Predictions of Thermodynamic Steam Table

By:

Mohammed Ahmed

NYU Tandon School of Engineering

Quad Chart

Project Overview

İt focuses to predict thermodynamic properties, at given temperatures and pressures This extends the applicability of thermodynamic steam tables to the supercritical region were the distinction between liquid and gas phase do not exist. We utilized various machine learning regression techniques to overcome these challenges.

Technical Approach

We collected steam tables data that covered a wide range of temperatures and pressures. The data was then separated for training and testing. We chose multiple regression models and recorded their Mean Square Error (MSE) per target varible. Based on which model had the lowest average MSE we then used that model to predict the thermodynamic properties of steam at extreme conditions.

Results and Key Findings

For nonlinear relationships, **Polynomial Regression and Neural Network Regressor** indicated good performance.

Linear Regression was less effective for handling extreme conditions

Data sparsity is managed significantly by **RandomForestRegressor**

Performance Metrics:

Best Regressor (Lowest Average MSE):
Random Forest (1.39e+03)
Worst Regressor (Highest Average MSE):
Support Vector Regression (6.01e+05)
Specific Volume:
Best predicted by Gradient Boosting (MSE: 5.23e-07).
Enthalpy: Best predicted by XGBoost (MSE: 2.71e+03).
Viscosity: Best predicted by Random Forest (MSE: 1.44e+02).

Impact and Future Work

With the integration of machine learning techniques and models, we can develop new methods combined with traditional physics-based techniques. This will greatly simplify work in fields that heavily rely on steam tables while enhancing accuracy and efficiency.

The successful implementation of this project has the potential to significantly advance both scientific and industrial applications involving steam and other fluids under extreme conditions. By addressing the limitations of existing models, such as IAPWS-IF97 and IAPWS-95, at pressures beyond 1250 MPa, this project paves the way for more accurate predictions of thermodynamic and transport properties in uncharted regimes, bridging the gap between traditional methods and modern computational advancements.

Contributions

This project aims to predict steam table properties under supercritical conditions. While thermodynamic steam table data are traditionally obtained through experimentation, this process is often costly and time-consuming, especially at extreme temperature and pressure ranges. Using machine learning techniques, we aim to develop a model capable of providing accurate predictions at these conditions.

Although similar studies have been conducted, they have not focused on capturing the thermodynamic properties of steam at extreme temperatures and pressures. Previous projects have developed neural network models for boilers, but do not capture the thermodynamic properties of steam but instead capture work output and steam velocity. Other projects have created accurate data for the steam tables at nominal conditions such as CoolProp

In our study, we utilized CoolProp to generate testing and training data. CoolProp uses formulations from the International Association for the Properties of Water and Steam (IAPWS), specifically:

I. IAPWS-IF97:
    A. Optimized for industrial applications like power plants.
    B. Fast and computationally efficient, but less accurate near the critical point.

II. IAPWS-95:
    A. A highly accurate scientific formulation based on Helmholtz energy.
    B. Designed for applications requiring high precision in scientific and engineering contexts.

The models simulate various properties, including saturation properties, phase boundaries, thermodynamic properties (e.g., enthalpy, entropy, specific volume), and transport properties (e.g., viscosity, thermal conductivity). However, these models encounter significant limitations in extreme conditions, particularly beyond 1250 MPa, where accuracy deteriorates.

Our project addresses these challenges by attempting to capture the fluid properties of steam under such extreme conditions. If the methods developed in this study prove effective, they could not only predict steam table data but also enable predictions of properties for other fluids in similar conditions.

Report

Introduction

Thermodynamic steam tables are critical in engineering design and analysis, specifically in mechanical engineering and scientific applications. The data generated in the steam tables are only experimentally gained These tables provide data on the properties of water and steam across a wide range of temperatures and pressures.

Steam Tables are used in thermodynamics, energy systems, and fluid mechanics. Enabling accurate design and analysis of power plants, refrigeration systems, and industrial equipment. These tables fail to capture the properties of steam when applied to extreme conditions. Since all data is experimentally extracted gaining these properties at extreme conditions can become extremely costly as more advanced machines and technology must be used. Such scenarios are common where precise data at extreme pressures and temperatures are crucial.

At these extreme conditions steam enter a supercritical state the fluid exhibits properties of both liquid and gas. These conditions causes failures of previous made models to predict steam properties. Steam at supercritcal conditions is highly compressible, and its density and other thermodynamic properties are affected significantly by slight changes in pressure and temperature.

In this project, we used machine learning techniques to predict thermodynamic properties of steam at extreme conditions, beyond the ranges typically found in steam tables. Our main aim for training the models is to provide accurate and reliable predictions for key parameters such as enthalpy, entropy, and specific volume under unconventional conditions.

The following are the target variables to be predicted at given Pressure and Temperature

- Specific Volume (m^3/kg)
- Enthalpy (kJ/kg)
- Entropy (kJ/kg·K)
- Internal Energy (kJ/kg)
- Viscosity (μPa·s)
- Thermal Conductivity (mW/m·K)
- Density (kg/m^3)
- Prandtl Number (dimensionless)

Results

Below, Figure 1 illustrates the varying trends of steam viscosity as temperature increases at different pressures, highlighting the complex behavior of the fluid under varying conditions. The plot showcases how viscosity responds differently to temperature changes depending on the pressure, emphasizing the non-linear and pressure-dependent nature of fluid behavior. It is important to note that such variations are not universal and can differ substantially depending on the specific target variable being observed, such as thermal conductivity, density, or other transport properties. This variability highlights the need for robust predictive models that can accurately capture these nuanced behaviors across a wide range of thermodynamic conditions.

