Combined Cycle Power Plant Energy Predictions

Manikandan Perumal,

Darin L. Verduzco,

and Israel Romero Olvera

Shiley-Marcos School of Engineering, University of San Diego

AAI-500: Probability and Statistics for Artificial Intelligence

Dallin Munger, M.S., and Leonid Shpaner, M.S.

October 21st, 2024

Abstract

Since the Nineteenth Century, Electric Power has been the source of advancement of Human Technology. It has been a basic need for more than 100 years. The Power Plants of today are significantly more efficient than ever, like the Combined Cycle Power Plants which take advantage of the heat exhaust generated by their predecessor designs and reuse it to generate even more electricity. The level of efficiency is so high that we can now measure the effects of the environment on the productivity of the plant. Measuring the variance in the output of an electric plant is essential not only for the workers of the facility, but to the investors, the end users, and to the Economy. This paper explores how linear regression, and other models can be used to predict the output of a power plant based on ambient temperature, atmospheric pressure, and other environment variables automatically.

Keywords: machine learning, electric power, linear regression, random forest regression

Introduction

In the Era of Information it is quite hard to imagine our lives without computers, without 24/7 access to the Internet, without a fridge and a microwave in our kitchens, or even without LED light bulbs that illuminate our nights. None of the prior modern necessities can be resolved without a key element: electric power. Without it, there's no modern technology, no research, no economy. Electric Power is critical in our daily lives, and just like Hollywood's visual FX, when electricity works fine it is almost invisible and easy to forget it's there; the moment there's a power outage we surely notice it's gone. Predicting its output could be beneficial for multiple purposes; think of Energy Demand Forecasting, Financial Planning, Carbon Emission Management, Energy Trading, Integration with Renewable Energy, and many more that exceed our imagination. This paper explores how a machine-learning model can work to accurately predict the output of a Combined Cycle Power Plant (CCPP) based on ambient temperature, atmospheric pressure, and other environmental variables that directly affect the productivity of the plant.

Definition of a Combined Cycle Power Plant (CCPP)

Our work is based on the data and research performed by Tüfecki (2014), where they analyze the data captured by sensors in a Combined Cycle Power Plant. A Combined Cycle Power Plant is a facility designed to generate electricity using different methods combined. On their first stage, they burn fuel to make a turbine spin to induce electricity, like regular fuel power plants do, except regular fuel power plants let the heat that results from the burning escape into the atmosphere, while CCPPs make use of that heat to burn water and conduct that steam into another turbine, increasing the output of the plant with this secondary source. Some plants have more than 2 stages and can generate obviously more power. In general, the power that CCPPs can generate go from about 100 MW to 1000 MW per hour.

Data Cleaning and Preparation

We observed data collected in a CCPP for over 6 years (from 2006 to 2011) and identified the independent variables being captured: Ambient temperature (AT), Atmospheric Pressure (AP), Relative Humidity (RH), and Exhaust Vacuum (V); the latter is a very particular measurement that represents the pressure condition at the exhaust of a steam turbine, which can sometimes be lower than the atmospheric pressure (Tüfecki, 2014).

The dependent variable in the data set is the Electrical Power (PE), measured in Megawatts (MW). This is the variable our model will predict based on the other 4 input variables. Table 1 shows the variables and their measuring units (Tüfecki, 2014).

Table 1

Variables definition on the Combined Cycle Power Plant data (Tüfecki, 2014).

Variables Table

Variable Name	Role	Type	Description	Units	Missing Values
AT	Feature	Continuous	in the range 1.81°C and 37.11°C	C	no
V	Feature	Continuous	in teh range 25.36-81.56 cm Hg	cm Hg	no
AP	Feature	Continuous	in the range 992.89-1033.30 milibar	milibar	no
RH	Feature	Continuous	in the range 25.56% to 100.16%	%	no
PE	Target	Continuous	420.26-495.76 MW	MW	no

Table 2 describes basic statistics on all 5 variables to get a better understanding of their minimum and maximum values, mean, standard deviation, and quartiles. We can observe that the data set contains over 9,500 observations, as well as the min and max values of each variable's range.

Table 2

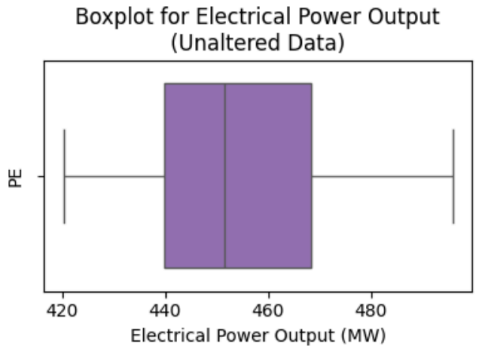
Description of the Data Set Variables.

	count	mean	std	min	25%	50%	75%	max
Ambient_Temperature	9568.0	19.651231	7.452473	1.81	13.5100	20.345	25.72	37.11
Vacuum	9568.0	54.305804	12.707893	25.36	41.7400	52.080	66.54	81.56
Atmospheric_Pressure	9568.0	1013.259078	5.938784	992.89	1009.1000	1012.940	1017.26	1033.30
Relative_Humidity	9568.0	73.308978	14.600269	25.56	63.3275	74.975	84.83	100.16
Electrical_Power_Output	9568.0	454.365009	17.066995	420.26	439.7500	451.550	468.43	495.76

For the dataset, we confirmed there are no null values and identified 41 candidate duplicate values. We confirmed the index values of the matching duplicates were not in sequential order, so did not appear to be errors with entry and therefore were not removed from the dataset. As shown in Figure 1, we found no outliers in the target variable (PE) in the unaltered dataset.

Figure 1

No outliers for Electrical Power Output (PE) in the unaltered dataset.



However, to refine the range of PE values that were present for certain ranges of the independent variable values, we produced binned boxplots of PE within 10 bins of values for each independent variable. A binned box plot of power output should have well defined limits for ranges of independent variable values in order for a regression model to more easily predict electrical power

output. A python function was made to remove PE outliers for all independent variable binned box plots with the following formula for upper and lower bounds:

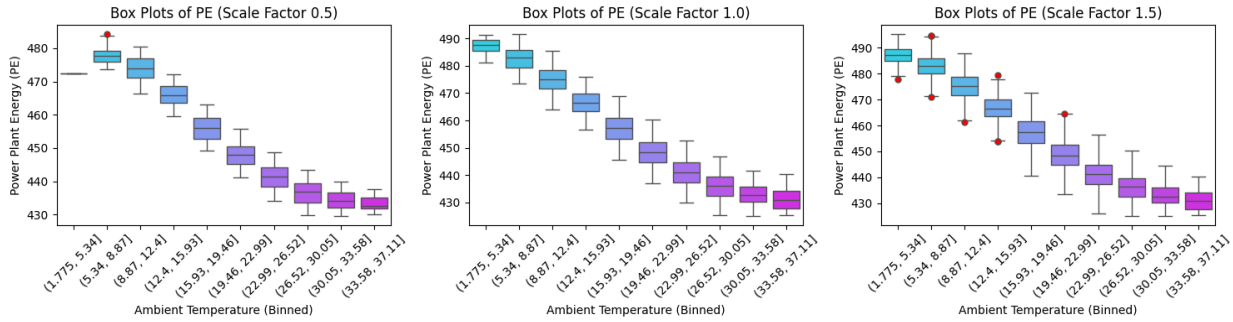
$$Lower\ bound = Q_1 - k \times IQR$$

$$Upper\ bound = Q_3 + k \times IQR$$

The lower and upper bounds serve as the limits to remove values below the lower bound and above the upper bound of each PE binned box plot, where Q_1 is the lower quartile of the PE binned box plot, Q_3 is the upper quartile of each PE binned box plot, k is the value of the scale factor, and IQR is the interquartile range found from subtracting Q_1 from the Q_3 values. After the function was created to remove outliers from the binned PE box plots, it was then necessary to test various scale factors before we found a value that would not remove too many valid data points from our dataset, as well as remove a meaningful amount of binned PE outliers. Scale factors of 0.5, 1.0, and 1.5 were tested to compare binned plots of PE and determine which scale value would be satisfactory in removing outliers, as shown in Figure 2.

Figure 2

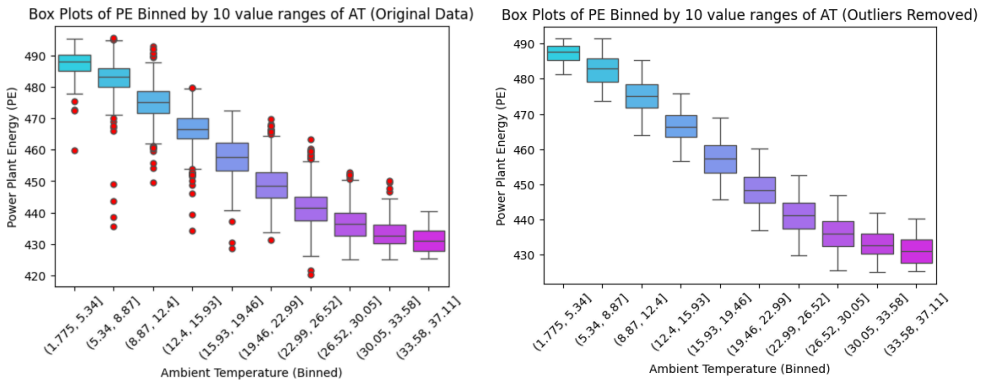
Comparison of outliers remaining for PE binned vs AT using various scale factors to define upper and lower bounds for outlier removal.



We determined that a scale value of 1.0 for calculating upper and lower bounds as the limits for outlier removal had the best effect at removing outliers, as seen in the replotted binned PE box plots. As a result, 1,256 outliers were removed, reducing our dataset to 8,312 observations. Figure 3 demonstrates how outliers for PE emerge if we separate this target variable by 10 value ranges of each independent variable.

Figure 3

Comparison between raw dataset and the new dataset with outliers removed by defining Q1,Q3,IQR and scale factor of 1.0 to define bounds.



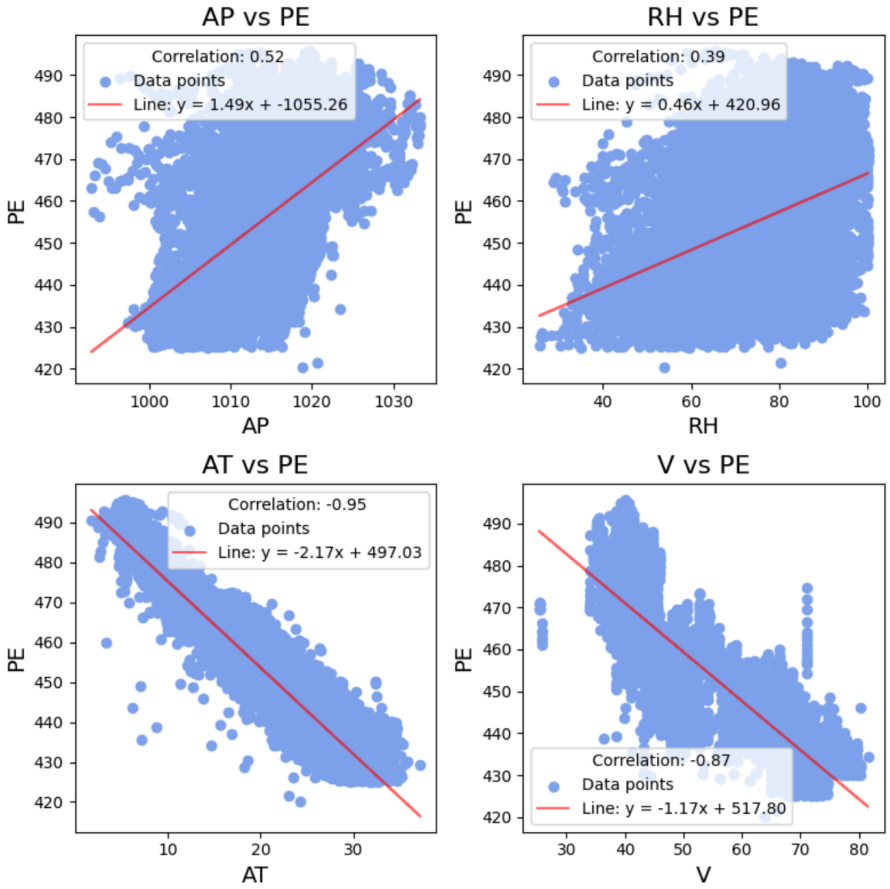
We further divided data into three parts: 64% for training, 16% for validation, and 20% for testing. The first part will be used to train the models, and the second and third parts will be used to evaluate the model's performance.

Exploratory Data Analysis

The next step is to understand the correlations between the variables. We made regression plots for all independent variables, and we found that they show moderate to strong correlations to the dependent variable power output (PE), as shown in Figure 4. We also found that ambient temperature (AT) and Vacuum (V) values have a strong negative correlation to power output (PE); relative humidity (RH) and atmospheric pressure (AP) have a positive moderate and strong correlation to PE.

Figure 4

Regression plot of independent variable vs. PE with a best fit line and correlation value.



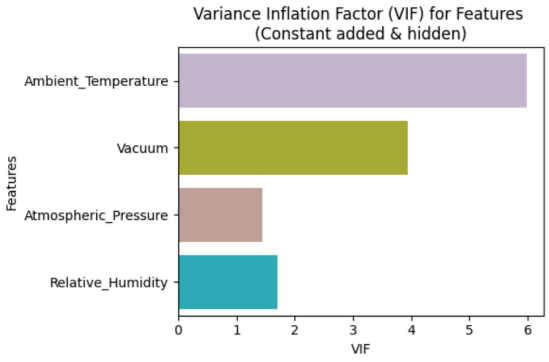
We consider that a linear regression model should work to predict the output variable. In order to consider building a linear regression model with all independent variables, we also analyzed multicollinearity. Independent variables with multicollinearity can be an issue with linear regression models because multicollinearity can inflate variance for the estimates of coefficients, forming unstable and potentially false conclusions about relationships between independent variables. This can produce potentially false predicted values due to a model that struggles to determine the contribution for each independent variable. We first analyzed multicollinearity by calculating VIF values for all independent variables using the following formula:

$$VIF(X_i) = \frac{1}{(1-R_i^2)}$$

In this formula, R_i^2 is the R-squared model fit from the regression of X_i, each independent variable. We found that AT had the highest VIF value compared to the other feature variables, as shown in Figure 5. However, all VIF values are less than 10, which means multicollinearity should not be a concern going forward in our analysis.

