use a Gradient Boosting Classifier (GBC) model, the weapon of choice for most data scientists operating on tabular data.

Our GBC model is instantiated from the sklearn.ensemble module and configured with specific hyperparameters to optimize its performance and avoid overfitting.

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(
	n_estimators=150,
	learning_rate=0.1,
	max_depth=4,
	min_samples_leaf=20,
	max_features='sqrt',
	subsample=0.8,
	random_state=42
)

gbc.fit(X_train_transformed, energy_levels_train)
```

```
y_pred_gbc = gbc.predict(X_test_transformed)
gbc_accuracy = accuracy_score(energy_levels_test, y_pred_gbc)
print(f'Accuracy: {gbc_accuracy:.4f}')
>>> 0.8075
```

The out-of-sample accuracy of 81% is considerably better than our prior Prophet model results.

## Using AutoML to streamline things

Now that we’ve seen how to featurize the time-series problem and the benefits of applying powerful ML models like Gradient Boosting, a natural question emerges:

Which supervised ML model should we apply?

Of course, we could experiment with many models, tune their hyperparameters, and ensemble them together.

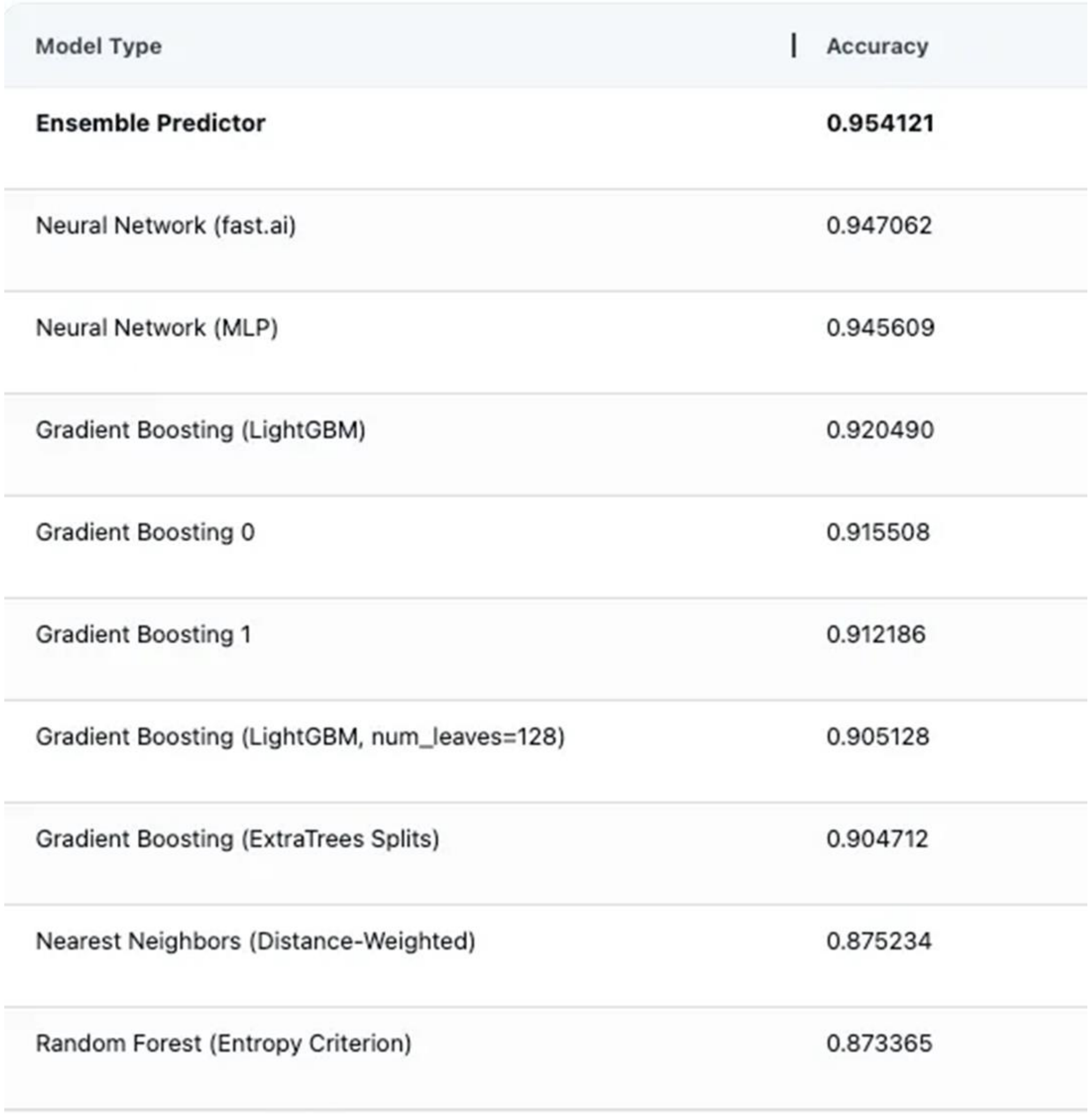
An easier solution is to let AutoML handle all of this for us.

Here we’ll use a simple AutoML solution provided in Cleanlab Studio, which involves zero configuration.

We just provide our tabular dataset, and the platform automatically trains many types of supervised ML models (including Gradient Boosting among others), tunes their hyperparameters, and determines which models are best to combine into a single predictor.

Here’s all the code needed to train and deploy an AutoML supervised classifier:

