Portfolio Assessment-1: “Hello Machine Learning for Engineering”

Name: Thenura Dulnath Kuruppuarachchi
ID: 103512993
Session: Studio 1-7(Thursday 6.30pm-8.30pm)

Data Set selection

The dataset selected: Combined power plant.

The reason for the choice:

Even thou I’m not an electrical major most of the mentioned projects were in pointed in related to the EE. So, I have chosen to do the first topic, exploring the combined cycle power plant dataset to understand energy production and efficiency.

Summery of the EDA conducted in studio 1:

In the first EDA, we have used various statical methods were used to understand the data distribution and identify patterns. In the given data set we can see a total of five columns. Let’s see what they are and what they are going to use to.

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

The dataset consists of five columns: Ambient Temperature (AT), Exhaust Vacuum (V), Ambient Pressure (AP), Relative Humidity (RH), and Power Output (PE). Correlation analysis showed significant relationships between temperature and power output, suggesting potential areas for efficiency optimization.

Class labeling for target variable/ developing ground truth data:

The target variable which is PE (hourly electrical energy output was taken in the Mega watts (MW). So, to develop the ground truth data these steps were taken.

- Data Verification: The given PE values were checked against the plant's operational records to ensure that they appropriately reflect the plant's output at full capacity. It guarantees that the PE values accurately reflect the genuine power output, hence providing a valid ground truth for model training.
- Consistency check: PE values were compared to ambient circumstances (AT, V, AP, and RH). This phase ensures that the data appropriately reflects the link between ambient conditions and power output.
- No class labelling: Since the PE is a continuous variable class labeling was not performed. However, the integrity of the PE values was strictly maintained to ensure that they can be used as a trustworthy goal for regression modelling.

Feature engineering and feature selection:

Feature Engineering

- Scaling**:** Standard scaling was applied to all features to ensure they are on a similar scale, which is important for certain machine learning models like linear regression. This scaling helps prevent features with larger ranges from disproportionately influencing the model's predictions.
- Polynomial Features**:** Polynomial features were generated to capture non-linear relationships between the features and the target variable (PE). New features such as AT^2, V^2 and interaction terms like AT*V to improve the model's accuracy.

Feature Selection

- Correlation Analysis: We found strong negative correlations between AT and PE (-0.948) and V and PE (-0.870), which means these features are important predictors for the model.
- SelectKBest: We used the SelectKBest method to keep the top features that have the strongest connection to PE. This helps reduce the number of features to the most important ones, making the model more efficient and less likely to overfit.

Model Training and Development

Linear Regression on Normal Dataset:

- Equation: $y=-14.7991*AT+-2.9493\cdot V+0.3694*AP+-2.3084*RH+454.3729$
- R^2 score: 0.9301
- MSE: 20.0799
- MAE: 3.0563

Linear Regression on Feature Engineered Dataset:

- Equation: $y=0.0*AT+-13.4240*V+-3.8072*AP+0.7609*RH+453.1795$
- R^2 score: 0. 9383
- MSE: 17.9031
- MAE: 3.3513

Decision Tree Regressor on Normal Dataset

- Feature Importances:
  - AT: 0.9058
  - V: 0.0567
- R^2 score: 0. 9295
- MSE: 20.4490
- MAE: 3.0760

Comparison Table

Model	R^2	MSE	MAE
Linear Reg (given)	0.93	20.27	3.59
Decision Tree (given)	0.92	20.44	3.07
Linear Regression (engineered)	0.93	17.90	3.35
Decision Tree (engineered)	0.92	21.66	3.20

Comparison table Summery & Conclusion

- **Linear Regression:** This model worked well on both the regular and feature-engineered datasets. The accuracy and error metrics improved slightly after adding polynomial features.
- **Decision Tree:** This model performed well, especially in understanding the relationships between features. However, it showed signs of overfitting, particularly after adding polynomial features.
- **Conclusion:** Feature engineering made the linear regression model a bit better, but it caused the decision tree model to perform worse because it overfitted the complex dataset.