Figure 5

VIF values plotted to show levels of collinearity. AT had the highest value of 5.98, but only moderate collinearity is present with all VIF values being less than 10.



Model Selection

We explored several regression models that would likely fit our data. To decide what model will describe it best we will use several statistical tools such as P-values, Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared, which will be defined in detail further along.

We started with a linear regression model using all four features. The predictive model linear equation is:

$$ElectricalPowerOutput = 412.23 - 1.9 \times AmbientTemperature - 0.23 \times Vacuum - 0.14 \times RelativeHumidity - 0.1 \times AtmosphericPressure$$

The linear regression model explains 94% of the variance in the electrical power output of the plant. The model is statistically significant because the p-values of the features are less than 0.05. With this model, we observed that temperature has a high influence on electrical power. It decreases by 1.9 MW for a one-unit increase in the ambient temperature for adjusting the other features.

In order to check the statistical significance of the model, we formulated the null and alternative hypothesis. The null hypothesis is that the features are not statistically significant. The alternative hypothesis is that the features are statistically significant. We used the F-test with all model coefficients as zero to test the null hypothesis. The p-value of the F-test(31138) is less than 0.05, which indicates that the features are statistically significant. Therefore, we are rejecting the null hypothesis and accepting the alternative hypothesis, confirming that the linear regression model is statistically significant.

Next, we explored 8 additional models and compared their performance with the already statistically significant linear regression model. The performance of all models was assessed using three metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-Squared. The results are presented in Table 3.

Table 3

Model metrics with multiple regression models.

Models	Train			Validation			Test		
	MSE	MAE	R^2	MSE	MAE	R^2	MSE	MAE	R^2
Linear Regression	16.7	3.37	0.94	16.79	3.37	0.94	16.18	3.31	0.94
OLS	20.04	3.65	0.92	19.85	3.62	0.93	19.58	3.6	0.93
Decision Tree Regressor	0	0	1	16.23	2.94	0.94	16.94	3.02	0.94
Random Forest Regressor	**1.23**	**0.84**	**1**	**8.81**	**2.29**	**0.97**	**9.25**	**2.34**	**0.97**
Gradient Boosting Regressor	9.76	2.49	0.96	11.48	2.73	0.96	11.23	2.71	0.96
Ada Boost Regressor	14.37	3.14	0.95	15.1	3.19	0.94	15.2	3.2	0.94
Bagging Regressor	1.75	0.95	0.99	9.94	2.43	0.96	10.37	2.45	0.96
SVR	194.65	11.38	0.26	199.06	11.5	0.27	191.83	11.43	0.28
MLP Regressor	20.12	3.65	0.92	19.87	3.62	0.93	19.56	3.59	0.93

The Mean Squared Error (MSE) is a metric used to measure the average of the squares of the differences between predicted and actual values. Lower MSE values indicate better model performance, as they reflect smaller discrepancies between predicted and observed outcomes. In this study, the MSE values for the Random Forest Regressor were 1.23 for the training dataset, 8.81 for the validation dataset, and 9.25 for the test dataset. The low MSE of 1.23 in the training dataset suggests that the model fits the training data well. However, the higher MSE values in the validation and test datasets indicate that the model's predictions are less accurate when applied to new, unseen data.

The Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, and because it is an absolute value, their sign is ignored. The MAE values for the Random Forest Regressor model were 0.84 for the training dataset, 2.29 for the validation dataset, and 2.34 for the test dataset. The relatively low MAE in the training dataset again suggests a good fit to the training

data, while the higher values in the validation and test datasets indicate larger deviations in predictions when the model encounters new data.

R-Squared, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, with higher values indicating better predictive accuracy. An R-Squared value of 0.97 in Table 3 suggests that the Random Forest Regressor explains a high proportion (97%) of the variance in the outcome variable, making it a strong model in terms of predictive power.

Overall, these metrics provide a comprehensive view of the model's performance, highlighting its strengths in fitting the training data and its varying levels of accuracy in predicting new data.

With the results on Table 3 we concluded that Random Forest Regressor is the model that best fits our data and provides the highest R^2 with the lowest MSE and MAE for our Training, Validation, and Test datasets. However, due to the high computational resources required by this model, we decided to compare its performance against the Linear Regression model. In the next section we will compare and analyze the performance of both selected models: Linear Regression and Random Forest Regressor.

Model Analysis

Prior to interpreting the results of a linear regression model, it is essential to verify that the model's assumptions are satisfied. The assumptions include:

- **Linearity**: linear relationship between dependent variables and independent variable
- **Independence**: residuals of the model should not be influenced by the dependent variable
- **Normality of residuals**: residuals of the model should be normally distributed
- **Homoscedasticity:** variance of residuals is constant across the independent variables

Linearity

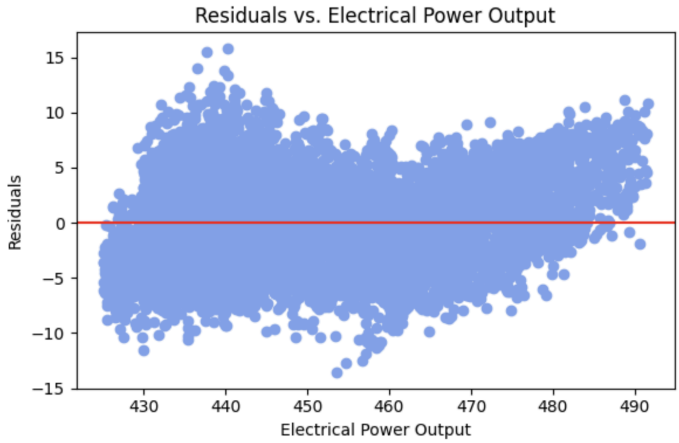
We observed on Figure 4 that the relationship between all independent variables and the dependent variable is linear. The Pearson correlation of independent variables ambient temperature(AT), exhaust vacuum(V), relative humidity(RH), and atmospheric pressure(AP) with the dependent variable electrical power(PE) of the plant are -0.95,-0.87, 0.53, and 0.39 respectively. These values indicate a strong linear relationship for electrical power with ambient temperature and exhaust vacuum, and weak linear relationship for electrical power with relative humidity and atmospheric pressure.

Independence

We checked the residuals of the model against electrical power. We plotted the residuals against the predicted values on Figure 6 and confirmed they are randomly distributed around zero, so there is no relationship between the residuals and the dependent variable.

Figure 6

Residuals vs dependent variables.

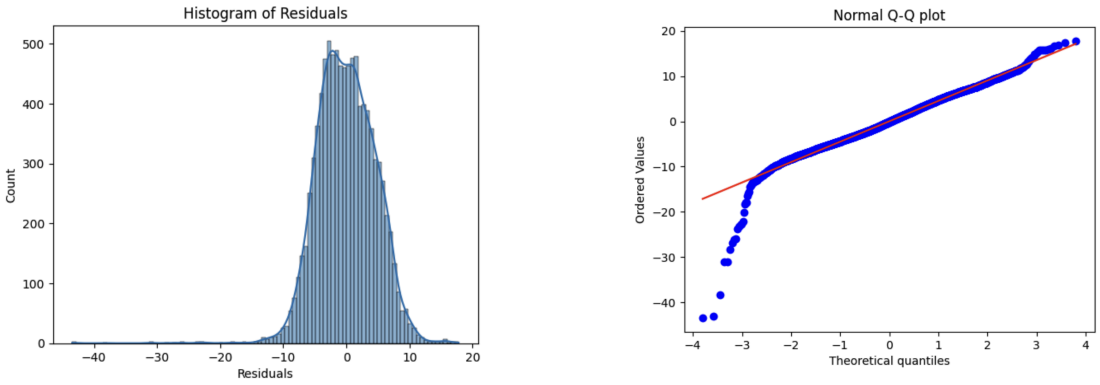


Normality

We checked the normality of the residuals by plotting their histogram on Figure 7 a), where we can see that residuals are normally distributed. Also, the QQ plots on Figure 7 b show the residuals follow a straight line. This confirms that the residuals are normally distributed.

Figure 7

Residual Charts: a) histogram of residuals, b) QQ plot of residuals.

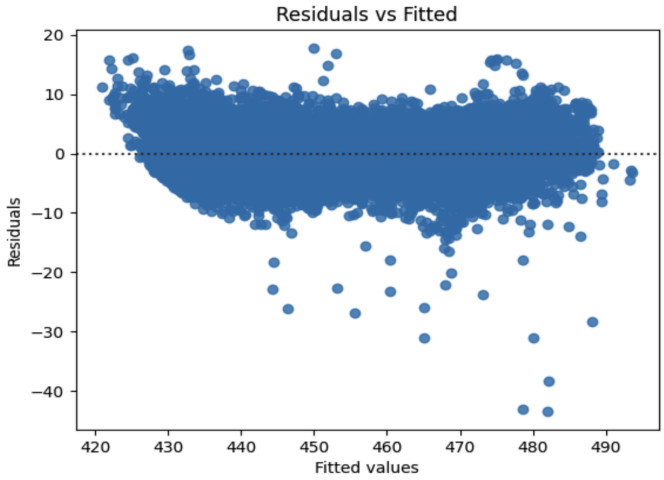


Homoscedasticity

Next, we checked the homoscedasticity of the residuals. We plotted the residuals against the predicted values. From Figure 8, we observe that the residuals are randomly distributed around zero and that their variance around the zero line remains constant. This shows our homoscedasticity assumption is met as expected, confirming that our model selection is adequate.

Figure 8

Homoscedasticity of Residuals.



Based on all the above analysis, we fit the model with the ambient temperature, relative humidity, and atmospheric pressure as dependent variables to predict the power output of the power plant. All 4 criteria to select a linear regression model are met.

On Figures 9 and 10 we can compare visually the predicted vs actual results for both Linear Regression and Random Forest Regressor models, with and without outliers. We found that without outliers, the R^2 improves only by 1% but the MSE gets a larger improvement.

The Random Forest Regressor model performs only about 3% above the Linear Regression model in R^2, while the MSE of the Random Forest Regressor is about half of the Linear, both with and without outliers, also observed in Figures 9 and 10.

Figure 9

Linear Regression model Predicted vs Actual values: a) Using original data; b) Data without outliers.

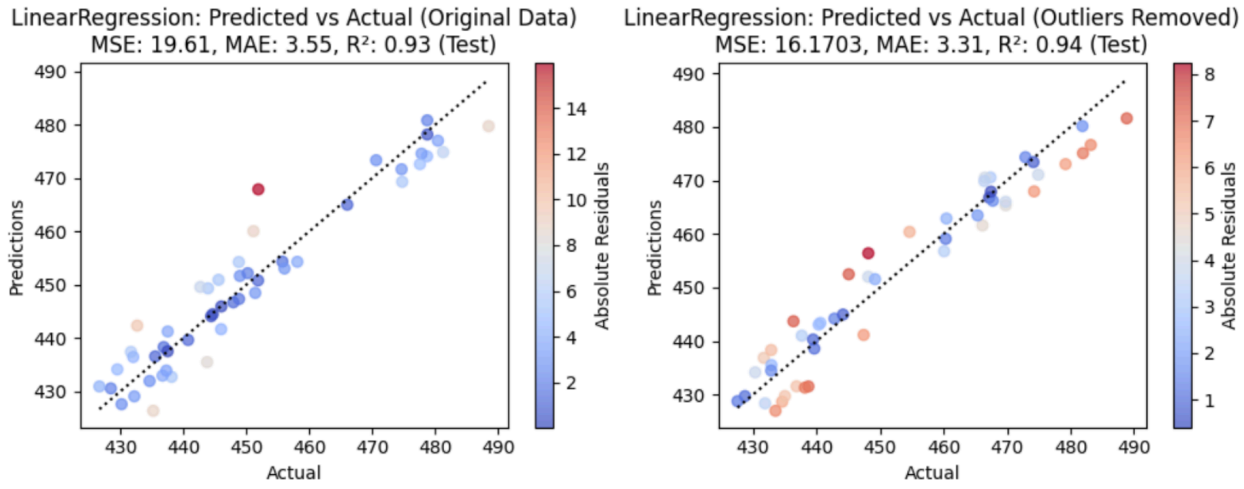
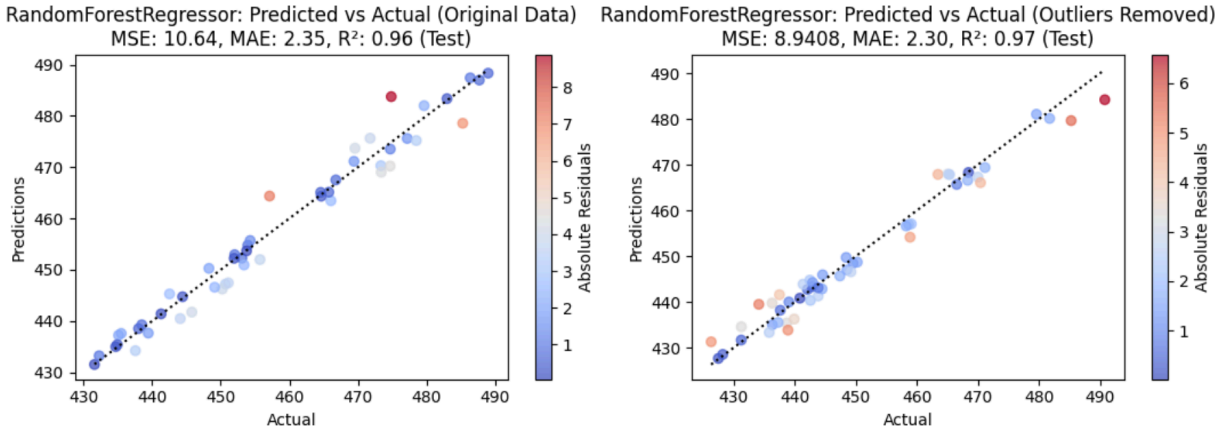


Figure 10

Random Forest Regressor model Predicted vs Actual values: a) Using original data; b) Data without outliers.



We decided to produce another visualization for the comparisons above using the sequential data in the dataset as a trend where we can see the difference between actual and predicted values for both models, also before and after the outliers removal, represented on Figures 11 and 12.

Figure 11

Linear Regression model Predicted vs Actual Trends: a) Using original data; b) Data without outliers.