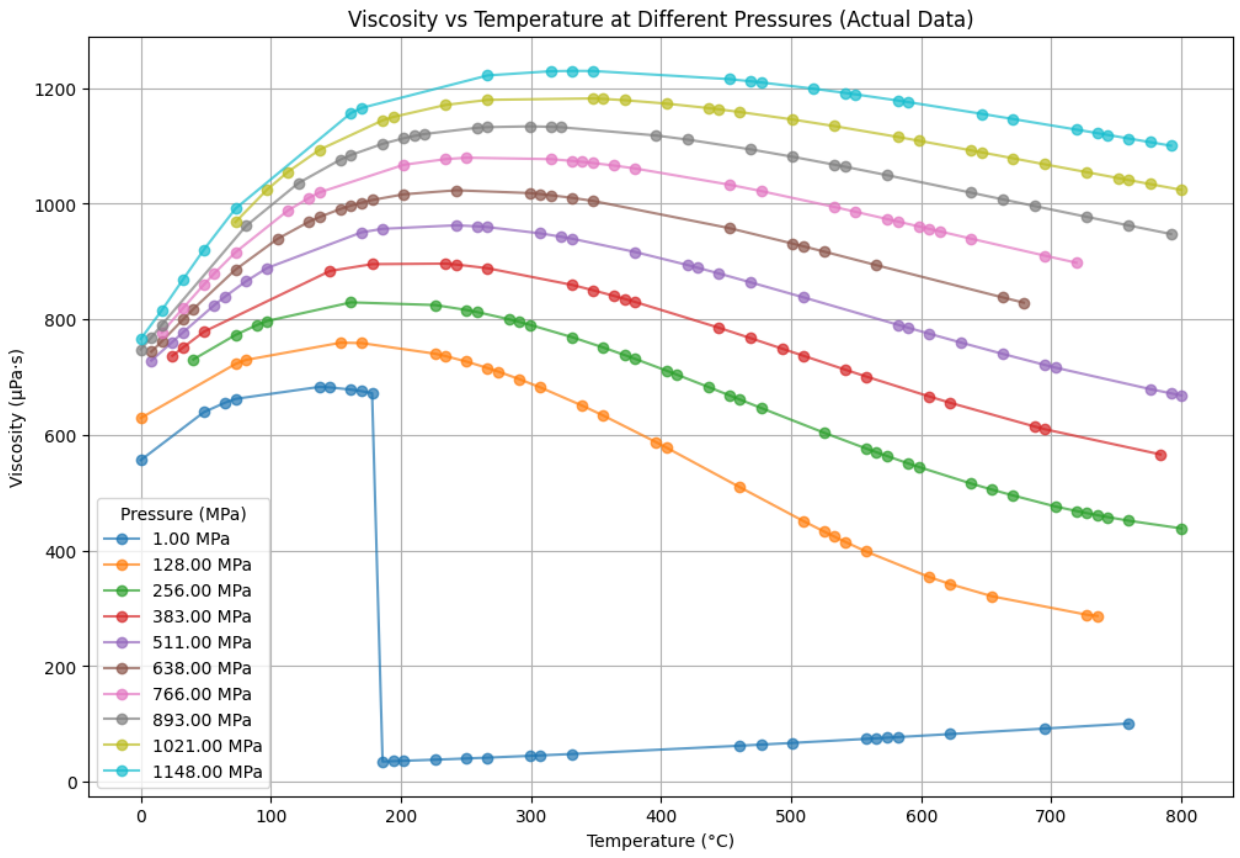

Figure 1 Viscosity vs Temperature Trends at Different Pressures

Table 1 provides a comprehensive overview of the regressors utilized in this study, along with their respective strengths and weaknesses. Each regressor brings unique advantages and challenges, making it essential to understand their suitability for different datasets and problem scenarios. Below is the detailed explanation of the table:

Regressor	Strength	Weaknesses
Linear Regression	- Simple and easy to interpret. - Fast to train and works well for linearly separable data.	- Struggles with complex, nonlinear relationships. - Sensitive to outliers.
Lasso Regression	- Performs feature selection by shrinking less important coefficients to zero. - Prevents overfitting.	- Can eliminate important features if the alpha parameter is not well-tuned. - Sensitive to outliers.
Polynomial Regression	- Captures nonlinear relationships. - Simple extension of linear regression.	- Prone to overfitting for high-degree polynomials. - Requires careful feature scaling.
Support Vector Regression	- Effective for small datasets with nonlinear relationships. - Robust to outliers with proper kernel and parameters.	- Computationally expensive for large datasets. - Requires careful tuning of kernel and parameters.
Gradient Boosting	- Strong performance on a wide variety of datasets. - Handles complex relationships and feature importance.	- Prone to overfitting if not tuned well. - Slow to train on large datasets.
Random Forest	- Handles large datasets and works well with high-dimensional data. - Robust to overfitting and outliers.	- Less interpretable compared to simpler models. - Slower prediction time for large forests.
XGBoost	- Highly efficient and performs well on structured data. - Built-in handling of missing data and regularization.	- Computationally expensive for very large datasets. - Requires extensive hyperparameter tuning.
K-Nearest Neighbors	- Simple to implement. - Non-parametric, adapts well to complex relationships.	- Sensitive to the choice of k. - Computationally expensive for large datasets (requires distance computation).
Neural Network Regression	- Captures highly complex and nonlinear relationships. - Scalable to very large datasets.	- Requires large amounts of data for good performance.

		- Computationally expensive and prone to overfitting. - Hard to interpret.

Table 1. Strength and Weakness of Regressors used

Figures 2-4 provide a visual representation of the target variables plotted against temperature for each regression model each Figure are plotted at different Pressures. This allows for a comparison of model predictions to actual data. In these plots, the black line represents the actual data, serving as a reference to evaluate the accuracy of each regression model. The individual regression models are depicted as separate curves, illustrating how well each model captures the relationship between the target variable and temperature. These figures make it easier to assess how closely the predicted trends align with the true data across different temperature ranges and regression techniques, highlighting the strengths and limitations of each model in handling the dataset.

