Forecasting Brent Crude Oil Price and OECD Oil Demand

Final Project Report Template

Revision 3 for 5.9.25 Assignment 9

Owen Retzlaff, Jolea Wallisch

ITSCM 180: Introduction to Programming for Business Applications

Dr. Tim Carone

May 7, 2025

Executive Summary

Detailed Analysis

1. Introduction

Brent crude–a global benchmark–serves as our price target, while refined-products consumption in the Organization for Economic Cooperation and Development (OECD) member countries reflects core demand trends in developed markets. By combining time-series and machine-learning approaches, we seek both robust average forecasts (via Prophet) and flexible, lag-driven predictions (via XGBoost).

2. Data & Preprocessing

2.1 Data Sources

Monthly Brent crude oil prices and OECD demand data was retrieved from the U.S. Energy Information Administration (EIA) Open-Data Application Programming Interface (API) from January 1990 through April 2025.

2.2 Cleaning & Feature Construction

The raw JavaScript Object Notation (JSON) response data from the EIA Open-Data API is parsed into pandas DataFrames, dates are coerced to datetime, and values are converted to numeric, with any invalid rows dropped. Each series is then resampled to month-end frequency and any small gaps are filled by linear interpolation to ensure a continuous time index.

3. Methodology

3.1 Exploratory Data Analysis & Feature Selection

To explore the relationship between price and demand, the two month-end series are merged on their data index and standardized via z-score scaling. A correlation heatmap reveals a modest positive correlation of approximately 0.21, and a pairplot of the scaled series shows their joint distributions. We then created lagged features at 1, 3, 6, and 12 months for both series and trained a preliminary XGBoost regressor on Brent price. The resulting feature importance chart confirms that the one-month lag of Brent is the strongest predictor, with longer lags and demand lags contributing more modestly.

3.2 Predictive Modeling

Each series is modeled with two approaches. Prophet fits an additive model with automatic trend and yearly seasonality. XGBoost uses the engineered lag features as inputs to a gradient-boosted tree regressor (objective = reg:squarederror). We train on all but the final twelve months of data, reserving those last twelve months for a hold-out test.

3.3 Validation & Performance Metrics

Each model produced point forecasts for the twelve-month hold-out. Performance was measured by root mean squared error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). The best model per target was selected by lowest RMSE.

4. Results

4.1 Forecast Plots

Forecasts and actuals for Brent crude oil price and OECD oil demand appear in four time-series panels (Appendix A: Figures 1-4) and are saved under /reports. Additional plots show the correlation between price and demand (Figures 6-7) and the quality of the XGBoost model's predictors (Figure 5).

4.2 Performance Summary

Model accuracy on the hold-out year is measured by RMSE, MAE, and MAPE.

Series	Model	Test RMSE	Test MAE	Test MAPE
Brent Price (USD)	XGBoost	5.5	4.7	5.9
	Prophet	11.7	10.7	13.3
OECD Demand (TBPD)	XGBoost	645	554	1.2
	Prophet	1437	1288	2

We further assess stability using five-fold time-series cross-validation, which retrains on expanding windows and forecasts the next twelve months in each fold. Cross-validation confirms XGBoost consistently outperforms Prophet on average, though error variability increases around major market disruptions (such as the COVID-19 pandemic or 2008 recession).

5. Discussion

Visualizations of actual vs. forecasted values over the past twenty years highlight how XGBoost captures short-term fluctuations more closely, while Prophet provides a smoother trend and uncertainty band. The cross-validation results underscore that model performance can vary substantially across historical regimes, suggesting the benefit of combining methods. Perhaps, XGBoost for precise short-term decisions and Prophet for robust, seasonal scenario planning.

6. Conclusions & Recommendations

We recommend a dual-model strategy to a reliable forecasting model for oil pricing and demand: deploy XGBoost for operational, short-horizon forecasts and use Prophet as a stable baseline for strategic planning. To further improve accuracy of predictions, future work should incorporate additional explanatory variables (e.g. GDP, inventory levels, rig counts), experiment with SARIMAX and ensemble stacking, and extend the demand series via alternative data sources, such as BP or OECD CSV exports.