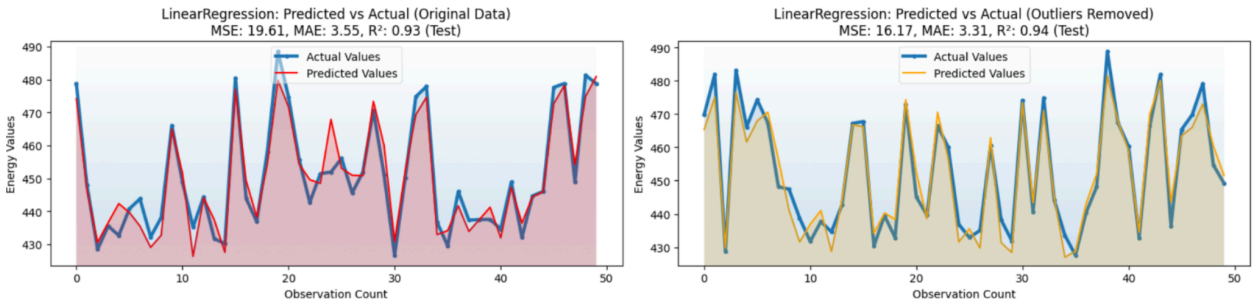
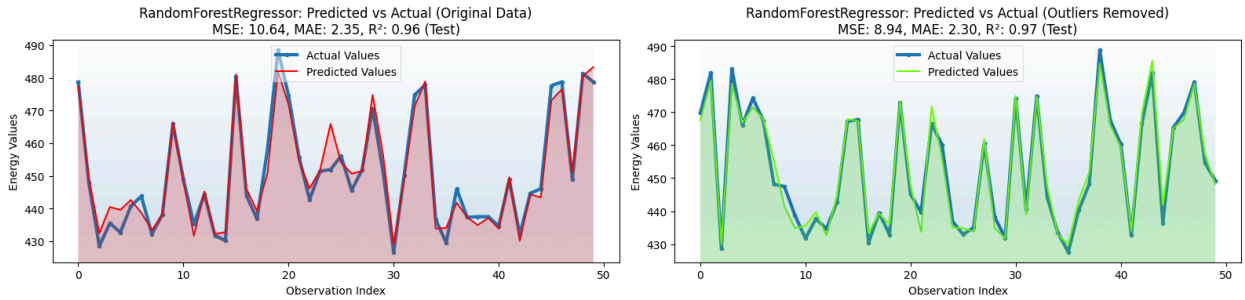


Figure 12

Random Forest Regressor model Predicted vs Actual Trends: a) Using original data; b) Data without outliers.



It is clear that the Random Forest Regressor model performs better matching against the original data, but the Linear Regression model is not that far behind. This is important to consider because the Linear Regression model requires much less computational resources than the Random Forest Regressor Model, which translates in less cost of implementation in the real world.

Conclusion and Recommendations.

Our analysis revealed that the Random Forest Regressor Model is about 3% more accurate than the Linear Regression Model. While every percentage point counts when it comes to matching the dependent variable, the cost is also an important factor to consider. It is well known that Random Forest Regression models require extensive computational power. As a vague example, using our testing data set, the Linear Regression model returned all the predictions in less than 1 second, while the Random Forest Regressor model took over 25 seconds with our laptops. This means that for a real-time application, the Random Forest Regressor model will require a server with multiple processors running in parallel, which are significantly more pricey than a regular laptop or desktop personal computer. More testing with different hardware equipment is highly encouraged, both in terms of personal computers and professional multi-processor servers.

Our recommendation in terms of usage of these models will rely on the need for precision, suggesting the investment on a high-performance hardware for power plants in locations with extreme weather conditions or a high variability of ambient temperature throughout the day.

Future research is also recommended to optimize the algorithm of the Random Forest Regressor model to improve its performance and accuracy. Further refinement with more complex combinations of values for scale factors is also open to further investigation when removing binned target variable outliers from a dataset.

Another area of opportunity for the research on this project is the detection of outliers. While we found a good method to remove outliers at first glance, more research is needed to calibrate the outlier detection algorithm with the local weather conditions for each power plant instance; this will

ensure the reliability of the predictions and prevent biases due to extreme measurements or even potential sensor equipment failure.

Similarly to the Random Forest Generator, the Linear Regression model can likely be improved along with the outlier detection algorithm, which could potentially increase its accuracy while keeping the hardware requirements low in cost.

Finally, future research may reveal additional variables that influence the power output of the plant. Should these additional variables not show multicollinearity with the existing ones, this could improve the accuracy of both models.

References

Tüfekci, P. (2014). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, *60*, 126–140. https://doi.org/10.1016/J.IJEPES.2014.02.027

Tüfekci, P. (2014). Combined Cycle Power Plant [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5002N

OpenAI. (2024). *ChatGPT* (v2.0) [Large language model]. https://chat.openai.com/chat

CCPP_Final

October 21, 2024

1 Appendix: CCPP Outlier Removal, Data Analysis and Regression Models

Group 2: Verduzco, Darin; Perumal, Manikandan; Romero Olvera, Israel

GitHub: https://github.com/isralennon/MSAAI500

Installs:

[1]:
```
# pip install mplcyberpunk
```

2 Import Libraries

[2]:
```
### Libraries for Machine Learning
import pandas as pd
import numpy as np
import statistics as stats
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import mplcyberpunk

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,␣
↪explained_variance_score, accuracy_score
from statsmodels.stats import anova
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor, AdaBoostRegressor,␣
↪BaggingRegressor, RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
```

3 Initial Data Descriptions

[3]:
```
df = pd.read_csv('CombinedCyclePowerPlant.csv', sep=',')
df.head()
```

[3]:

	AT	V	AP	RH	PE
0	8.34	40.77	1010.84	90.01	480.48
1	23.64	58.49	1011.40	74.20	445.75
2	29.74	56.90	1007.15	41.91	438.76
3	19.07	49.69	1007.22	76.79	453.09
4	11.80	40.66	1017.13	97.20	464.43

[4]:
```
df.describe().T
```

[4]:

	count	mean	std	min	25%	50%	75%	max
AT	9568.0	19.651231	7.452473	1.81	13.5100	20.345	25.72	37.11
V	9568.0	54.305804	12.707893	25.36	41.7400	52.080	66.54	81.56
AP	9568.0	1013.259078	5.938784	992.89	1009.1000	1012.940	1017.26	1033.30
RH	9568.0	73.308978	14.600269	25.56	63.3275	74.975	84.83	100.16
PE	9568.0	454.365009	17.066995	420.26	439.7500	451.550	468.43	495.76

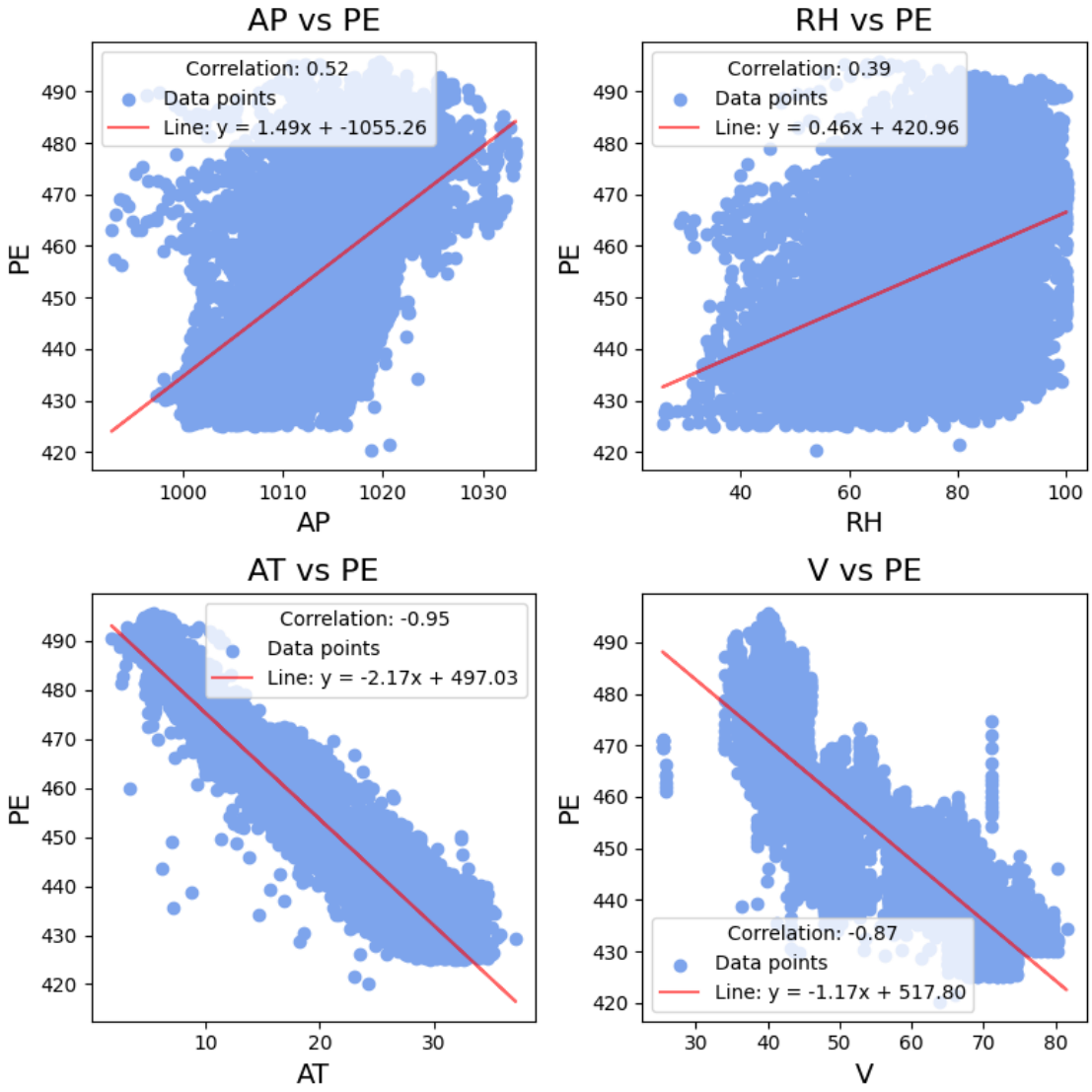
[5]:
```
# Check for null values
null_values = df.isnull().sum()

# Check for duplicates
duplicates = df.duplicated().sum()

print(f'\033[4mNull values:\033[0m\n{null_values}\n')
print(f'\033[4mDuplicates:\033[0m\n{duplicates}')
```

Null values:
```
AT    0
V     0
AP    0
RH    0
PE    0
dtype: int64
```

Duplicates:
```
41
```

4 Regression plots for all feature variables

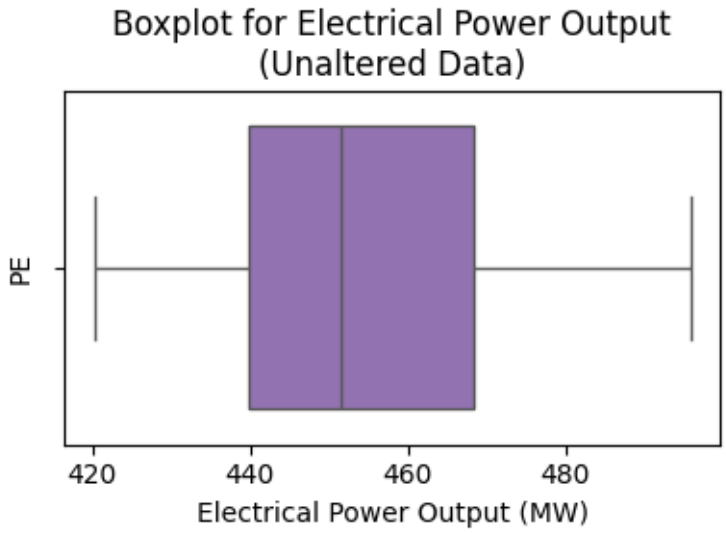
[6]:
```
# Define the independent variables
independent_vars = ['AP', 'RH', 'AT', 'V']
y_name = 'PE'

# Create a figure for all plots
plt.figure(figsize=(8, 8)) # (height, width)

# Loop through independent variables and create plots
for i, x_name in enumerate(independent_vars):
    plt.subplot(2, 2, i + 1)  # 2x2 grid of plots
    x = df[x_name]
    y = df[y_name]

    plt.scatter(x, y, color='#7da4ec', label='Data points')

    # Fit a linear regression line (y = mx + b)
    m, b = np.polyfit(x, y, 1)

    # Plot the regression line
    plt.plot(x, m*x + b, color='red', alpha=0.6, label=f'Line: y = {m:.2f}x +␣
 ↪{b:.2f}')

    # Calculate correlation
    correlation = np.corrcoef(x, y)[0, 1]

    # Add the correlation value to the legend
    plt.legend(title=f'Correlation: {correlation:.2f}')

    # Add labels and title
    plt.xlabel(x_name, size=14)
    plt.ylabel(y_name, size=14)
    plt.title(f'{x_name} vs {y_name}', size=16)

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```



5 Box Plot of Electrical Power Output (PE)

[7]:
```
# Normal PE boxplot, renamed columns for readablility
df.columns=[
    'Ambient_Temperature',
    'Vacuum',
    'Atmospheric_Pressure',
    'Relative_Humidity',
    'Electrical_Power_Output'
]
plt.figure(figsize=(4, 3))
sns.boxplot(x=df['Electrical_Power_Output'], color='#9467bd',orient='h')
```

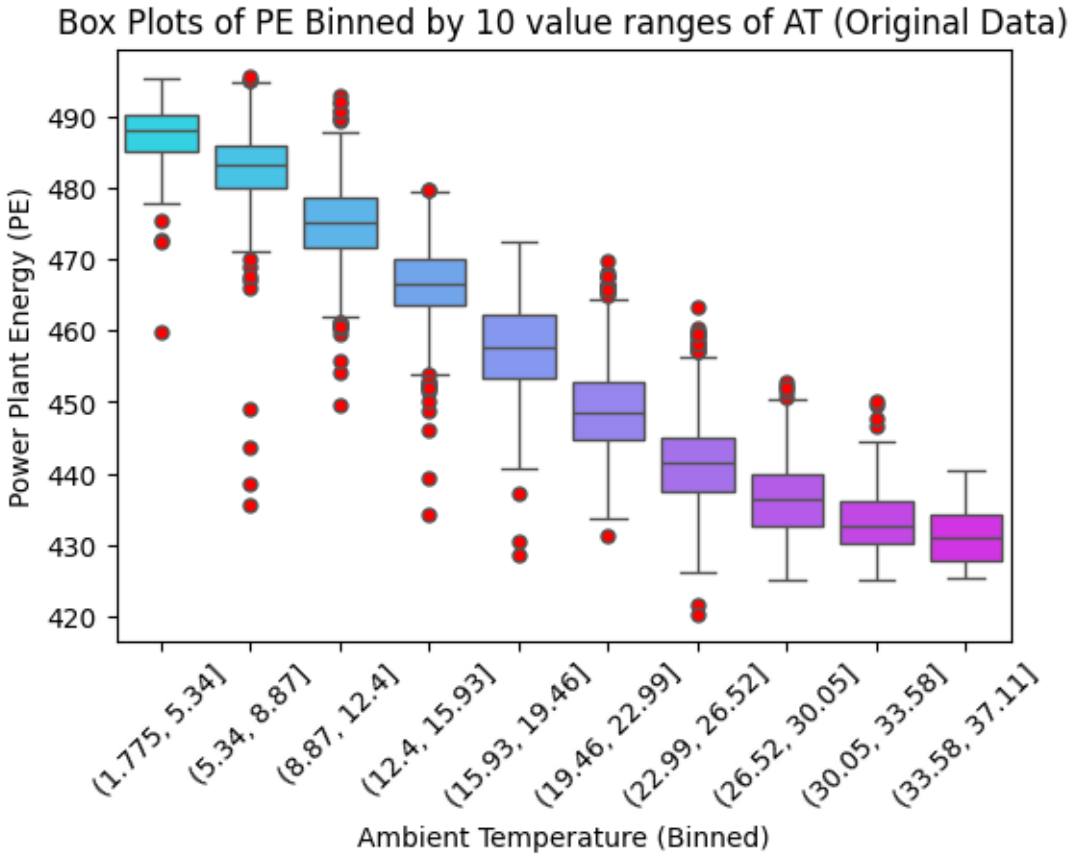
```
plt.title(f'Boxplot for Electrical Power Output\n(Unaltered Data)')
plt.ylabel('PE')
plt.xlabel('Electrical Power Output (MW)')
plt.tight_layout()
plt.show()
```



6 Box plots of PE binned by AT

[8]:
```
# Create a copy of the original df to avoid altering it
binned_df = df.copy()
# Make a column with 10 bins of Ambient Temperature values
binned_df['Ambient_Temperature_bin'] = pd.cut(binned_df['Ambient_Temperature'],␣
 ↪bins=10)