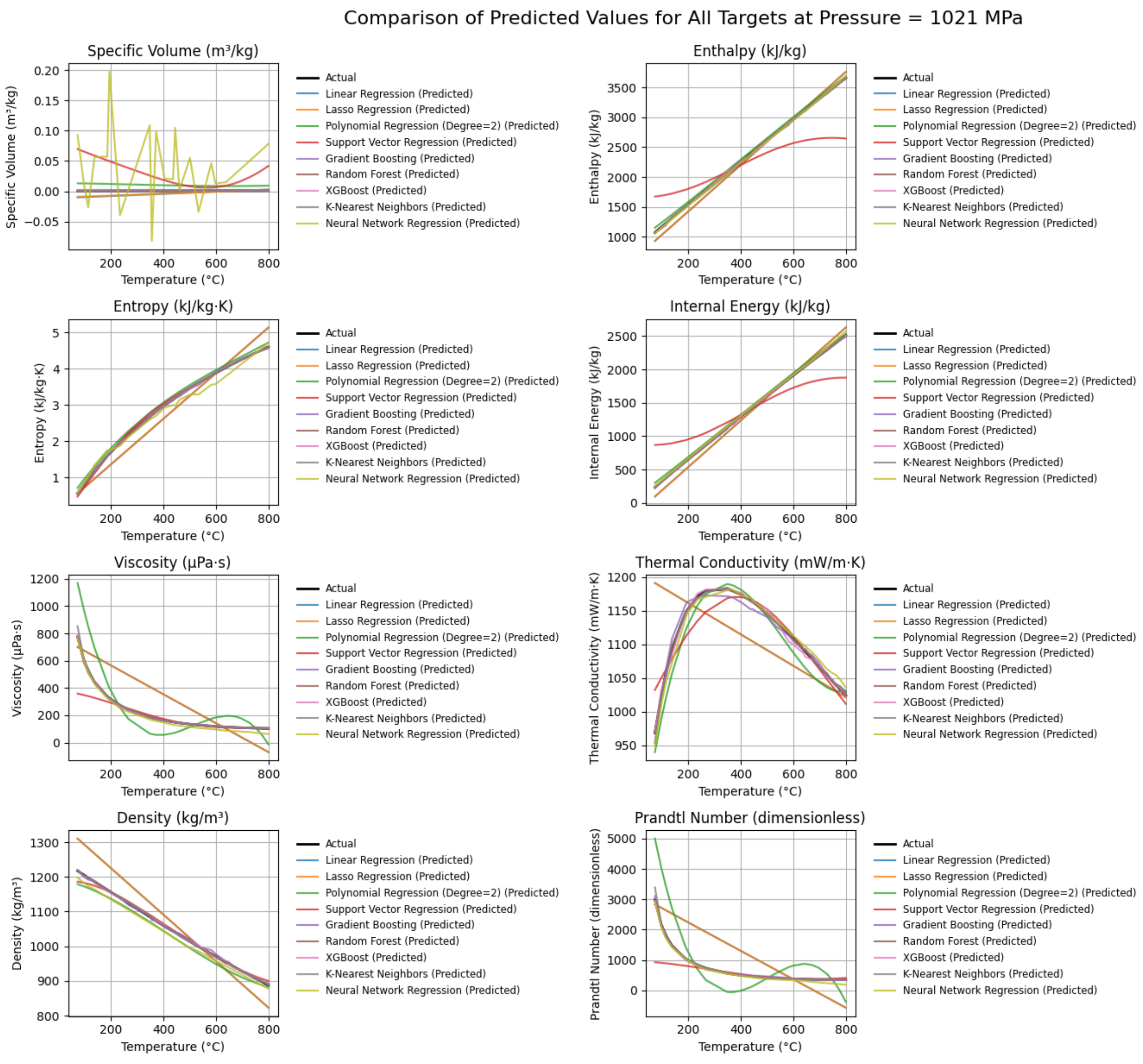

Figure 2. Predicted values per Regressor at 1021 MPa

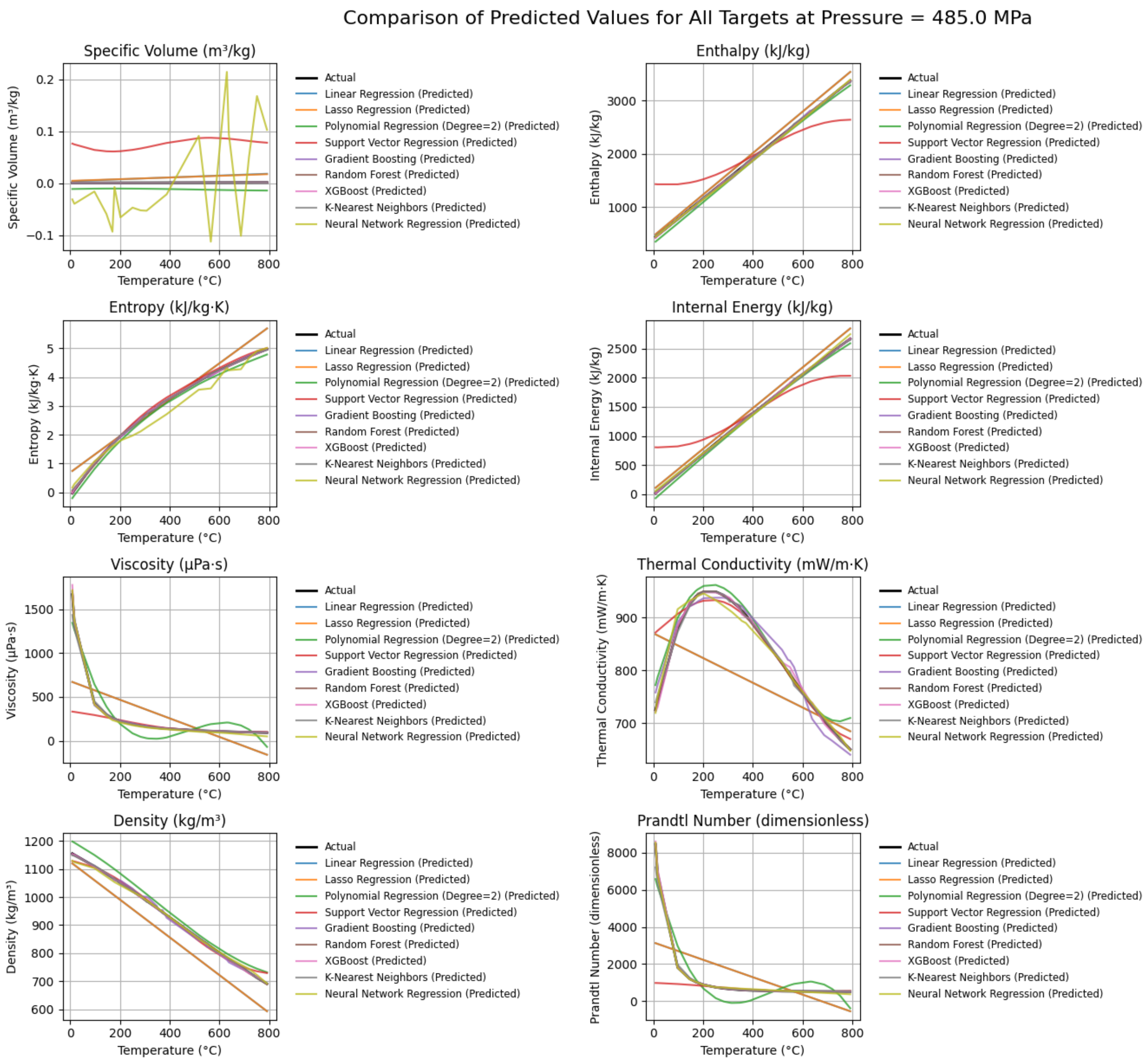

Figure 3. Predicted values per Regressor at 485 MPa

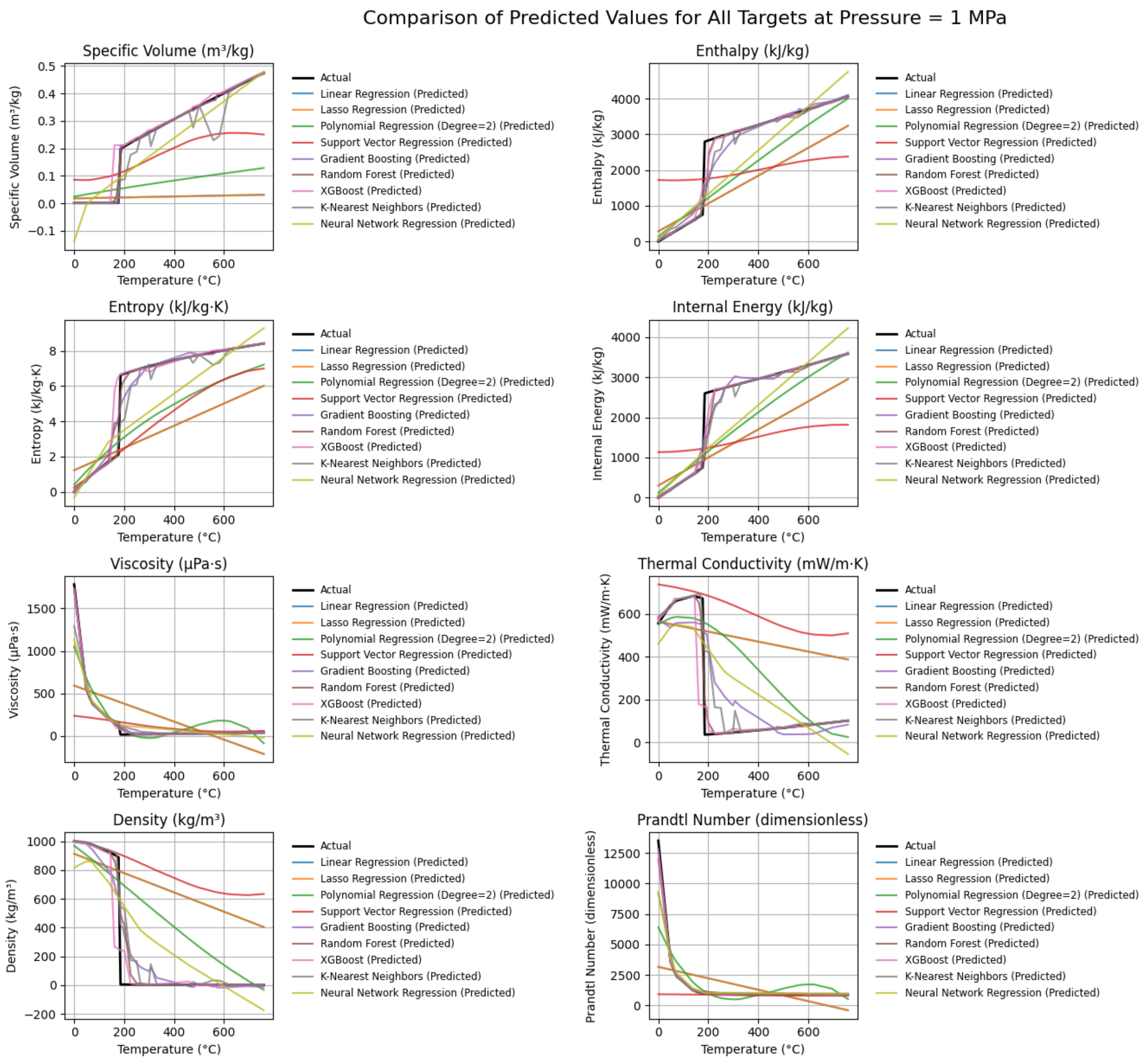

Figure 4. Predicted values per Regressor at 1 MPa

Target Variable	Linear Regression	Lasso Regression	Polynomial Regression (Degree=2)	Support Vector Regression	Gradient Boosting	Random Forest	XGBoost	K-Nearest Neighbors	Neural Network Regression
Specific Volume (m^3/kg)	1.2982E-03	1.2949E-03	1.1001E-03	4.379E-03	5.2281E-07	6.257E-06	8.9649E-05	1.5762E-04	4.7696E-03
Enthalpy (kJ/kg)	4.5582E+04	4.5582E+04	2.2357E+04	3.2031E+05	4.9181E+03	3.1994E+03	2.7101E+03	3.5055E+03	1.3932E+04
Entropy (kJ/kg·K)	2.8013E-01	2.8011E-01	1.2223E-01	1.2109E-01	1.4124E-02	8.4142E-03	3.463E-02	1.894E-02	1.2828E-01
Internal Energy (kJ/kg)	3.8522E+04	3.8522E+04	1.6481E+04	2.2863E+05	2.5329E+03	3.2831E+03	1.8804E+03	2.892E+03	1.2256E+04
Viscosity (μPa·s)	1.1597E+05	1.1597E+05	3.7225E+04	1.5275E+05	1.9952E+02	1.4389E+02	1.6456E+02	6.2787E+03	9.3877E+02
Thermal Conductivity (mW/m·K)	1.6237E+04	1.6237E+04	2.0857E+03	9.4737E+03	8.0673E+02	8.7817E+01	5.6562E+02	3.7538E+02	1.3139E+03
Density (kg/m^3)	1.2611E+04	1.2611E+04	4.1695E+03	1.4267E+04	6.8771E+02	4.5428E+02	9.5219E+02	6.6103E+02	1.9963E+03
Prandtl Number (dimensionless)	2.8571E+06	2.8571E+06	9.2796E+05	4.0831E+06	7.349E+03	3.9536E+03	2.9982E+03	1.4364E+05	2.8275E+04
Average MSE	3.8575E+05	3.8575E+05	1.2628E+05	6.0106E+05	2.0617E+03	1.3903E+03	1.1589E+03	1.967E+04	7.339E+03

Table 2. MSE of each Target Varible per Regression

Scatter plots revealed that both Random Forest and Gradient Boosting produced predictions closely aligned with actual values, while Support Vector Regression exhibited significant outliers in error distribution graphs. Polynomial Regression showed good clustering overall but struggled under extreme conditions. MSE comparison bar charts consistently identified Random Forest, Gradient Boosting, and XGBoost as top-performing models across most metrics, with Gradient Boosting providing the closest match to true values. Random Forest achieved the lowest average MSE across most target variables, demonstrating its robustness, whereas Support Vector Regression consistently showed the highest deviations. Neural Network Regression delivered promising results but exhibited higher variance compared to other models. Based on these results and having lowest average MSE Random forest Regression was chosen to the primary model.

Based on these results, and its consistently low MSE, Random Forest Regression was selected as the primary model for further analysis. The following figures will exclusively utilize actual data and predictions generated by the Random Forest Regression model to provide deeper insights into the behavior of the target variables under various conditions.