References

- Tfekci, P & Kaya, H 2014, ‘UCI Machine Learning Repository’, *archive.ics.uci.edu*, viewed <https://archive.ics.uci.edu/dataset/294/combined+cycle+power+plant>

In [8]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler

DATASET_FILE = f'C:/assignments/Ai for eng/Portfolio 1/dataSet/Folds5x2_pp.xlsx'
```

In [12]:
```
df = pd.read_excel(DATASET_FILE)
print(df)
```

```
         AT      V       AP     RH      PE
0     14.96  41.76  1024.07  73.17  463.26
1     25.18  62.96  1020.04  59.08  444.37
2      5.11  39.40  1012.16  92.14  488.56
3     20.86  57.32  1010.24  76.64  446.48
4     10.82  37.50  1009.23  96.62  473.90
...     ...    ...      ...    ...     ...
9563  16.65  49.69  1014.01  91.00  460.03
9564  13.19  39.18  1023.67  66.78  469.62
9565  31.32  74.33  1012.92  36.48  429.57
9566  24.48  69.45  1013.86  62.39  435.74
9567  21.60  62.52  1017.23  67.87  453.28

[9568 rows x 5 columns]
```

In [13]:
```
print(df.head(5))
```

```
      AT      V       AP     RH      PE
0  14.96  41.76  1024.07  73.17  463.26
1  25.18  62.96  1020.04  59.08  444.37
2   5.11  39.40  1012.16  92.14  488.56
3  20.86  57.32  1010.24  76.64  446.48
4  10.82  37.50  1009.23  96.62  473.90
```