# Red outliers
flierprops = dict(marker='o', markerfacecolor='red', markersize=5,␣
 ↪linestyle='none')

# Plot PE vs binned AT, with red outliers
plt.figure(figsize=(6, 4))
sns.boxplot(
    x='Ambient_Temperature_bin',
    y='Electrical_Power_Output',
    data=binned_df,
    palette='cool',
    hue='Ambient_Temperature_bin',
    legend=False,
```

```
    flierprops=flierprops
)
plt.title('Box Plots of PE Binned by 10 value ranges of AT (Original Data)')
plt.xlabel('Ambient Temperature (Binned)')
plt.ylabel('Power Plant Energy (PE)')
plt.xticks(rotation=45, size=10)

# Show the plot
plt.show()
```



[9]:
```
# Function to remove outliers from target var, based on feature
def remove_outliers(df, feature, target):
    # Define Q1,Q3,IQR
    Q1 = df.groupby(feature, observed=False)[target].quantile(0.25)
    Q3 = df.groupby(feature, observed=False)[target].quantile(0.75)
    IQR = Q3 - Q1
    # Define lower and upper limit for removal, adjust scale factor integer as␣
 ↪needed
```

```
    # Using a scale factor (like IQR) allows for adjustment based on the␣
 ↪dataset's characteristics,
    # making the outlier detection process more flexible and context-sensitive
    lower_bound = Q1 - 1.0 * IQR
    upper_bound = Q3 + 1.0 * IQR
    # Merge bounds to original dataframe
    bounds_df = pd.DataFrame({'lower_bound': lower_bound, 'upper_bound':␣
 ↪upper_bound})
    df = df.merge(bounds_df, how='left', left_on=feature, right_index=True)
    # Outlier index tracking
    outlier_indices = df[(df[target] < df['lower_bound']) | (df[target] >␣
 ↪df['upper_bound'])].index
    # Make dataframe without outliers, keep less than upper and above lower␣
 ↪bounds
    df_filtered = df[(df[target] >= df['lower_bound']) & (df[target] <=␣
 ↪df['upper_bound'])]
    # Drop the columns with bounds
    df_filtered = df_filtered.drop(['lower_bound', 'upper_bound'], axis=1)

    return df_filtered, outlier_indices
```

7 Outlier remove function (1.0 scale factor)

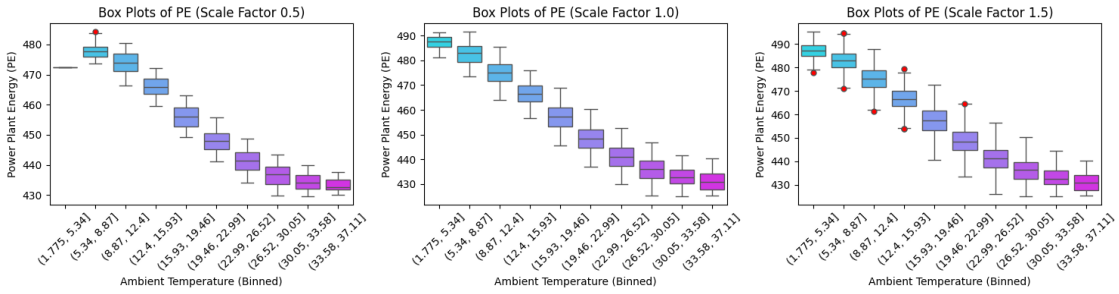
[10]:
```
# Function to remove outliers from target var, based on feature, with scale␣
 ↪factor
def remove_outliers(df, feature, target, scale_factor=1.0):
    # Define Q1, Q3, IQR
    # Q1: The 25th percentile of the target for each group defined by feature
    Q1 = df.groupby(feature, observed=False)[target].quantile(0.25)
    # Q3: The 75th percentile of the target for each group defined by feature
    Q3 = df.groupby(feature, observed=False)[target].quantile(0.75)
    IQR = Q3 - Q1

    # Define lower and upper limit for removal, using the scale factor
    lower_bound = Q1 - scale_factor * IQR
    upper_bound = Q3 + scale_factor * IQR

    # Merge bounds to original dataframe
    bounds_df = pd.DataFrame({'lower_bound': lower_bound, 'upper_bound':␣
 ↪upper_bound})
    df = df.merge(bounds_df, how='left', left_on=feature, right_index=True)

    # Outlier index tracking
    outlier_indices = df[(df[target] < df['lower_bound']) | (df[target] >␣
 ↪df['upper_bound'])].index
```

```
    # Make dataframe without outliers,
    # Keep values less than upper bounds and keep values above lower bounds
    df_filtered = df[(df[target] >= df['lower_bound']) & (df[target] <=␣
 ↪df['upper_bound'])]

    # Drop the columns with bounds
    df_filtered = df_filtered.drop(['lower_bound', 'upper_bound'], axis=1)

    return df_filtered, outlier_indices
```

8 Test for various scale factors during binned PE vs AT outlier removal:

[11]:
```
# Create a copy of the original df to avoid altering it
binned_df_test = df.copy()

# Feature bin creation in the new dataframe
binned_df_test['Ambient_Temperature_bin'] = pd.
 ↪cut(binned_df_test['Ambient_Temperature'], bins=10)
binned_df_test['Relative_Humidity_bin'] = pd.
 ↪cut(binned_df_test['Relative_Humidity'], bins=10)
binned_df_test['Atmospheric_Pressure_bin'] = pd.
 ↪cut(binned_df_test['Atmospheric_Pressure'], bins=10)
binned_df_test['Vacuum_bin'] = pd.cut(binned_df_test['Vacuum'], bins=10)

# List of binned features
binned_features = [
    'Atmospheric_Pressure_bin',
    'Relative_Humidity_bin',
    'Ambient_Temperature_bin',
    'Vacuum_bin'
]

# Different scale factors to test
scale_factors = [0.5,1.0, 1.5]

# Set up subplots
plt.figure(figsize=(15, 4))
plot_number = 1

# Red outliers
flierprops = dict(marker='o', markerfacecolor='red', markersize=5,␣
 ↪linestyle='none')

for scale in scale_factors:
    # Copy the binned dataframe for outlier removal
```

```
    df_outliers_removed_test = binned_df_test.copy()
    removed_indices = []  # Set an empty list for removed_indices before adding␣
 ↪to list

    # Apply remove_outliers() to each binned feature for the current scale␣
 ↪factor
    for feature in binned_features:
        df_outliers_removed_test, outliers = remove_outliers(
            df_outliers_removed_test,
            feature,
            'Electrical_Power_Output',
            scale_factor=scale
)
        removed_indices.extend(outliers.tolist())
    # Plot the result for this scale factor as a subplot (1 row x 3 plots)
    plt.subplot(1, 3, plot_number)
    sns.boxplot(
        x='Ambient_Temperature_bin',
        y='Electrical_Power_Output',
        data=df_outliers_removed_test,
        palette='cool',
        hue='Ambient_Temperature_bin',
        legend=False,
        flierprops=flierprops
)
    plt.title(f'Box Plots of PE (Scale Factor {scale})')
    plt.xlabel('Ambient Temperature (Binned)')
    plt.ylabel('Power Plant Energy (PE)')
    plt.xticks(rotation=45, size=10)
    # Move to next subplot
    plot_number += 1

# Show all subplots
plt.tight_layout()
plt.show()
```



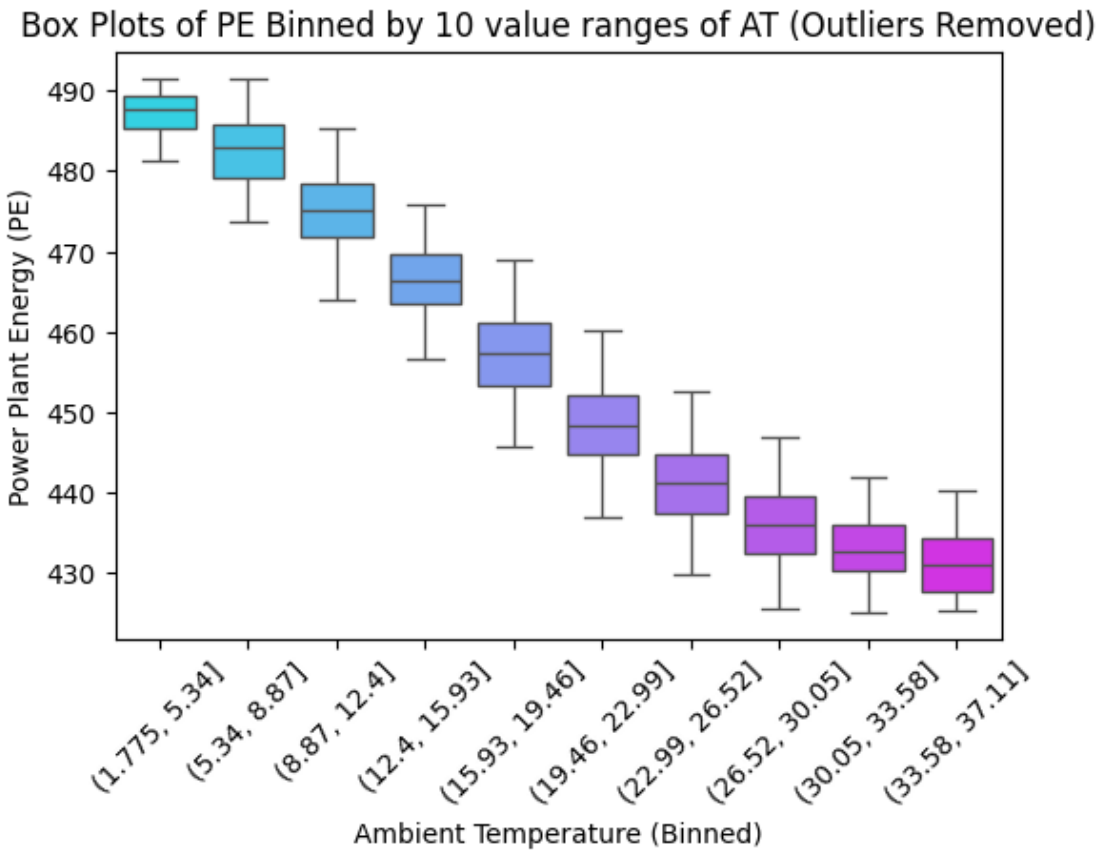
9 Bin PE by all features, then remove all outliers

[12]:
```
# Bin and remove outliers
# Create a copy of the original df to avoid altering it
binned_df = df.copy()

# Feature bin creation in the new dataframe
binned_df['Ambient_Temperature_bin'] = pd.cut(binned_df['Ambient_Temperature'],␣
 ↪bins=10)
binned_df['Relative_Humidity_bin'] = pd.cut(binned_df['Relative_Humidity'],␣
 ↪bins=10)
binned_df['Atmospheric_Pressure_bin'] = pd.
 ↪cut(binned_df['Atmospheric_Pressure'], bins=10)
binned_df['Vacuum_bin'] = pd.cut(binned_df['Vacuum'], bins=10)

# List of binned features
binned_features = [
    'Atmospheric_Pressure_bin',
    'Relative_Humidity_bin',
    'Ambient_Temperature_bin',
    'Vacuum_bin'
]
df_outliers_removed = binned_df.copy() # copy binned df to remove outliers from
removed_indices = [] # Set an empty list for removed_indices before adding to␣
 ↪list

# Apply outlier_removal() to each binned feature
for feature in binned_features:
    df_outliers_removed, outliers = remove_outliers(df_outliers_removed,␣
 ↪feature, 'Electrical_Power_Output')
    removed_indices.extend(outliers.tolist())

# Outlier and index checks
print(f"Count of total removed binned PE outliers: {len(removed_indices)}")
remaining_indices = set(removed_indices).intersection(df_outliers_removed.index)
if remaining_indices:
    print(f"Original binned PE outliers remaining after removal:␣
 ↪{list(remaining_indices)}")
else:
    print("No original binned PE outliers remaining.")
```

```
Count of total removed binned PE outliers: 1256
No original binned PE outliers remaining.
```

10 Plot of removed outliers (AT vs PE)

[13]:
```
# Create a copy of the df_outliers_removed to avoid altering it
binned_df = df_outliers_removed.copy()
binned_df['Ambient_Temperature_bin'] = pd.cut(binned_df['Ambient_Temperature'],␣
 ↪bins=10)

plt.figure(figsize=(6, 4))