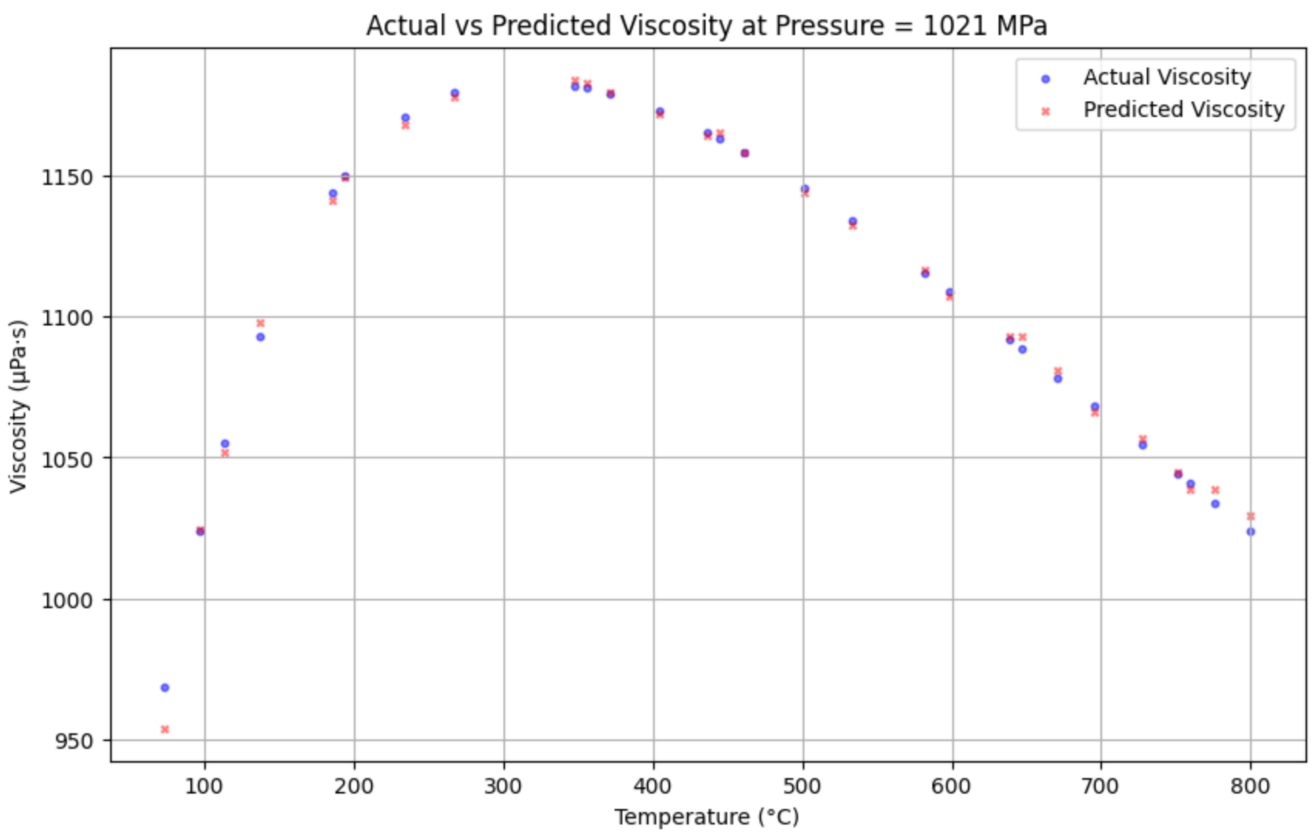

Figure 5. Predicted/Actual values for Viscosity using Random Tree at 1021 MPa

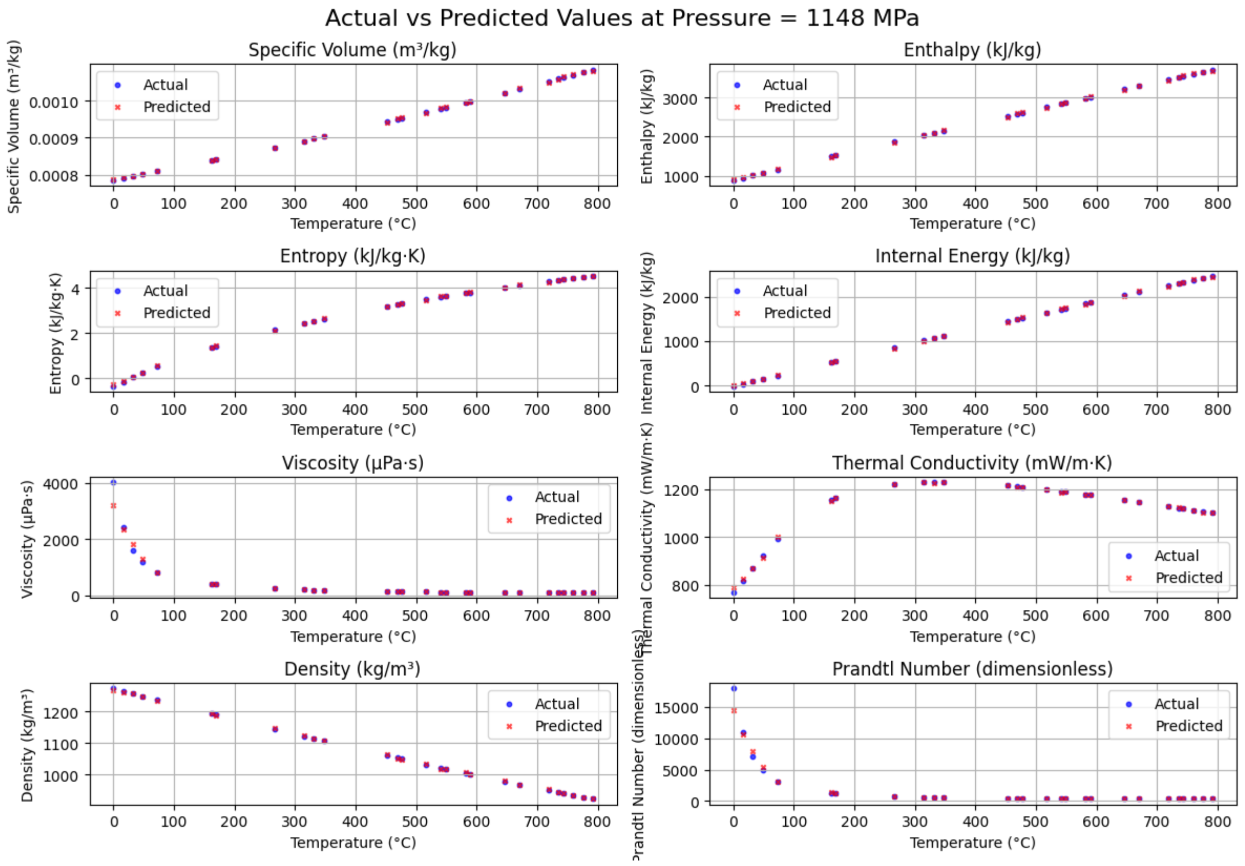

Figure 6. Predicted/Actual values for Target Values using Random Tree at 1148 MPa

Figures 5 and 6 demonstrate that the Random Forest model excels in accurately predicting the target values, particularly when operating under nominal conditions. The plots reveal that the predicted values from the Random Forest model closely align with the actual data, reflecting its ability to capture the underlying patterns and relationships within the dataset. This consistency across nominal conditions highlights the robustness of Random Forest in handling structured data and its ability to generalize well without significant overfitting. Additionally, the results emphasize the model's strength in dealing with high-dimensional data and its capability to manage complex interactions between variables, making it an ideal choice for predictions under standard operating conditions.