Appendices

A. Forecast Plots

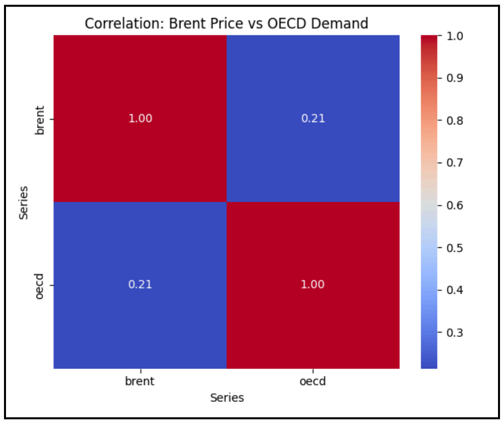

Figure 1: This heatmap shows the Pearson correlation coefficient between monthly Brent crude oil prices and OECD refined petroleum consumption over the sample period. The single cell in the matrix is annotated with its correlation value (≈ 0.21), indicating a modest positive relationship: higher OECD demand tends to accompany higher Brent prices, but the strength of the linkage is relatively weak. The color scale—from deep blue (–1) through white (0) to deep red (+1)—makes it easy to see that the two series are only lightly correlated.

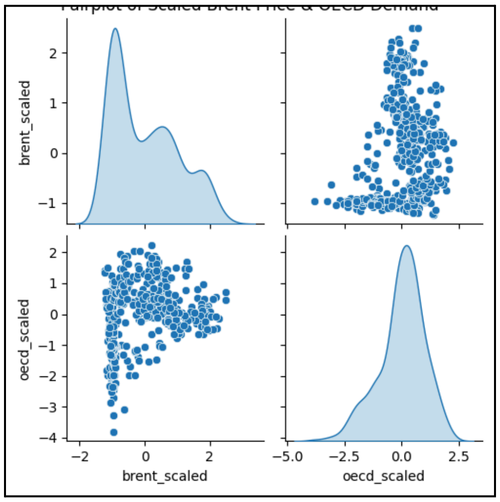

Figure 2: This pairplot displays the joint distribution and marginal density estimates for the z-score–normalized Brent price and OECD demand series. On the diagonal, kernel density estimates reveal that both series exhibit right-skewed distributions even after scaling. The off-diagonal scatterplot shows a loose cloud of points, reaffirming the modest positive relationship seen in the heatmap. This visualization helps confirm that the two variables share some co-movement but also display distinct variability patterns.

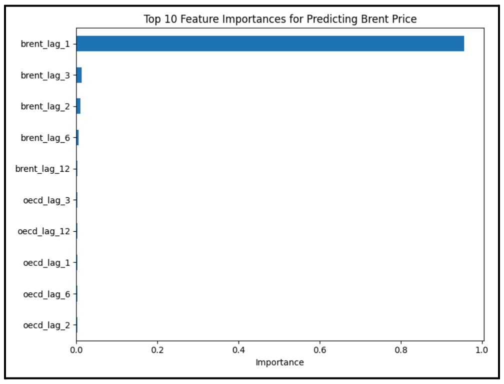

Figure 3: This horizontal bar chart ranks the ten most important lagged predictors for forecasting Brent price, as determined by an XGBoost regressor. The one-month lag of Brent is the most influential feature by a wide margin, followed by the three- and six-month lags. OECD demand lags appear lower on the list, indicating they contribute less to explaining short-term price movements.

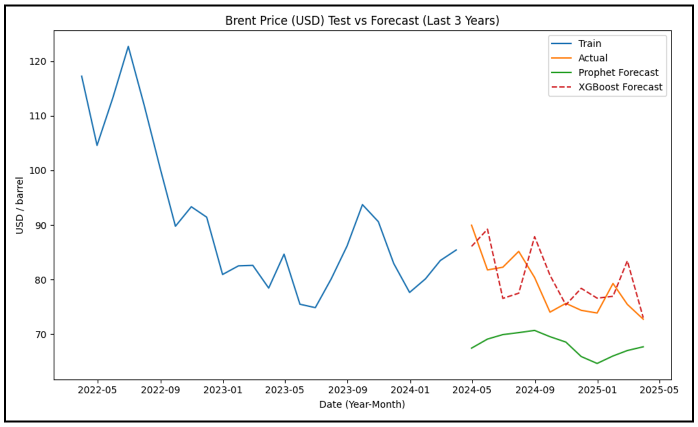

Figure 4: This time-series line chart overlays the historical Brent price (train and actual test data) with the one-year hold-out forecasts from both Prophet (solid line) and XGBoost (dashed line). This shows that XGBoost tracks sharp reversals more closely, whereas Prophet produces a smoother trend that under-reacts to sudden price jumps.