# Plot and labels
sns.boxplot(
    x='Ambient_Temperature_bin',
    y='Electrical_Power_Output',
    data=binned_df,
    palette='cool',
    hue='Ambient_Temperature_bin',
    legend=False
)
plt.title('Box Plots of PE Binned by 10 value ranges of AT (Outliers Removed)')
plt.xlabel('Ambient Temperature (Binned)')
plt.ylabel('Power Plant Energy (PE)')
plt.xticks(rotation=45, size=10)
plt.show()
```



[14]:
```
df_outliers_removed.describe().T
```

[14]:

	count	mean	std	min	25%	\
Ambient_Temperature	8312.0	20.413526	7.207974	1.81	14.350	
Vacuum	8312.0	55.425673	12.647239	25.36	42.340	
Atmospheric_Pressure	8312.0	1013.344418	6.008049	992.89	1009.200	
Relative_Humidity	8312.0	73.314429	14.241457	25.56	63.755	
Electrical_Power_Output	8312.0	452.400439	16.302910	425.11	438.730	

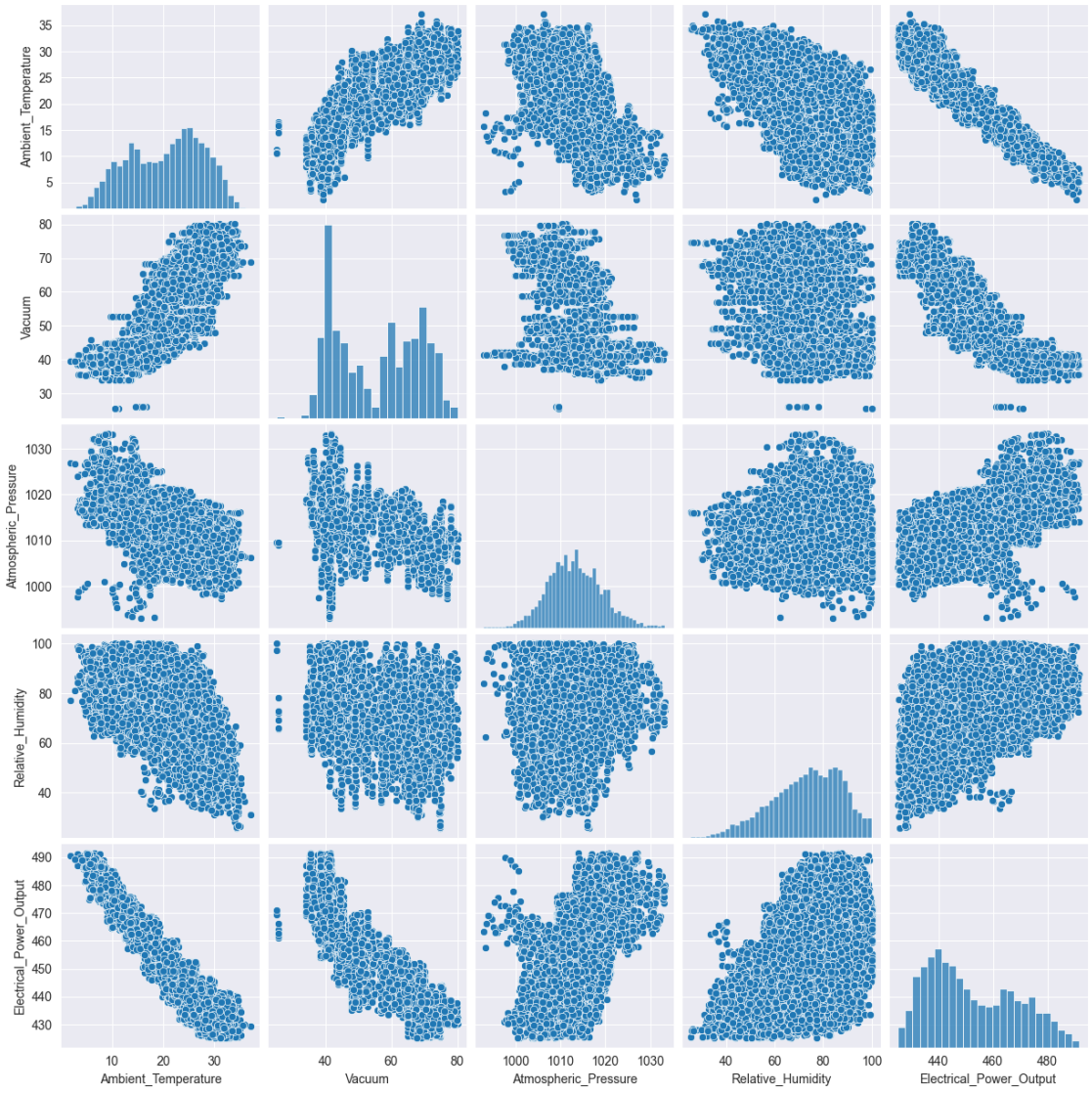
	50%	75%	max
Ambient_Temperature	21.40	26.17	37.11
Vacuum	57.25	67.45	80.18
Atmospheric_Pressure	1013.08	1017.39	1033.30
Relative_Humidity	74.85	84.42	100.16
Electrical_Power_Output	448.79	465.78	491.54

[15]:
```
# Define each dataframe's features for analysis
original_df_features = df[[
                        'Ambient_Temperature',
                        'Atmospheric_Pressure',
```

```
                        'Relative_Humidity',
                        'Vacuum'
]]

no_out_features = df_outliers_removed[[
                      'Ambient_Temperature',
                      'Atmospheric_Pressure',
                      'Relative_Humidity',
                      'Vacuum'
]]
```

[16]:
```
sns.set_style('darkgrid')
sns.pairplot(df_outliers_removed)
```

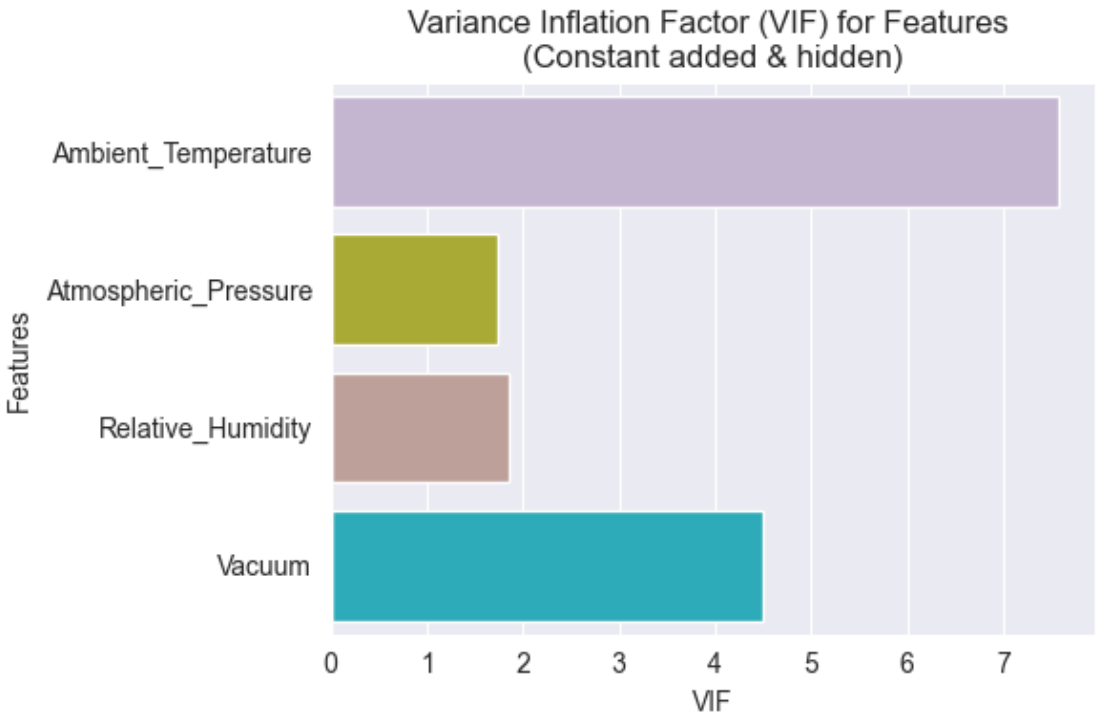
[16]: <seaborn.axisgrid.PairGrid at 0x1b8bd27a150>



11 Calculate VIF(X_i)=1 / 1- R_i^2

[17]:
```
# Define feature variables (AP, RH, AT, V)
X = no_out_features
# Add a constant for the intercept in the linear regression
X = sm.add_constant(X)
y = df['Electrical_Power_Output']
plt.figure(figsize=(6, 4))

# Calculate VIF for each feature
vif = pd.DataFrame()
vif["Feature"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Filter out the constant from the plot
vif = vif[vif["Feature"] != "const"]

# Plot VIF values as a bar chart
plt.figure(figsize=(6, 4))
sns.barplot(x='VIF', y='Feature', data=vif,palette='tab20',hue=y,legend=False)
plt.title("Variance Inflation Factor (VIF) for Features \n(Constant added &␣
 ↪hidden)")
plt.xlabel("VIF")
plt.ylabel("Features")
plt.tight_layout()
plt.show()
```

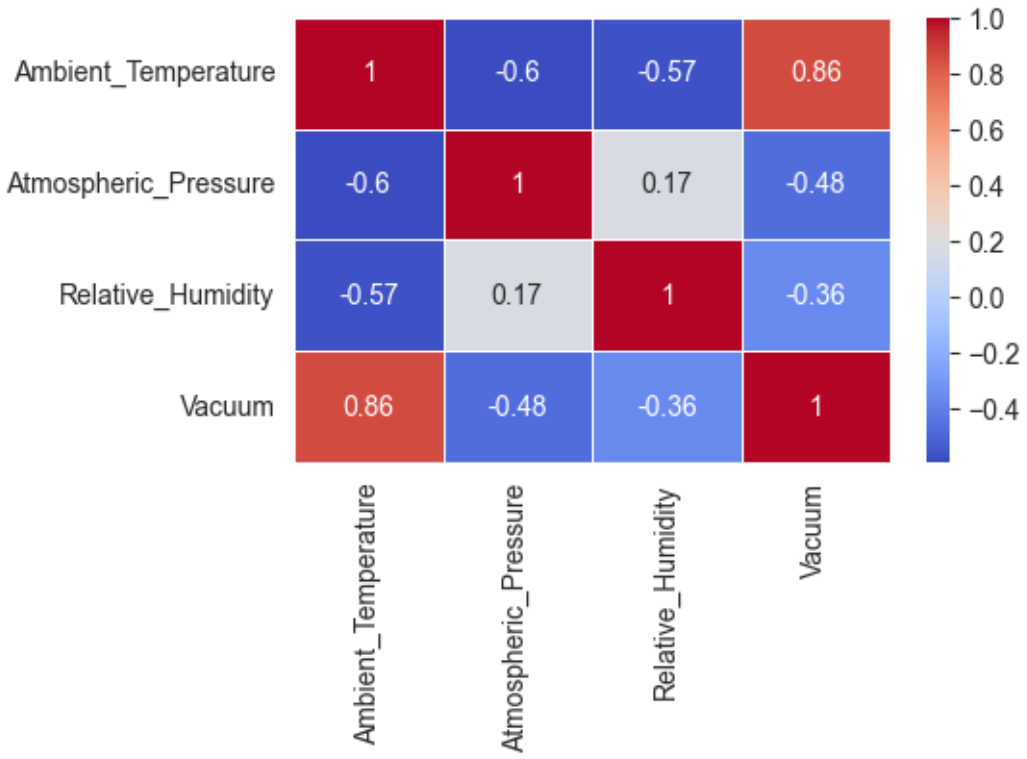
```
<Figure size 600x400 with 0 Axes>
```



12 Feature Correlations

[18]:
```
# Correlation between the features
plt.figure(figsize=(5, 3))
sns.heatmap(no_out_features.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
```

[18]: <Axes: >



13 Train OLS model

[19]:
```
df_outliers_removed.head()
```

[19]:
```
   Ambient_Temperature  Vacuum  Atmospheric_Pressure  Relative_Humidity  \
1                23.64   58.49                1011.40              74.20
2                29.74   56.90                1007.15              41.91
3                19.07   49.69                1007.22              76.79
4                11.80   40.66                1017.13              97.20
5                13.97   39.16                1016.05              84.60

   Electrical_Power_Output Ambient_Temperature_bin Relative_Humidity_bin  \
1                   445.75          (22.99, 26.52]        (70.32, 77.78]
2                   438.76          (26.52, 30.05]        (40.48, 47.94]
3                   453.09          (15.93, 19.46]        (70.32, 77.78]
4                   464.43            (8.87, 12.4]        (92.7, 100.16]
5                   470.96           (12.4, 15.93]        (77.78, 85.24]

  Atmospheric_Pressure_bin      Vacuum_bin
1     (1009.054, 1013.095]  (53.46, 59.08]
2     (1005.013, 1009.054]  (53.46, 59.08]
3     (1005.013, 1009.054]  (47.84, 53.46]
```

```
4     (1013.095, 1017.136]   (36.6, 42.22]
5     (1013.095, 1017.136]   (36.6, 42.22]
```

[20]:
```
X = no_out_features
y = df_outliers_removed['Electrical_Power_Output']
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Electrical_Power_Output   R-squared:
0.938
Model:                                OLS   Adj. R-squared:
0.937
Method:                     Least Squares   F-statistic:
3.116e+04
Date:                    Mon, 21 Oct 2024   Prob (F-statistic):
0.00
Time:                            19:42:28   Log-Likelihood:
-23472.
No. Observations:                   8312   AIC:
4.695e+04
Df Residuals:                       8307   BIC:
4.699e+04
Df Model:                              4
Covariance Type:                nonrobust
==============================================================================
========
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const                    412.2340     10.153     40.602      0.000     392.332
432.136
Ambient_Temperature       -1.9009      0.017   -111.308      0.000      -1.934
-1.867
Atmospheric_Pressure       0.1009      0.010     10.272      0.000       0.082
0.120
Relative_Humidity         -0.1358      0.004    -31.759      0.000      -0.144
-0.127
Vacuum                    -0.2397      0.008    -31.926      0.000      -0.254
-0.225
==============================================================================
Omnibus:                      130.044   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               99.000
```

```
Skew:                          0.179   Prob(JB):                     3.18e-22
Kurtosis:                      2.602   Cond. No.                     2.31e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.31e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