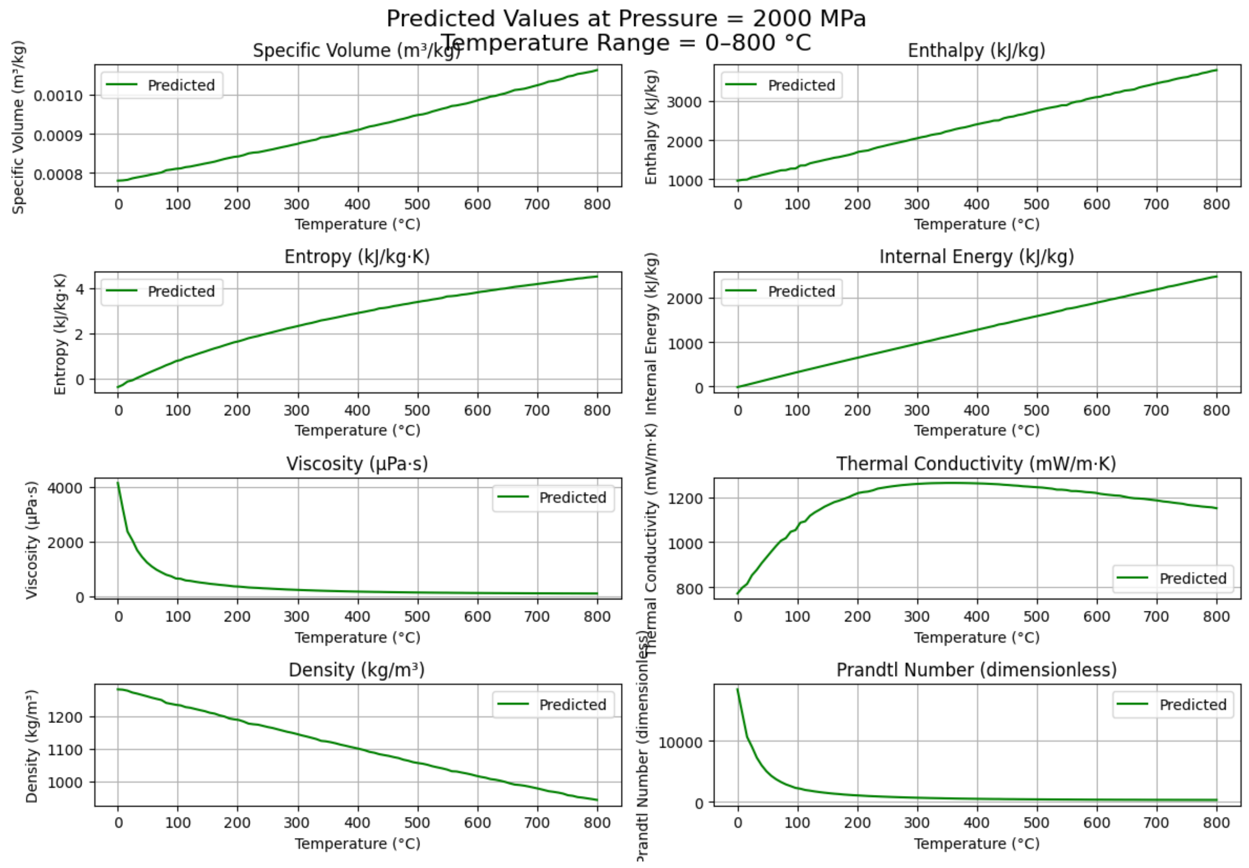

Figure 7. Predicted values at Extreme conditions using Random Forest at 2000 Mpa

Figure 7 showcases the predicted values for various target variables using the Random Forest model at a pressure of 2000 MPa and within a temperature range of 0–800 °C. This figure demonstrates that the Random Forest model is robust in predicting fluid properties at extreme conditions, accurately reflecting the expected physical trends across all target variables. The consistent alignment between predicted values and theoretical expectations highlights the model's ability to generalize and provide reliable results under extreme pressure and temperature conditions. This performance underscores the model's potential as a tool for predicting complex thermodynamic behaviors where experimental data are challenging to obtain.

However, due to the lack of accessible experimental data for steam at supercritical conditions, it remains difficult to quantitatively assess the accuracy of these predictions. While the model's trends appear consistent with theoretical expectations, the absence of validation against real-world measurements introduces uncertainty in determining the exact reliability of the predictions at such extreme conditions. This limitation highlights the need for future efforts to obtain experimental data at supercritical pressures and temperatures to further validate and refine the model’s predictive capabilities.

Conclusion

The main focus of this project is to demonstrate the potential of using machine learning methods to predict thermodynamic properties of steam at extreme conditions, addressing the limitations of traditional steam tables.

Random Forest, Gradient Boosting, and XGBoost were the best performing models, as they consistently provided accurate predictions across most thermodynamic properties. Nonlinear relationships and data sparsity were handled effectively and that performance made them well-suited for extreme conditions. As a result, when experimental data is unavailable or costly to obtain, they showed their ability to provide reliable approximations.

Our new methodology can also be extended to analyze other fluids beyond steam. This capability is particularly valuable in high-pressure, high-temperature applications such as high-efficiency turbines and specialized power generation equipment.

In the end, this project presents a new way for more versatile and scalable tools in thermodynamic analysis By bridging the gap between traditional empirical methods and modern computational techniques. Future work can focus on integrating hybrid physics-based machine learning models to enhance prediction accuracy and reliability further.

References

Thermopedia. "Water and Steam Properties (Thermodynamic Tables)." *Thermopedia*, 2011, https://thermopedia.com/content/1150/.

Asadi, R., et al. "Prediction of Steam Properties Using Artificial Neural Networks." *International Journal of Thermal Sciences*, vol. 49, no. 7, 2010, pp. 1150–1156.

Bell, Ian H., et al. "Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp." *Industrial & Engineering Chemistry Research*, vol. 53, no. 6, 2014, pp. 2498–2508, https://doi.org/10.1021/ie4033999.

The International Association for the Properties of Water and Steam (IAPWS). "IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use." *The International Association for the Properties of Water and Steam*, 1995, https://iapws.org/newform.html.

Bird, R. B., Stewart, W. E., Lightfoot, E. N., & Klingenberg, D. J. (2014). *Introductory Transport Phenomena*. Wiley.