In [14]:
```
print(df.tail(5))
```

```
         AT      V       AP     RH      PE
9563  16.65  49.69  1014.01  91.00  460.03
9564  13.19  39.18  1023.67  66.78  469.62
9565  31.32  74.33  1012.92  36.48  429.57
9566  24.48  69.45  1013.86  62.39  435.74
9567  21.60  62.52  1017.23  67.87  453.28
```

In [44]:
```
#dataset exploration
print(df.describe())
```

```
                AT            V           AP           RH           PE
count  9568.000000  9568.000000  9568.000000  9568.000000  9568.000000
mean     19.651231    54.305804  1013.259078    73.308978   454.365009
std       7.452473    12.707893     5.938784    14.600269    17.066995
min       1.810000    25.360000   992.890000    25.560000   420.260000
25%      13.510000    41.740000  1009.100000    63.327500   439.750000
50%      20.345000    52.080000  1012.940000    74.975000   451.550000
75%      25.720000    66.540000  1017.260000    84.830000   468.430000
max      37.110000    81.560000  1033.300000   100.160000   495.760000
```

In [16]:
```
#checking whether it has null values or not
print(df.isnull().sum())
```

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

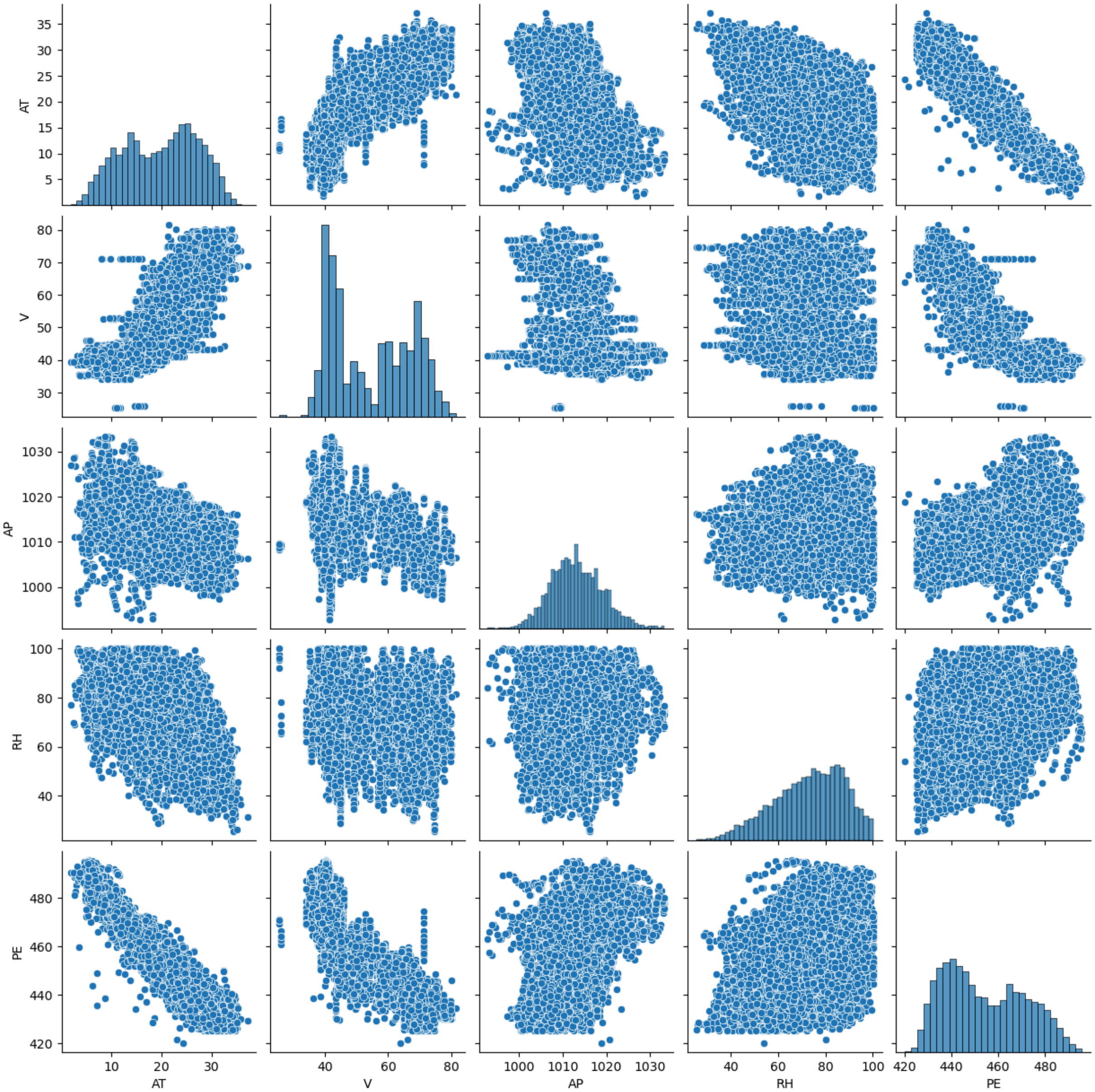
In [17]:
```
print(df.corr())
```