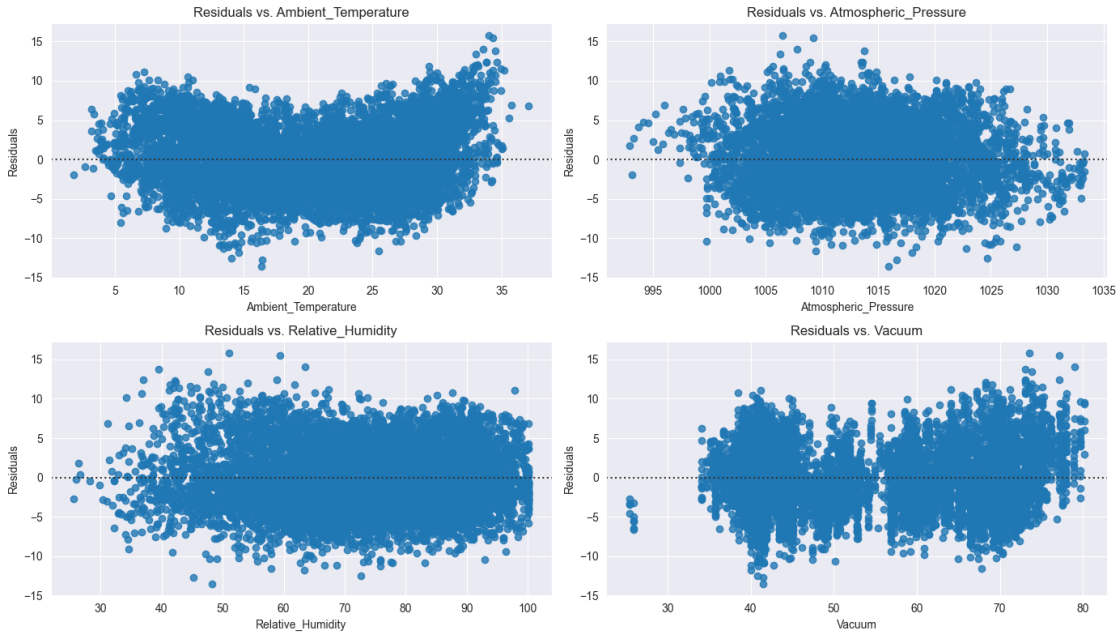
14 Variable Combination Check

14.0.1 Residuals should be homoscedastic (constant variance)

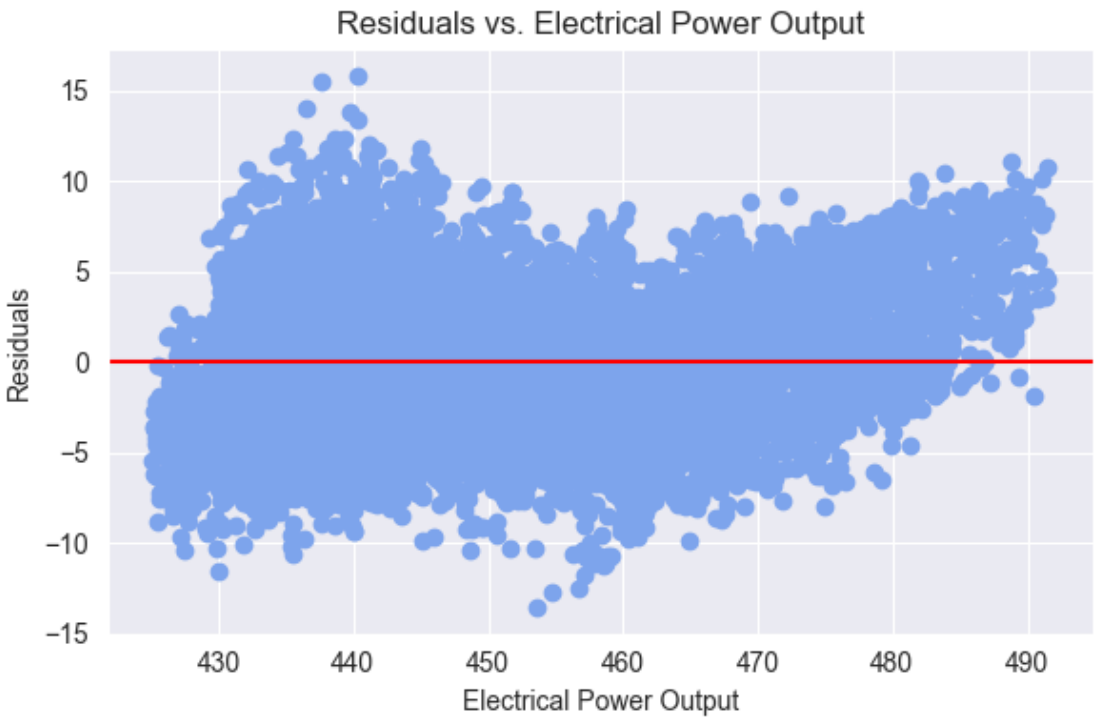
Step : check the residuals vs fitted plot. if the residuals are randomly scattered around the 0 line, the residuals are homoscedastic. if the residuals have a pattern, the residuals are heteroscedastic.

From the plot, we can see that the residuals are randomly scattered around the 0 line, so the **residuals are homoscedastic.**

[21]:
```
X = no_out_features
# Number of columns (features)
n_features = len(X.columns)

# Create subplots: number of rows and columns (adjust depending on how many␣
 ↪plots you want per row)
n_rows = (n_features + 1) // 2  # Arrange 2 plots per row
fig, axes = plt.subplots(n_rows, 2, figsize=(14, 4 * n_rows))  # Adjust figsize␣
 ↪based on the number of plots

# Flatten the axes for easy iteration
axes = axes.flatten()

# Plot residuals for each feature
for idx, i in enumerate(X.columns):
    sns.residplot(x=X[i], y=est2.resid, ax=axes[idx])  # Use ax parameter to␣
 ↪plot in the correct subplot
    axes[idx].set_xlabel(i)
    axes[idx].set_ylabel('Residuals')
    axes[idx].set_title(f'Residuals vs. {i}')

# Remove any unused subplots if the number of features is odd
if n_features % 2 != 0:
    fig.delaxes(axes[-1])  # Remove the last subplot if not needed

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



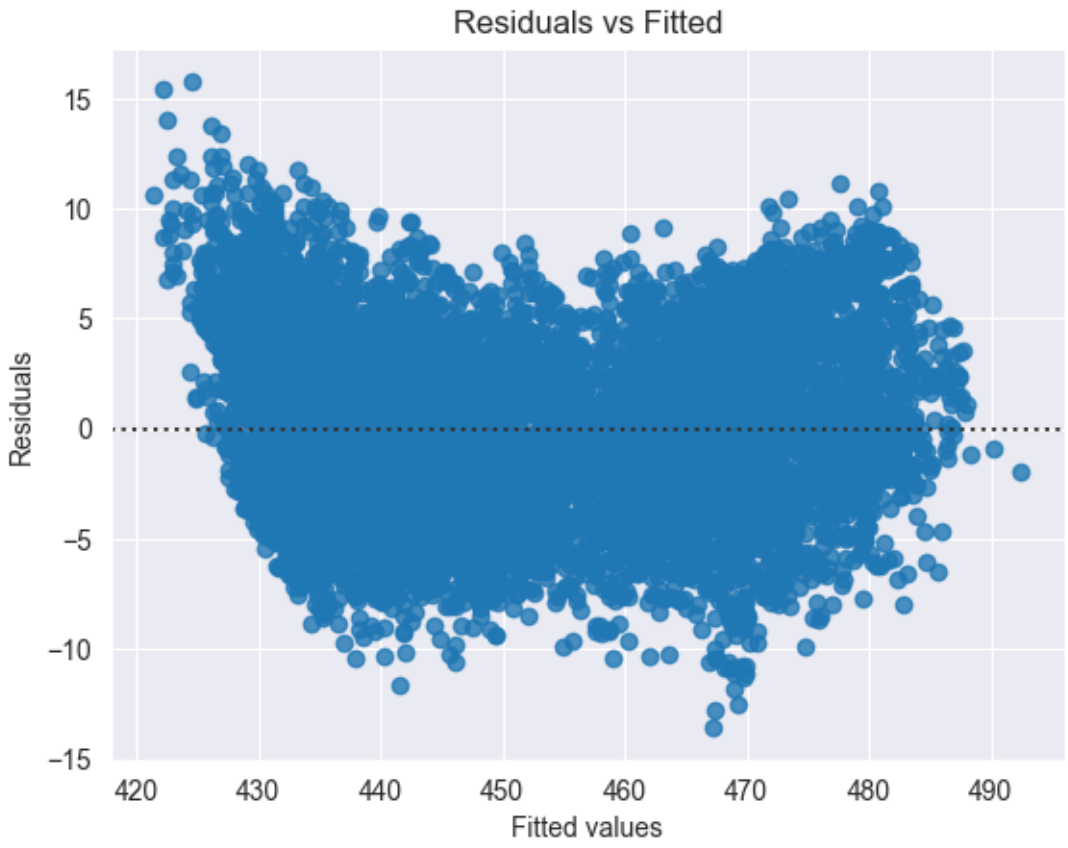
[22]:
```
## Residuals vs Y
plt.figure(figsize=(6, 4))
plt.scatter(df_outliers_removed['Electrical_Power_Output'], est2.resid,␣
 ↪color='#7da4ec')
plt.axhline(y=0, color='r', linestyle='-')
plt.title('Residuals vs. Electrical Power Output')
plt.xlabel('Electrical Power Output')
plt.ylabel('Residuals')
plt.tight_layout()
```



15 Check the residuals vs predictor plot

If the residuals are randomly scattered around the 0 line, the residuals are homoscedastic. if the residuals have a pattern, the residuals are heteroscedastic. From the plot, we can see that the residuals are randomly scattered around the 0 line, so the **residuals are homoscedastic**. So, the model satisfies the assumption of homoscedasticity.

[23]:
```
import seaborn as sns
sns.residplot(x=est2.fittedvalues, y=est2.resid)
plt.title('Residuals vs Fitted')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.show()
```

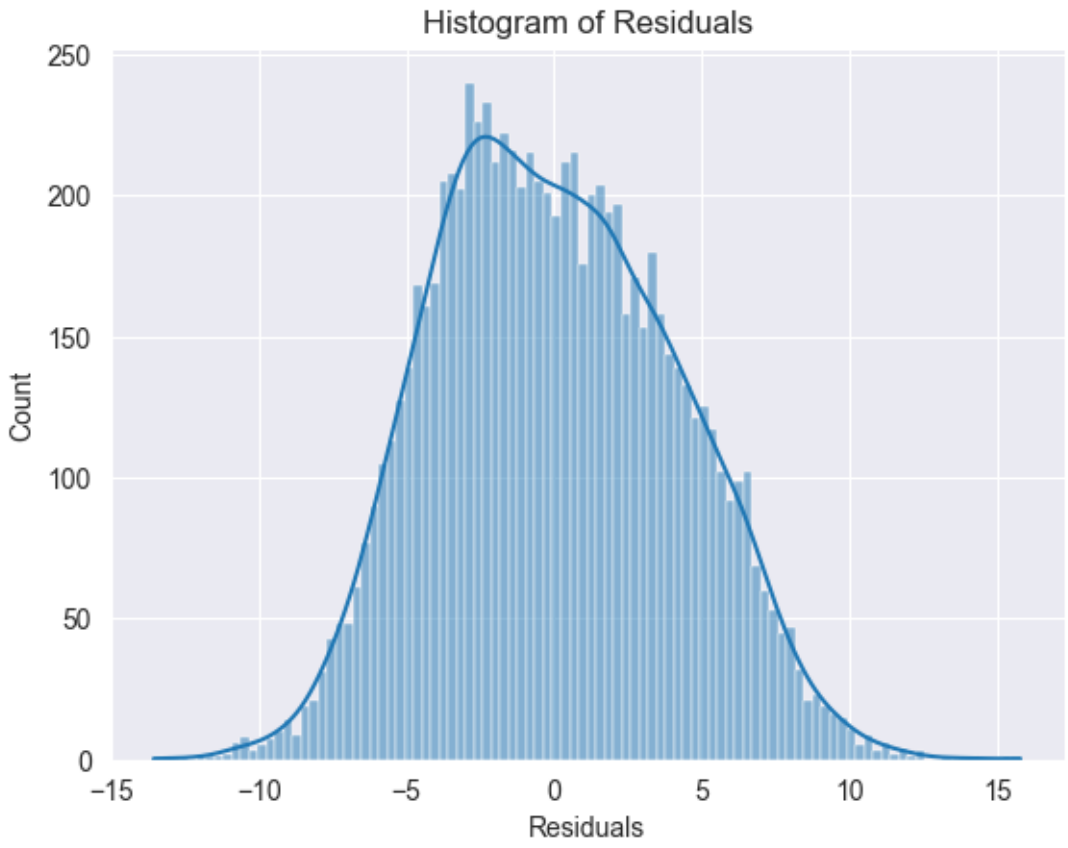


15.0.1 Residuals should be normally distributed

Step: Check the distribution of the residuals.

From the histogram, we can see that the residuals are normally distributed.

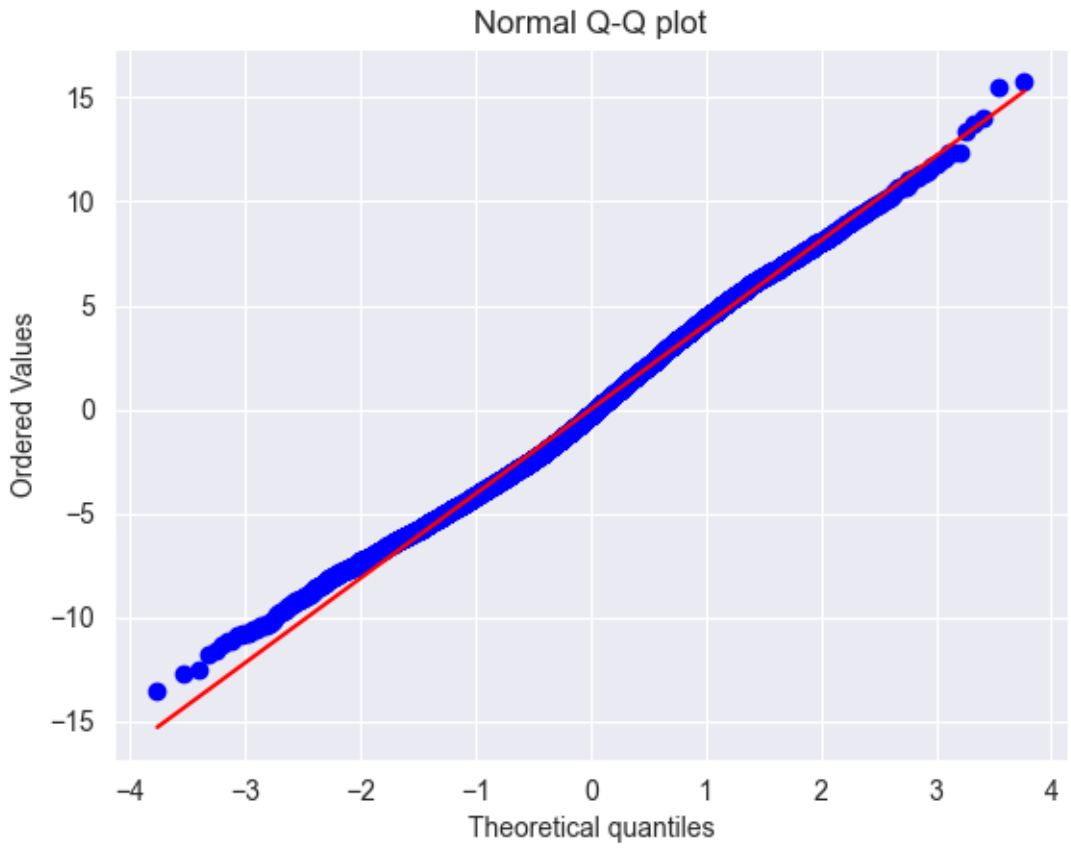
[24]:
```
import seaborn as sns
sns.histplot(est2.resid, kde=True, bins=100)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.show()
```



15.0.2 Check the QQ plot of the residuals

From the QQ plot, we can see that the residuals are normally distributed. How to interpret the QQ plot: - If the residuals are normally distributed, the points in the QQ plot should fall on the straight line. In the below QQ plot, we can see that the points are falling on the straight line, so the residuals are normally distributed.

[25]:
```
import scipy.stats as stats
stats.probplot(est2.resid, dist="norm", plot=plt)
plt.title("Normal Q-Q plot")
plt.show()
```



16 F-Test

Null Hypothesis: The model is not statistically significant

Alternative Hypothesis: The model is statistically significant

[26]:
```
X= no_out_features
X= sm.add_constant(X)
y = df_outliers_removed['Electrical_Power_Output']

model=sm.OLS(y, X).fit()