Code

All code was written by us and none was taken from outside sources

```
import os
import CoolProp.CoolProp as CP  # Thermodynamic Tables
import numpy as np
import pandas as pd
import csv
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

pressures = np.linspace(1, 1250, 50) #Thermodynamic Tables has maximum range of
pressure from 1 to 1250 MPa
#Cleaning Pressure values
pressures = np.round(pressures)
pressures = np.unique(pressures)
temperatures = np.linspace(0.1, 800, 100)
columns = ['Pressure (MPa)', 'Temperature (°C)', 'Specific Volume (m³/kg)',
              'Enthalpy (kJ/kg)', 'Entropy (kJ/kg·K)', 'Internal Energy (kJ/kg)',
              'Viscosity (µPa·s)', 'Thermal Conductivity (mW/m·K)',
              'Density (kg/m³)', 'Prandtl Number (dimensionless)']
#Values to be Predicted
target_labels = [
   'Specific Volume (m³/kg)', 'Enthalpy (kJ/kg)', 'Entropy (kJ/kg·K)',
   'Internal Energy (kJ/kg)', 'Viscosity (µPa·s)', 'Thermal Conductivity (mW/m·K)',
   'Density (kg/m³)', 'Prandtl Number (dimensionless)'
]
```

```
selected_pressures = pressures[::5]  # Select every 5th value (50 / 10 = 5)
#Regressors used
regressors = {
   "Linear Regression": LinearRegression(),
   "Lasso Regression": Lasso(alpha=0.1, random_state=3),
   "Polynomial Regression (Degree=2)": Pipeline([
       ('poly_features', PolynomialFeatures(degree=3, include_bias=False)),
       ('linear_regression', LinearRegression())
   ]),
   "Support Vector Regression": SVR(kernel='rbf', C=1.0, epsilon=0.1),
   "Gradient Boosting": GradientBoostingRegressor(random_state=3),
   "Random Forest": RandomForestRegressor(random_state=3),
   "XGBoost": XGBRegressor(random_state=3),
   "K-Nearest Neighbors": KNeighborsRegressor(n_neighbors=5),
   "Neural Network Regression": MLPRegressor(hidden_layer_sizes=(100, 50),
max_iter=1000, random_state=3)
}
results = {}

def data_generation(pressures, temperatures):
   data = []
   for P in pressures:
       for T in temperatures:
           try:
               # Calculate properties for the given pressure and temperature
               v = 1 / CP.PropsSI('D', 'P', P * 1e6, 'T', T + 273.15, 'Water')
#(m³/kg)
               h = CP.PropsSI('H', 'P', P * 1e6, 'T', T + 273.15, 'Water') / 1000  #
Enthalpy (kJ/kg)
               s = CP.PropsSI('S', 'P', P * 1e6, 'T', T + 273.15, 'Water') / 1000  #
Entropy (kJ/kg·K)
               u = CP.PropsSI('U', 'P', P * 1e6, 'T', T + 273.15, 'Water') / 1000  #
Internal energy (kJ/kg)
               eta = CP.PropsSI('VISCOSITY', 'P', P * 1e6, 'T', T + 273.15, 'Water') *
1e6  # Viscosity (μPa·s)
               lambd = CP.PropsSI('CONDUCTIVITY', 'P', P * 1e6, 'T', T + 273.15,
'Water') * 1e3  # Thermal conductivity (mW/m·K)
               rho = CP.PropsSI('D', 'P', P * 1e6, 'T', T + 273.15, 'Water')  #
Density (kg/m³)
               Pr = eta * CP.PropsSI('C', 'P', P * 1e6, 'T', T + 273.15, 'Water') /
lambd  # Prandtl number
```

```
                # Append data to the list
                data.append([P, T, v, h, s, u, eta, lambd, rho, Pr])
            except Exception as e:
                print(f"Error at P={P} MPa, T={T} °C: {e}")
                continue
    return data

# Define the output file path
output_file = 'steam_table_pressure_temperature.csv'
# Check if the file exists
if os.path.exists(output_file):
    # If the file exists, load it into a DataFrame
    df = pd.read_csv(output_file)
    print(f"Data loaded from existing file: {output_file}")
else:
    # Generate data
    data = data_generation(pressures, temperatures)
    df = pd.DataFrame(data, columns=columns)
    df = df.drop_duplicates(subset=['Pressure (MPa)', 'Temperature (°C)'])
    # Save the data to a CSV file
    df.to_csv(output_file, index=False)
    print(f"Data generated, deduplicated, and saved to file: {output_file}")

# Define the features (X) and target variables (y)
X = df[['Pressure (MPa)', 'Temperature (°C)']].values  # Features: Pressure and
Temperature
y = df.iloc[:, 2:].values  # Target variables: All other properties
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size=0.3, random_state=3,
shuffle=True)

def plot_actual_viscosity_vs_temperature_fixed(Xts, yts, pressures_to_plot):
    plt.figure(figsize=(12, 8))
    for pressure in pressures_to_plot:
        # Filter the data for the specific pressure
        indices_at_pressure = np.isclose(Xts[:, 0], pressure, atol=0.01)  # Check
pressure in the first column of Xts
        temperatures = Xts[indices_at_pressure, 1]  # Extract temperatures for the
filtered rows
        actual_viscosity = yts[indices_at_pressure, 5]  # Extract actual viscosity for
the filtered rows
        if len(temperatures) > 0:  # Ensure there are data points to plot
            # Sort data by temperature to ensure smooth lines
            sorted_indices = np.argsort(temperatures)
            temperatures_sorted = temperatures[sorted_indices]
            viscosity_sorted = actual_viscosity[sorted_indices]
            # Plot the sorted data
            plt.plot(temperatures_sorted, viscosity_sorted, label=f'{pressure:.2f}
MPa', marker='o', alpha=0.7)
        else:
            print(f"No data found for Pressure = {pressure} MPa")
    plt.xlabel('Temperature (°C)')
    plt.ylabel('Viscosity (µPa·s)')
    plt.title('Viscosity vs Temperature at Different Pressures (Actual Data)')
    plt.legend(title="Pressure (MPa)", loc='best')
    plt.grid(True)
    plt.show()

plot_actual_viscosity_vs_temperature_fixed(Xts, yts, selected_pressures)

# Initialize the best and worst results dictionary
best_worst_results = {label: {"best_regressor": None, "best_mse": float('inf'),
                              "worst_regressor": None, "worst_mse": float('-inf')}
                     for label in target_labels}

# Dictionary to store results for each regressor
results = {}

# Train each regressor and compute MSEs
for name, base_model in regressors.items():
    print(f"Training {name}...")
    model = MultiOutputRegressor(base_model)
    model.fit(Xtr, ytr)
    ypred = model.predict(Xts)

    # Calculate the MSE for each target variable
    mse = mean_squared_error(yts, ypred, multioutput='raw_values')
    avg_mse = np.mean(mse)  # Calculate average MSE across all targets
    results[name] = {
        "model": model,
        "ypred": ypred,
        "mse": mse,
        "avg_mse": avg_mse,
    }
    # Print MSE for each target variable
    print(f"\n{name} - Mean Squared Error for each target:")
    for label, error in zip(target_labels, mse):
        print(f"{label}: {error:.4e}")
        # Update the best regressor for the target
        if error < best_worst_results[label]["best_mse"]:
            best_worst_results[label]["best_regressor"] = name
            best_worst_results[label]["best_mse"] = error
        # Update the worst regressor for the target
        if error > best_worst_results[label]["worst_mse"]:
            best_worst_results[label]["worst_regressor"] = name
            best_worst_results[label]["worst_mse"] = error
    print(f"Average MSE for {name}: {avg_mse:.4e}")
    print("-" * 40)

# Compare average MSE across all models
print("\nComparison of Average MSE Across Regressors:")
for name, result in results.items():
    print(f"{name}: Average MSE = {result['avg_mse']:.4e}")

# Prepare data for the CSV file
output_data = []
header = ["Target Variable"] + list(results.keys())  # Header row: target variable +
regressor names

# Populate the rows with MSE values for each target variable
for i, label in enumerate(target_labels):
    row = [label]  # First column is the target variable name
    for name in results.keys():
        row.append(f"{results[name]['mse'][i]:.4e}")  # Add MSE for the current
regressor
    output_data.append(row)

# Add the row for average MSE
average_row = ["Average MSE"]
for name in results.keys():
    average_row.append(f"{results[name]['avg_mse']:.4e}")
output_data.append(average_row)

# Save to a CSV file
output_file = "regressor_mse_results.csv"