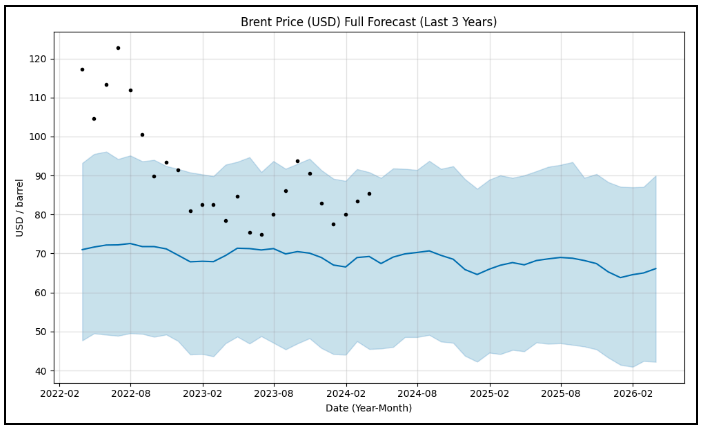

Figure 5: Here the Prophet model's full forecast (including historical fit and 24-month projection) is plotted over the last 20 years. The shaded uncertainty band around the forecast illustrates Prophet's confidence intervals, widening into the future to reflect increasing forecast uncertainty. The chart emphasizes seasonal cycles and long-term trends, making it useful for scenario planning even if it misses short-lived spikes.

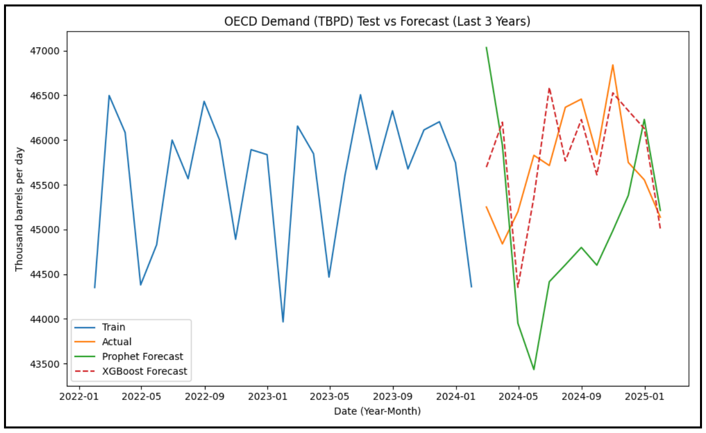

Figure 6: Analogous to Figure 4, this plot shows historical OECD refined-products consumption alongside one-year hold-out forecasts from Prophet and XGBoost, zoomed to the most recent 3 years. The dashed XGBoost line more accurately follows the small seasonal dips and rebounds in demand, while Prophet's smoother line captures the overall upward trend but lags during demand shocks.

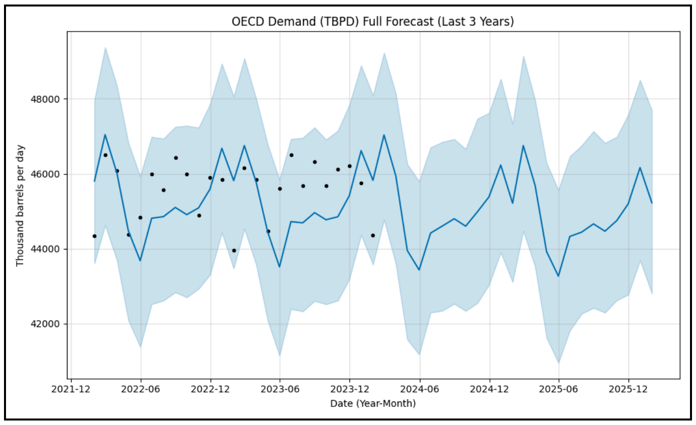

Figure 7: Similar to what is pictured in Figure 5, this chart presents the Prophet model's complete fit and 24-month demand projection for OECD consumption over the last 3 years. The forecast displays clear seasonal cycles—summer peaks and winter troughs.

B. Term Definitions

Additive model: A forecasting model in which components (trend, seasonality) sum to form the prediction.

Hold-out test: Reserving the last portion of data (12 months) for unbiased evaluation.

Prophet: A library for fitting additive time-series models with automated trend and seasonality.

XGBoost: A gradient-boosted tree algorithm, here trained on lagged values for regression.

Lagged feature: A prior value of the series (e.g. one-month-old price) used as an input.

Time-series CV: Cross-validation that respects time order by training on past windows and testing on subsequent periods.

RMSE / MAE / MAPE: Standard error measures: root-mean-square error, mean absolute error, and mean absolute percentage error.