```python
from cleanlab_studio import Studio

studio = Studio()
studio.create_project(
	dataset_id=energy_forecasting_dataset,
	project_name="ENERGY-LEVEL-FORECASTING",
	modality="tabular",
	task_type="multi-class",
	model_type="regular",
	label_column="daily_energy_level",
)

model = studio.get_model(energy_forecasting_model)
y_pred_automl = model.predict(test_data, return_pred_proba=True)
```

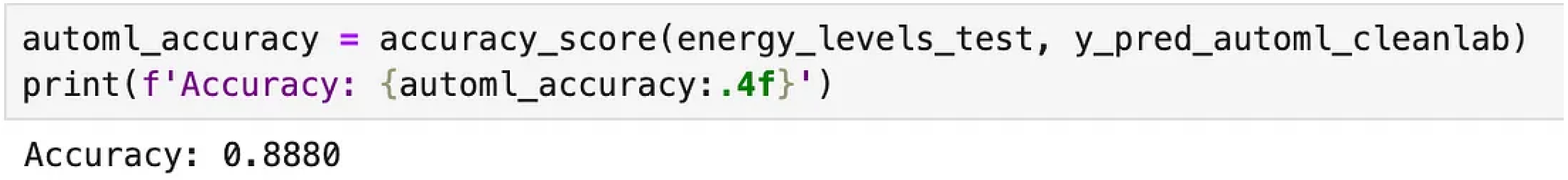


| Model Type | Accuracy |
|---|---|
| **Ensemble Predictor** | **0.954121** |
| Neural Network (fast.ai) | 0.947062 |
| Neural Network (MLP) | 0.945609 |
| Gradient Boosting (LightGBM) | 0.920490 |
| Gradient Boosting 0 | 0.915508 |
| Gradient Boosting 1 | 0.912186 |
| Gradient Boosting (LightGBM, num_leaves=128) | 0.905128 |
| Gradient Boosting (ExtraTrees Splits) | 0.904712 |
| Nearest Neighbors (Distance-Weighted) | 0.875234 |
| Random Forest (Entropy Criterion) | 0.873365 |

Below we can see model evaluation estimates in the AutoML platform, showing all of the different types of ML models that were automatically fit and evaluated (including multiple Gradient Boosting models), as well as an ensemble predictor constructed by optimally combining their predictions.

AutoML results across different types of models used

After running inference on our test data to obtain the next-day energy consumption level predictions, we see the test accuracy is 89%, a 8% raw percentage points improvement compared to our previous Gradient Boosting approach.



```
automl_accuracy = accuracy_score(energy_levels_test, y_pred_automl_cleanlab)
print(f'Accuracy: {automl_accuracy:.4f}')
```

```
Accuracy: 0.8880
```

AutoML test accuracy on our daily energy consumption level data

## Conclusion

For our PJM daily energy consumption data, we found that transforming the data into a tabular format and featurizing it achieved a **67% reduction in prediction error** (increase by 38% in raw percentage points in out-of-sample accuracy) compared to our baseline accuracy established with our Prophet forecasting model.

We also tried an easy AutoML approach for multiclass classification, which **resulted in a 42% reduction in prediction error** (increase by 8% in raw percentage points in out-of-sample accuracy) compared to our Gradient Boosting model and **resulted in a 81% reduction in prediction error** (increase by 46% in raw percentage points in out-of-sample accuracy) compared to our Prophet forecasting model.

By taking approaches like those illustrated above to model a time series dataset beyond the constrained approach of only considering forecasting methods, we can apply more general supervised ML techniques and achieve better results for certain types of forecasting problems.

Thank you for attention!