with open(output_file, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(header)  # Write the header row
    writer.writerows(output_data)  # Write all rows of data

print(f"\nResults saved to {output_file}")

def plot_all_targets_comparison_for_regressors(results, Xts, yts, pressure,
target_labels):
    # Filter the test data for the specified pressure
    indices_at_pressure = np.isclose(Xts[:, 0], pressure, atol=0.01)
    temperatures = Xts[indices_at_pressure, 1]  # Extract temperatures for the filtered
rows
    if len(temperatures) > 0:
        # Sort the data by temperature for smooth lines
        sorted_indices = np.argsort(temperatures)
        temperatures_sorted = temperatures[sorted_indices]
        num_targets = yts.shape[1]
        plt.figure(figsize=(16, 12))
        for i in range(num_targets):
            # Sort actual values by temperature
            actual_values = yts[indices_at_pressure, i]
            actual_values_sorted = actual_values[sorted_indices]
            # Plot actual data
            plt.subplot((num_targets + 1) // 2, 2, i + 1)
            plt.plot(temperatures_sorted, actual_values_sorted, color='black',
label='Actual', linewidth=2)
            # Plot predicted values for each regressor
            for name, result in results.items():
                ypred = result["ypred"]
                predicted_values = ypred[indices_at_pressure, i]
                predicted_values_sorted = predicted_values[sorted_indices]
                plt.plot(temperatures_sorted, predicted_values_sorted, label=f"{name}
(Predicted)", alpha=0.8)
            plt.title(target_labels[i])
            plt.xlabel('Temperature (°C)')
            plt.ylabel(target_labels[i])
            plt.grid(True)
            # Position the legend outside the plot area
            if i % 2 == 0:  # For left-side subplots
                plt.legend(loc='upper left', bbox_to_anchor=(1.05, 1),
fontsize='small', frameon=False)
            else:  # For right-side subplots
```

```
                plt.legend(loc='upper left', bbox_to_anchor=(1.05, 1),
fontsize='small', frameon=False)
        plt.tight_layout(rect=[0, 0, 0.85, 1])  # Adjust layout to accommodate legends
        plt.suptitle(f'Comparison of Predicted Values for All Targets at Pressure =
{pressure} MPa', y=1.02, fontsize=16)
        plt.show()
    else:
        print(f"No test data found for Pressure = {pressure} MPa")

plot_all_targets_comparison_for_regressors(results, Xts, yts,
pressure=1021,target_labels=target_labels)
plot_all_targets_comparison_for_regressors(results, Xts, yts,
pressure=485.0,target_labels=target_labels)
plot_all_targets_comparison_for_regressors(results, Xts, yts,
pressure=1,target_labels=target_labels)

# Train a multi-output regression model
model = MultiOutputRegressor(RandomForestRegressor(random_state=3))
model.fit(Xtr, ytr)
# Predict on the test set
ypred = model.predict(Xts)
# Calculate the Mean Squared Error for each target variable
mse = mean_squared_error(yts, ypred, multioutput='raw_values')
# Display the MSE for each target in a readable format
print("Mean Squared Error for each target variable:")
for label, error in zip(target_labels, mse):
    print(f"{label}: {error:.4e}")  # Formats the error in scientific notation with 4
decimal places

def plot_viscosity_comparison_at_pressure(Xts, yts, ypred, pressure):
    # Filter the test data for the specified pressure
    indices_at_pressure = np.isclose(Xts[:, 0], pressure, atol=0.01)  # Check pressure
in the first column of Xts
    temperatures = Xts[indices_at_pressure, 1]  # Extract temperatures for the filtered
rows
    actual_viscosity = yts[indices_at_pressure, 5]  # Actual viscosity for the filtered
rows
    predicted_viscosity = ypred[indices_at_pressure, 5]  # Predicted viscosity for the
filtered rows
    if len(temperatures) > 0:  # Ensure there are data points to plot
        plt.figure(figsize=(10, 6))
        plt.scatter(temperatures, actual_viscosity, color='blue', label='Actual
Viscosity', alpha=0.5, s=10, marker='o')
        plt.scatter(temperatures, predicted_viscosity, color='red', label='Predicted
Viscosity', alpha=0.5, s=10, marker='x')
        plt.xlabel('Temperature (°C)')
        plt.ylabel('Viscosity (µPa·s)')
        plt.title(f'Actual vs Predicted Viscosity at Pressure = {pressure} MPa')
        plt.legend()
        plt.grid(True)
        plt.show()
    else:
        print(f"No test data found for Pressure = {pressure} MPa")

# Example: Compare actual and predicted viscosity for a specific pressure in the test
set
plot_viscosity_comparison_at_pressure(Xts, yts, ypred, pressure=1021)

def plot_targets_comparison_at_pressure(Xts, yts, ypred, pressure, target_labels):
    # Filter the test data for the specified pressure
    indices_at_pressure = np.isclose(Xts[:, 0], pressure, atol=0.01)  # Check pressure
in the first column of Xts
    temperatures = Xts[indices_at_pressure, 1]  # Extract temperatures for the filtered
rows
    if len(temperatures) > 0:  # Ensure there are data points to plot
        num_targets = yts.shape[1]  # Number of target columns
        plt.figure(figsize=(12, 8))
        for i in range(num_targets):
            actual_values = yts[indices_at_pressure, i]  # Actual values for the
current target
            predicted_values = ypred[indices_at_pressure, i]  # Predicted values for
the current target
            # Plot actual and predicted values for the current target
            plt.subplot((num_targets + 1) // 2, 2, i + 1)  # Create subplots in a grid
format
            plt.scatter(temperatures, actual_values, color='blue', label='Actual',
alpha=0.7, s=10, marker='o')
            plt.scatter(temperatures, predicted_values, color='red', label='Predicted',
alpha=0.7, s=10, marker='x')
            plt.title(f'{target_labels[i]}')
            plt.xlabel('Temperature (°C)')
            plt.ylabel(target_labels[i])
            plt.legend()
```

```
           plt.grid(True)
       plt.tight_layout()  # Adjust layout for better visualization
       plt.suptitle(f'Actual vs Predicted Values at Pressure = {pressure} MPa',
y=1.02, fontsize=16)
       plt.show()
   else:
       print(f"No test data found for Pressure = {pressure} MPa")

# Call the function to plot all targets for a specific pressure
plot_targets_comparison_at_pressure(Xts, yts, ypred, pressure=1148,
target_labels=target_labels)

def plot_predicted_values_at_pressure_and_temperature_range_with_model(model, Xts,
pressure, temp_min, temp_max, target_labels):
   # Generate new input data for the specified pressure and temperature range
   temperatures = np.linspace(temp_min, temp_max, 100)  # Generate 100 points in the
temperature range
   pressures = np.full_like(temperatures, pressure)  # Fixed pressure value for all
points
   X_new = np.column_stack((pressures, temperatures))  # Combine into feature array
   # Predict values using the trained model
   ypred_new = model.predict(X_new)
   if len(temperatures) > 0:  # Ensure there are data points to plot
       num_targets = ypred_new.shape[1]  # Number of target variables
       plt.figure(figsize=(12, 8))
       for i in range(num_targets):
           # Plot predicted values for the current target
           plt.subplot((num_targets + 1) // 2, 2, i + 1)  # Create subplots in a grid
format
           plt.plot(temperatures, ypred_new[:, i], color='green', label='Predicted')
           plt.title(f'{target_labels[i]}')
           plt.xlabel('Temperature (°C)')
           plt.ylabel(target_labels[i])
           plt.legend()
           plt.grid(True)
       plt.tight_layout()  # Adjust layout for better visualization
       plt.suptitle(f'Predicted Values at Pressure = {pressure} MPa\nTemperature Range
= {temp_min}-{temp_max} °C', y=1.02, fontsize=16)
       plt.show()
   else:
       print(f"No predicted data found for Pressure = {pressure} MPa in the range
{temp_min}-{temp_max} °C")
```

```
plot_predicted_values_at_pressure_and_temperature_range_with_model(
   model, Xts, pressure=2000, temp_min=0, temp_max=800, target_labels=target_labels
)
```