C. Code Snippets

Overview:

Each of the following snippets illustrates one of the critical steps in our pipeline—data ingestion, preprocessing, feature engineering, validation, and visualization—and together they document the core logic of the forecasting program.

Snippet 1:

```
def fetch_eia_series(series_id, api_key):
    url = f'https://api.eia.gov/v2/seriesid/{series_id}?api_key={api_key}'
    resp = requests.get(url); resp.raise_for_status()
    data = resp.json().get('response')['data']
    df = pd.DataFrame(data).rename(columns={'period':'date','value':'value'})
    df['date']  = pd.to_datetime(df['date'], errors='coerce')
    df['value'] = pd.to_numeric(df['value'], errors='coerce')
    df.dropna(subset=['date','value'], inplace=True)
    df.set_index('date', inplace=True)
    return df[['value']]
```

This function encapsulates the entire workflow of pulling a monthly time series from the EIA API v2, handling HTTP errors, parsing the returned JSON, and coercing the raw "period" and "value" fields into a clean pandas DataFrame. It converts the "period" strings into datetime objects, forces "value" into numeric (dropping any malformed entries), sets the datetime as the index, and returns a single-column DataFrame. By centralizing all of these steps, we ensure that any series fetched through this function is immediately ready for resampling and modeling.

Snippet 2:

```
def prepare_monthly_df(df):
    monthly = df.resample('ME').mean().reset_index()
    monthly.rename(columns={'date':'ds','value':'y'}, inplace=True)
    return monthly[['ds','y']]
```

Prophet expects its input DataFrame to have two columns named ds (datestamp) and y (value), with a uniform time frequency. This helper function takes the raw daily or irregularly-spaced series returned by Snippet 1, resamples it to the last calendar day of each month ('ME'), computes the average value within each month, and renames the columns appropriately. The result is a tidy month-end DataFrame that plugs directly into Prophet's fit and predict methods.

Snippet 3:

```
df_xgb = df.set_index('ds').copy()
for lag in (1,2,3,6,12):
    df_xgb[f'lag_{lag}'] = df_xgb['y'].shift(lag)
df_xgb.dropna(inplace=True)
X = df_xgb.drop('y', axis=1)
y = df_xgb['y']
```

Machine-learning models like XGBoost require a matrix of predictors. For time-series forecasting, a common approach is to create "lagged" versions of the target variable as features. This snippet takes the prepared ds/y DataFrame, shifts the y column by 1, 2, 3, 6, and 12 months to create new columns, and then drops any rows with missing values (which occur at the start of the series). The resulting X matrix contains the lagged predictors, and y is the value to forecast. This simple feature-engineering step lets XGBoost learn patterns in how past values drive future outcomes.

Snippet 4:

```
from sklearn.model_selection import TimeSeriesSplit

tscv = TimeSeriesSplit(n_splits=5)
for fold, (train_idx, test_idx) in enumerate(tscv.split(X), start=1):
    X_tr, X_te = X.iloc[train_idx], X.iloc[test_idx]
    y_tr, y_te = y.iloc[train_idx], y.iloc[test_idx]
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    # compute RMSE/MAE/MAPE...
```

Standard k-fold cross-validation randomly shuffles data, which is inappropriate for time series. Instead, we use TimeSeriesSplit, which creates sequential train/test splits that respect chronological order. In each of five folds, the training set is an expanding window of the earliest observations, and the test set is the immediately following slice. We fit the model on each training window, predict the next year, and compute error metrics. Aggregating these five results gives a robust sense of model stability over different historical periods.

Snippet 5:

```
plt.figure(figsize=(10,6))
plt.plot(train['ds'], train['y'],    label='Train')
plt.plot(test['ds'],  test['y'],     label='Actual')
plt.plot(test['ds'],  pred_p,        label='Prophet Forecast')
plt.plot(test['ds'],  pred_xgb,      label='XGBoost Forecast', linestyle='--')
plt.xlabel('Date (Month-Year)')
plt.ylabel(series_name)
plt.title(f"{series_name} Test vs Forecast")
plt.legend()
plt.tight_layout()
plt.show()
```

Clear visualization is key to interpreting model performance. This snippet overlays the historical training data, the actual values in the hold-out period, and the forecasts from both Prophet and XGBoost on a single time-series plot. Descriptive axis labels ("Date (Month–Year)" and the series name with units) and a legend make the chart self-explanatory. Using solid and dashed line styles differentiates the two model forecasts, and tight_layout() ensures no clipping of labels.

D. Code Repository

All source code, data files, figures, and this report are publicly available on GitHub: https://github.com/retzlaffo13/oil_forecasts