# F test
f_test_results=model.f_test(
    'Relative_Humidity=0,'
    'Vacuum=0,'
    'Atmospheric_Pressure=0,'
    'Ambient_Temperature=0'
)
print(f_test_results)
```

```
# F test p-value
f_test_results.pvalue
# p-value is less than 0.05, so we reject the null hypothesis and conclude
# that the model is statistically significant
if f_test_results.pvalue<0.05:
    print('Reject the null hypothesis')
else:
    print('Fail to reject the null hypothesis')
```

```
<F test: F=31155.131866986307, p=0.0, df_denom=8.31e+03, df_num=4>
Reject the null hypothesis
```

17 Model Comparison Class

[27]:
```
##############################################################################################
#
#   Base Class for the Model
#
##############################################################################################
class Model:
    def __init__(
        self,
        X,y,
        test_size=0.2,
        random_state=42,
        val_size=0.25,
        model="LinearRegression"
):
        '''
        Parameters:
        X: Features
        y: Target
        test_size: Test size
        random_state: Random state
        val_size: Validation size
        model: Model to be used
        '''
        self.model_name = model
        self.models = {
            "LinearRegression": {
                "model": lambda X, y: LinearRegression(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "OLS": {
                "model":lambda X, y: sm.OLS(y, X),
                "fit": lambda X, y: self.model.fit()
            },
```

```
            "DecisionTreeRegressor": {
                "model": lambda X, y: DecisionTreeRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "RandomForestRegressor": {
                "model": lambda X, y: RandomForestRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "GradientBoostingRegressor": {
                "model": lambda X, y: GradientBoostingRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "AdaBoostRegressor": {
                "model": lambda X, y: AdaBoostRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "BaggingRegressor": {
                "model": lambda X, y: BaggingRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            },
            "SVR": {
                "model": lambda X, y: SVR(),
                "fit": lambda X, y: self.model.fit(X, y)
            }
            ,
            "MLPRegressor": {
                "model": lambda X, y: MLPRegressor(),
                "fit": lambda X, y: self.model.fit(X, y)
            }
        }
        self.X = X
        self.y = y
        self.test_size = test_size
        self.random_state = random_state
        self.val_size = val_size
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            self.X,
            self.y,
            test_size=self.test_size,
            random_state=self.random_state
)
        self.X_train, self.X_val, self.y_train, self.y_val = train_test_split(
            self.X_train,
            self.y_train,
            test_size=self.val_size,
            random_state=self.random_state
)
```

```
        self.model = self.models[self.model_name]['model'](self.X_train, self.
 ↪y_train)

    @staticmethod
    def get_models():
        return [
            "LinearRegression",
            "OLS",
            "DecisionTreeRegressor",
            "RandomForestRegressor",
            "GradientBoostingRegressor",
            "AdaBoostRegressor",
            "BaggingRegressor",
            "SVR",
            "MLPRegressor"
]

    def train(self):
        self.model=self.models[self.model_name]['fit'](self.X_train, self.
 ↪y_train)
        return self.evaluate(self.X_train, self.y_train)

    def predict(self, X):
        return self.model.predict(X)

    def validate(self):
        return self.evaluate(self.X_val, self.y_val)

    def test(self):
        return self.evaluate(self.X_test, self.y_test)

    def evaluate(self, X, y):
        y_pred = self.predict(X)
        mse = mean_squared_error(y, y_pred)
        mae = mean_absolute_error(y, y_pred)
        r2 = r2_score(y, y_pred)
        return np.round((mse, mae, r2), decimals=2)

    def summary(self):
        return self.model.summary()

    def plotResiduals(self, X, y):
        residuals = y - self.predict(X)
        sns.scatterplot(x=self.predict(X), y=residuals)
        plt.xlabel('Predictions')
        plt.ylabel('Residuals')
        plt.axhline(y=0, color='r', linestyle='--')
```

```
        plt.title('Residuals vs. Predictions')
        plt.show()

    def plot_residuals(self):
        self.plotResiduals(self.X_val, self.y_val)

    def plot_residuals_test(self):
        self.plotResiduals(self.X_test, self.y_test)

    def plot_residuals_val(self):
        self.plotResiduals(self.X_val, self.y_val)
```

18 Multiple Model comparison (outliers removed)

[28]:
```
import matplotlib.pyplot as plt
from matplotlib.table import Table
X = no_out_features
y = df_outliers_removed['Electrical_Power_Output']
# Collect the data into a list of lists
data = [['Model',
         'Train MSE',
         'Train MAE',
         'Train R²',
         'Validation MSE',
         'Validation MAE',
         'Validation R²',
         'Test MSE',
         'Test MAE',
         'Test R²'
]]

for model_name in Model.get_models():
    model = Model(X, y, model=model_name)
    train_results = model.train()
    validation_results = model.validate()
    test_results = model.test()

    # Append each row of model results to the data list
    data.append([model_name,
                 f'{train_results[0]:.4f}', f'{train_results[1]:.4f}',␣
 ↪f'{train_results[2]:.4f}',
                 f'{validation_results[0]:.4f}', f'{validation_results[1]:.
 ↪4f}', f'{validation_results[2]:.4f}',
                 f'{test_results[0]:.4f}', f'{test_results[1]:.4f}',␣
 ↪f'{test_results[2]:.4f}'])
```

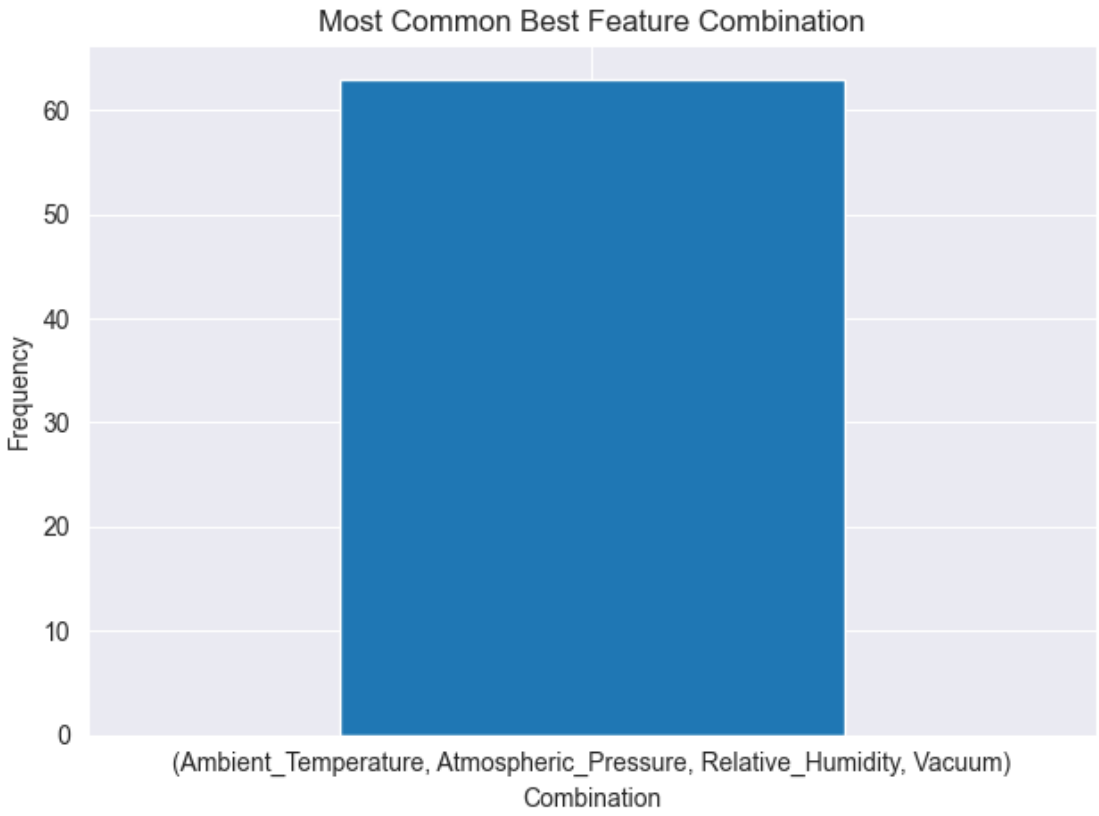
```
# Plotting the table
fig, ax = plt.subplots(figsize=(22, 2.5))  # Create a figure and a set of␣
 ↪subplots
ax.axis('tight')
ax.axis('off')

# Create a table plot
table = ax.table(cellText=data, colLabels=None, cellLoc='center', loc='center')

# Set font size for the table
table.auto_set_font_size(False)
table.set_fontsize(10)

# Adjust column width for better readability
table.scale(1.2, 1.2)

# Display the table plot
plt.show()
```

Model	Train MSE	Train MAE	Train R^2	Validation MSE	Validation MAE	Validation R^2	Test MSE	Test MAE	Test R^2
LinearRegression	16.7000	3.3700	0.9400	16.7900	3.3700	0.9400	16.1800	3.3100	0.9400
OLS	20.0400	3.6500	0.9200	19.8500	3.6200	0.9300	19.5800	3.6000	0.9300
DecisionTreeRegressor	0.0000	0.0000	1.0000	15.8600	2.9200	0.9400	16.1600	2.9400	0.9400
RandomForestRegressor	1.2200	0.8400	1.0000	8.8100	2.3000	0.9700	9.3000	2.3500	0.9600
GradientBoostingRegressor	9.7600	2.4900	0.9600	11.4800	2.7300	0.9600	11.2300	2.7100	0.9600
AdaBoostRegressor	14.0900	3.1100	0.9500	14.9500	3.1900	0.9400	14.9500	3.1700	0.9400
BaggingRegressor	1.7200	0.9400	0.9900	9.9700	2.4200	0.9600	10.2700	2.4600	0.9600
SVR	194.6500	11.3800	0.2600	199.0600	11.5000	0.2700	191.8300	11.4300	0.2800
MLPRegressor	22.6000	3.9300	0.9100	22.8900	3.9200	0.9200	22.7700	3.9200	0.9100

19 Best feature combinations

[29]:
```
# for all combinations of columns
from itertools import combinations
X = no_out_features
y = df_outliers_removed['Electrical_Power_Output']
cols = X.columns
# Store the results in  matrix
best_combinations=[]
for trial in range(100):
    results = {}
    for i in range(2, len(cols)+1):
        for comb in combinations(cols, i):
            X = df_outliers_removed[list(comb)]
            model = Model(X, y,random_state=np.random.randint(0,100))
            model.train()
            mse, mae, r2=model.validate()
            results[comb] = (mse, mae, r2)
    # Find the best combination
    best_comb = min(results, key=results.get)
    best_combinations.append(best_comb)
```

[30]:
```
# Plot best variable combination
pd.DataFrame(best_combinations).value_counts().plot(kind='bar')

# Set labels and title
plt.xlabel('Combination')
plt.ylabel('Frequency')
plt.title('Most Common Best Feature Combination')

# Rotate x-axis labels to horizontal format
plt.xticks(rotation=0, ha='center', wrap=True)  # Rotate labels horizontally␣
 ↪and center align them

# Show the plot
plt.tight_layout()
plt.show()
```



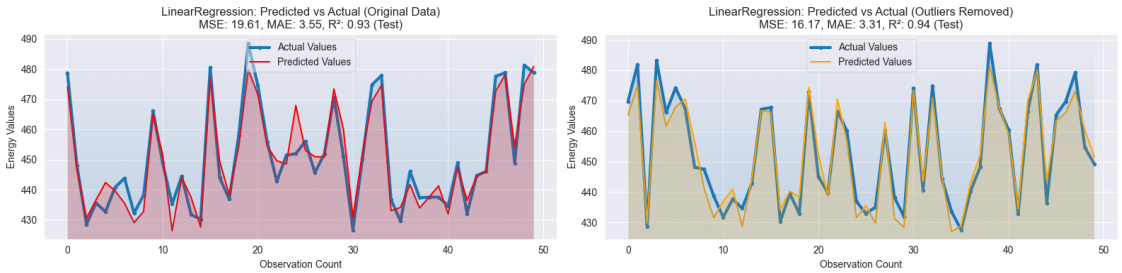
20 Prediction plots (Linear Regression)

[31]:
```
# Limit observations to plot (test set for plot)
start_obs = 150
end_obs = 200

# Set random seed for reproducibility
np.random.seed(42)
# Features (X) and target (y) - Original df
X = original_df_features
y = df['Electrical_Power_Output']

# First split: training + test sets (original df)
# Test size: 20%, training size: 80%
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.
↪2, random_state=42)

# Second split: training into training + validation sets (original df)
# split 80% training into 64% training and 16% validation.
X_train, X_val, y_train, y_val = train_test_split(
```

```
    X_train_full,
    y_train_full,
    test_size=0.2,
    random_state=42
)
# Initialize and fit the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set (original df)
y_test_predictions = model.predict(X_test)

# Calculate performance metrics on the test set
mse_test = mean_squared_error(y_test, y_test_predictions)
mae_test = mean_absolute_error(y_test, y_test_predictions)
r2_test = r2_score(y_test, y_test_predictions)

# Now for the outliers removed DataFrame (df_outliers_removed)
# Features (X) and target (y) - Outliers removed
X_outliers = no_out_features
y_outliers = df_outliers_removed['Electrical_Power_Output']

# Train-test split for outliers removed
X_train_full_outliers, X_test_outliers, y_train_full_outliers, y_test_outliers␣
 ↪= train_test_split(
    X_outliers,
    y_outliers,
    test_size=0.2,
    random_state=42
)
X_train_outliers, X_val_outliers, y_train_outliers, y_val_outliers =␣
 ↪train_test_split(
    X_train_full_outliers,
    y_train_full_outliers,
    test_size=0.2,
    random_state=42
)
# Initialize and fit the Linear Regression model on outliers removed
model_outliers = LinearRegression()
model_outliers.fit(X_train_outliers, y_train_outliers)

# Make predictions on the test set (outliers removed)
y_test_predictions_outliers = model_outliers.predict(X_test_outliers)

# Calculate performance metrics on the test set for outliers removed
mse_test_outliers = mean_squared_error(y_test_outliers,␣
 ↪y_test_predictions_outliers)
```

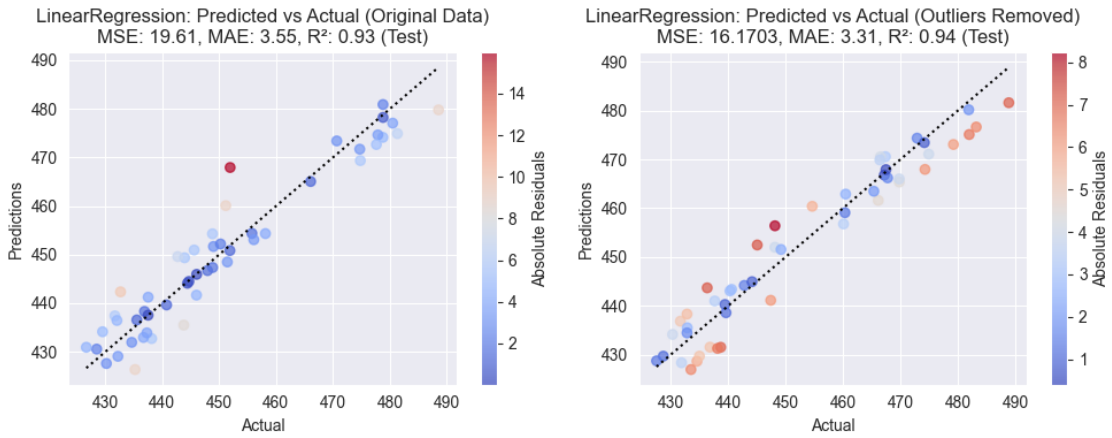
```
mae_test_outliers = mean_absolute_error(y_test_outliers,␣
 ↪y_test_predictions_outliers)
r2_test_outliers = r2_score(y_test_outliers, y_test_predictions_outliers)

# Plot size
plt.figure(figsize=(16, 4))

# Selecting part of the data for plotting (original df)
predictions_num = y_test_predictions[start_obs:end_obs]
y_test_num = y_test.iloc[start_obs:end_obs]
x_values_num = np.arange(len(predictions_num))

# Selecting part of the data for plotting (outliers removed)
predictions_num_outliers = y_test_predictions_outliers[start_obs:end_obs]
y_test_num_outliers = y_test_outliers.iloc[start_obs:end_obs]

# Plot for original df
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.plot(x_values_num, y_test_num, linewidth=3, marker='.', label='Actual␣
 ↪Values')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='top')  #␣
 ↪underglow specs
plt.plot(x_values_num, predictions_num, linewidth=1.5, label='Predicted␣
 ↪Values', color='red')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='zero') ␣
 ↪# underglow specs
plt.title(f'LinearRegression: Predicted vs Actual (Original Data)\n'
          f'MSE: {mse_test:.2f}, MAE: {mae_test:.2f}, R²: {r2_test:.2f} (Test)')
plt.xlabel('Observation Count')
plt.ylabel('Energy Values')
plt.legend(loc='upper center', framealpha=0.5)
plt.tight_layout()

# Plot for df_outliers_removed
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.plot(x_values_num, y_test_num_outliers, linewidth=3, marker='.',␣
 ↪label='Actual Values')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='top')  #␣
 ↪underglow specs
plt.plot(x_values_num, predictions_num_outliers, linewidth=1.5,␣
 ↪label='Predicted Values', color='orange')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='zero') ␣
 ↪# underglow specs
plt.title(f'LinearRegression: Predicted vs Actual (Outliers Removed)\n'
          f'MSE: {mse_test_outliers:.2f}, MAE: {mae_test_outliers:.2f}, R²:␣
 ↪{r2_test_outliers:.2f} (Test)')
```

```
plt.xlabel('Observation Count')
plt.ylabel('Energy Values')
plt.legend(loc='upper center', framealpha=0.5)
plt.tight_layout()

# Show the plots
plt.show()
```



[32]:
```
# Model used
model = LinearRegression()
model.fit(X_train, y_train)

# Train-test split sizes
test_size_original = X_test.shape[0]
test_size_outliers_removed = X_test_outliers.shape[0]

# Calculate residuals
residuals = predictions_num - y_test_num

# Calculate residuals
residuals_outliers = predictions_num_outliers - y_test_num_outliers

# Plot size
plt.figure(figsize=(10, 4))

# Plot for original data
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.scatter(y_test_num, predictions_num, c=np.abs(residuals), cmap='coolwarm',␣
 ↪alpha=0.7)
plt.colorbar(label='Absolute Residuals')
plt.title(f'LinearRegression: Predicted vs Actual (Original Data)\n'
          f'MSE: {mse_test:.2f}, MAE: {mae_test:.2f}, R²: {r2_test:.2f} (Test)')
plt.xlabel('Actual')
plt.ylabel('Predictions')
plt.plot([min(y_test_num), max(y_test_num)], [min(y_test_num),␣
 ↪max(y_test_num)], color='black', linestyle=':')
```

```
# Plot for outliers removed
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.scatter(y_test_num_outliers, predictions_num_outliers, c=np.
 ↪abs(residuals_outliers), cmap='coolwarm', alpha=0.7)
plt.colorbar(label='Absolute Residuals')
plt.title(f'LinearRegression: Predicted vs Actual (Outliers Removed)\n'
          f'MSE: {mse_test_outliers:.4f}, MAE: {mae_test_outliers:.2f}, R²:␣
 ↪{r2_test_outliers:.2f} (Test)')
plt.xlabel('Actual')
plt.ylabel('Predictions')
plt.plot(
    [min(y_test_num_outliers),
     max(y_test_num_outliers)],
    [min(y_test_num_outliers),
     max(y_test_num_outliers)],
    color='black',
    linestyle=':'
)
# Show the side-by-side plots
plt.tight_layout()
plt.show()
```



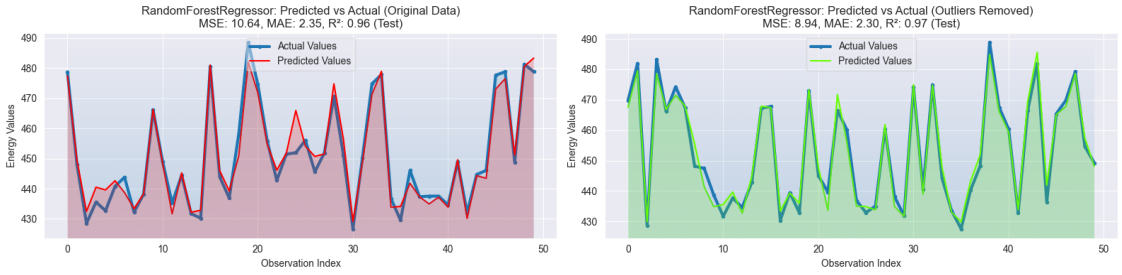
21 Prediction plots (RandomForestRegressor)

[33]:
```
# Limit observations to plot (test set for plot)
start_obs = 150
end_obs = 200

# Set random seed for reproducibility
```

```
np.random.seed(42)

# Features (X) and target (y) - Original df
X = original_df_features
y = df['Electrical_Power_Output']
# First split: training + test sets (original df)
# Test size: 20%, training size: 80%
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)

# Second split: training into training + validation sets (original df)
# split 80% training into 64% training and 16% validation.
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full,
    y_train_full,
    test_size=0.2,
    random_state=42
)
# Initialize and fit the Random Forest model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set (original df)
y_test_predictions = model.predict(X_test)

# Calculate performance metrics on the test set
mse_test = mean_squared_error(y_test, y_test_predictions)
mae_test = mean_absolute_error(y_test, y_test_predictions)
r2_test = r2_score(y_test, y_test_predictions)

# Now for the outliers removed DataFrame (df_outliers_removed)
# Features (X) and target (y) - Outliers removed
X_outliers = no_out_features
y_outliers = df_outliers_removed['Electrical_Power_Output']

# Train-test split for outliers removed
X_train_full_outliers, X_test_outliers, y_train_full_outliers, y_test_outliers␣
 ↪= train_test_split(
    X_outliers,
    y_outliers,
    test_size=0.2,
    random_state=42
)
X_train_outliers, X_val_outliers, y_train_outliers, y_val_outliers =␣
 ↪train_test_split(
    X_train_full_outliers,
    y_train_full_outliers,
```

```
    test_size=0.2,
    random_state=42
)
# Initialize and fit the Random Forest model on outliers removed
model_outliers = RandomForestRegressor(random_state=42)
model_outliers.fit(X_train_outliers, y_train_outliers)

# Make predictions on the test set (outliers removed)
y_test_predictions_outliers = model_outliers.predict(X_test_outliers)

# Calculate performance metrics on the test set for outliers removed
mse_test_outliers = mean_squared_error(y_test_outliers,␣
 ↪y_test_predictions_outliers)
mae_test_outliers = mean_absolute_error(y_test_outliers,␣
 ↪y_test_predictions_outliers)
r2_test_outliers = r2_score(y_test_outliers, y_test_predictions_outliers)

# Plot size
plt.figure(figsize=(16, 4))

# Selecting part of the data for plotting (original df)
predictions_num = y_test_predictions[start_obs:end_obs]
y_test_num = y_test.iloc[start_obs:end_obs]
x_values_num = np.arange(len(predictions_num))

# Selecting part of the data for plotting (outliers removed)
predictions_num_outliers = y_test_predictions_outliers[start_obs:end_obs]
y_test_num_outliers = y_test_outliers.iloc[start_obs:end_obs]

# Plot for original df
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.plot(x_values_num, y_test_num, linewidth=3, marker='.', label='Actual␣
 ↪Values')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='top')  #␣
 ↪underglow specs
plt.plot(x_values_num, predictions_num, linewidth=1.5, label='Predicted␣
 ↪Values', color='red')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='zero') ␣
 ↪# underglow specs
plt.title(f'RandomForestRegressor: Predicted vs Actual (Original Data)\n'
          f'MSE: {mse_test:.2f}, MAE: {mae_test:.2f}, R²: {r2_test:.2f} (Test)')
plt.xlabel('Observation Index')
plt.ylabel('Energy Values')
plt.legend(loc='upper center', framealpha=0.5)
plt.tight_layout()
```

```
# Plot for df_outliers_removed
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.plot(x_values_num, y_test_num_outliers, linewidth=3, marker='.',␣
 ↪label='Actual Values')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='top')  #␣
 ↪underglow specs
plt.plot(x_values_num, predictions_num_outliers, linewidth=1.5,␣
 ↪label='Predicted Values', color='#68ff00')
mplcyberpunk.add_gradient_fill(alpha_gradientglow=0.2, gradient_start='zero') ␣
 ↪# underglow specs
plt.title(f'RandomForestRegressor: Predicted vs Actual (Outliers Removed)\n'
          f'MSE: {mse_test_outliers:.2f}, MAE: {mae_test_outliers:.2f}, R²:␣
 ↪{r2_test_outliers:.2f} (Test)')
plt.xlabel('Observation Index')
plt.ylabel('Energy Values')
plt.legend(loc='upper center', framealpha=0.5)
plt.tight_layout()

# Show the plots
plt.show()
```



[34]:
```
# Model used
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Train-test split sizes
test_size_original = X_test.shape[0]
test_size_outliers_removed = X_test_outliers.shape[0]

# Calculate residuals
residuals = predictions_num - y_test_num

# Calculate residuals
residuals_outliers = predictions_num_outliers - y_test_num_outliers

# Plot size
```

```
plt.figure(figsize=(11, 4))

# Plot for original data
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.scatter(y_test_num, predictions_num, c=np.abs(residuals), cmap='coolwarm',␣
 ↪alpha=0.7)
plt.colorbar(label='Absolute Residuals')
plt.title(f'RandomForestRegressor: Predicted vs Actual (Original Data)\n'
          f'MSE: {mse_test:.2f}, MAE: {mae_test:.2f}, R²: {r2_test:.2f} (Test)')
plt.xlabel('Actual')
plt.ylabel('Predictions')
plt.plot([min(y_test_num), max(y_test_num)], [min(y_test_num),␣
 ↪max(y_test_num)], color='black', linestyle=':')

# Plot for outliers removed
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.scatter(y_test_num_outliers, predictions_num_outliers, c=np.
 ↪abs(residuals_outliers), cmap='coolwarm', alpha=0.7)
plt.colorbar(label='Absolute Residuals')
plt.title(f'RandomForestRegressor: Predicted vs Actual (Outliers Removed)\n'
          f'MSE: {mse_test_outliers:.4f}, MAE: {mae_test_outliers:.2f}, R²:␣
 ↪{r2_test_outliers:.2f} (Test)')
plt.xlabel('Actual')
plt.ylabel('Predictions')
plt.plot(
    [min(y_test_num_outliers),
     max(y_test_num_outliers)],
    [min(y_test_num_outliers),
     max(y_test_num_outliers)],
    color='black',
    linestyle=':'
)
# Show the side-by-side plots
plt.tight_layout()
plt.show()
```