```
          AT         V        AP        RH        PE
AT  1.000000  0.844107 -0.507549 -0.542535 -0.948128
V   0.844107  1.000000 -0.413502 -0.312187 -0.869780
AP -0.507549 -0.413502  1.000000  0.099574  0.518429
RH -0.542535 -0.312187  0.099574  1.000000  0.389794
PE -0.948128 -0.869780  0.518429  0.389794  1.000000
```

In [75]:
```
sns.pairplot(df)
```

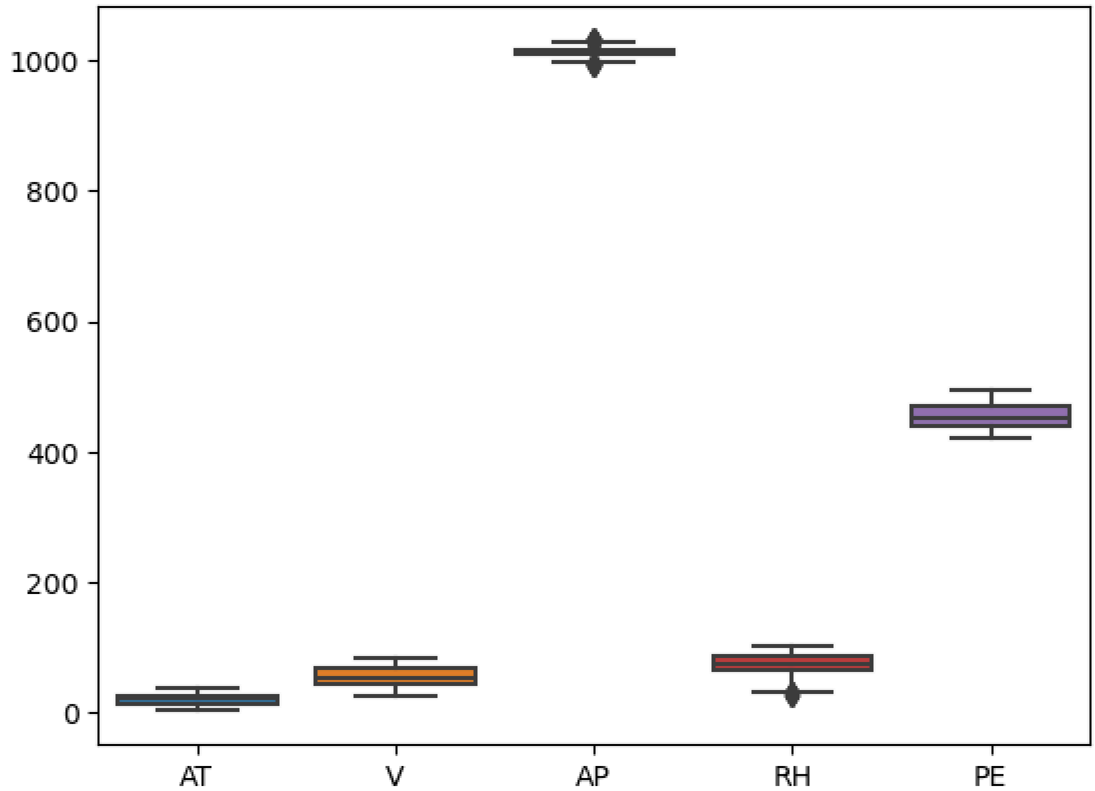
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[75]:
```
<seaborn.axisgrid.PairGrid at 0x26f5446d6d0>
```



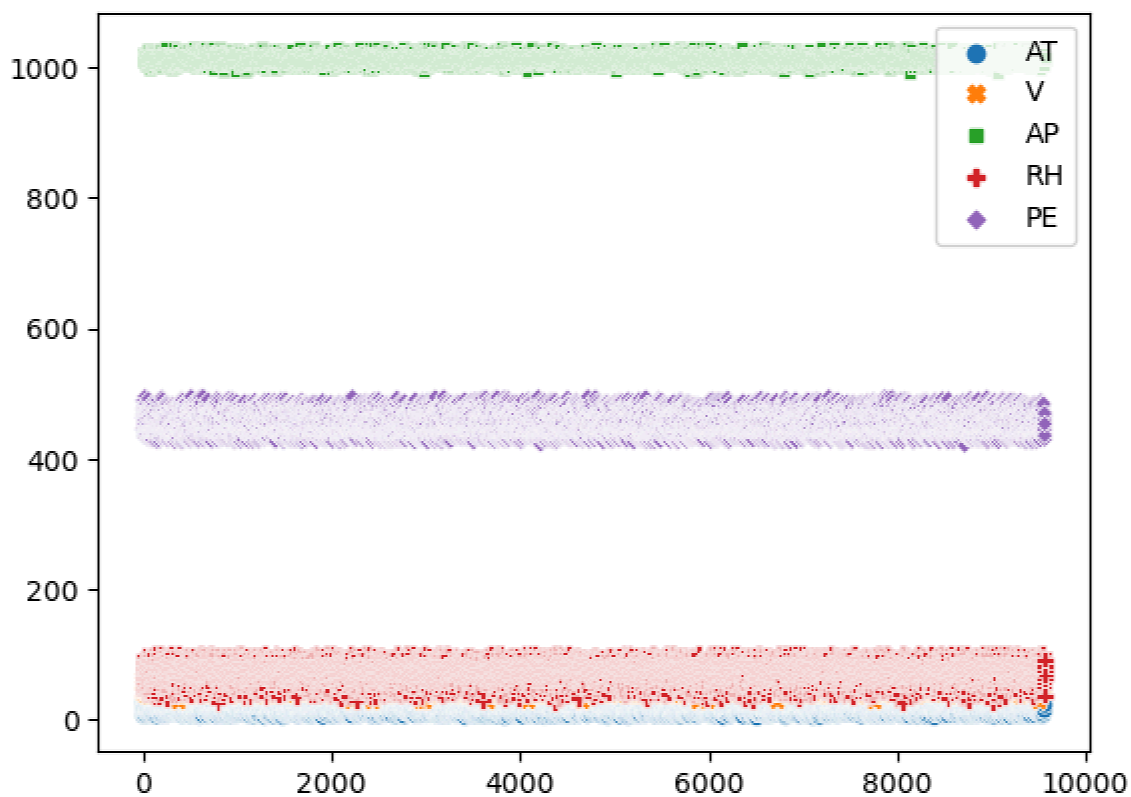
In [19]:
```
sns.boxplot(data=df)
```

Out[19]: <Axes: >



In [20]:
```
sns.scatterplot(data=df)
```

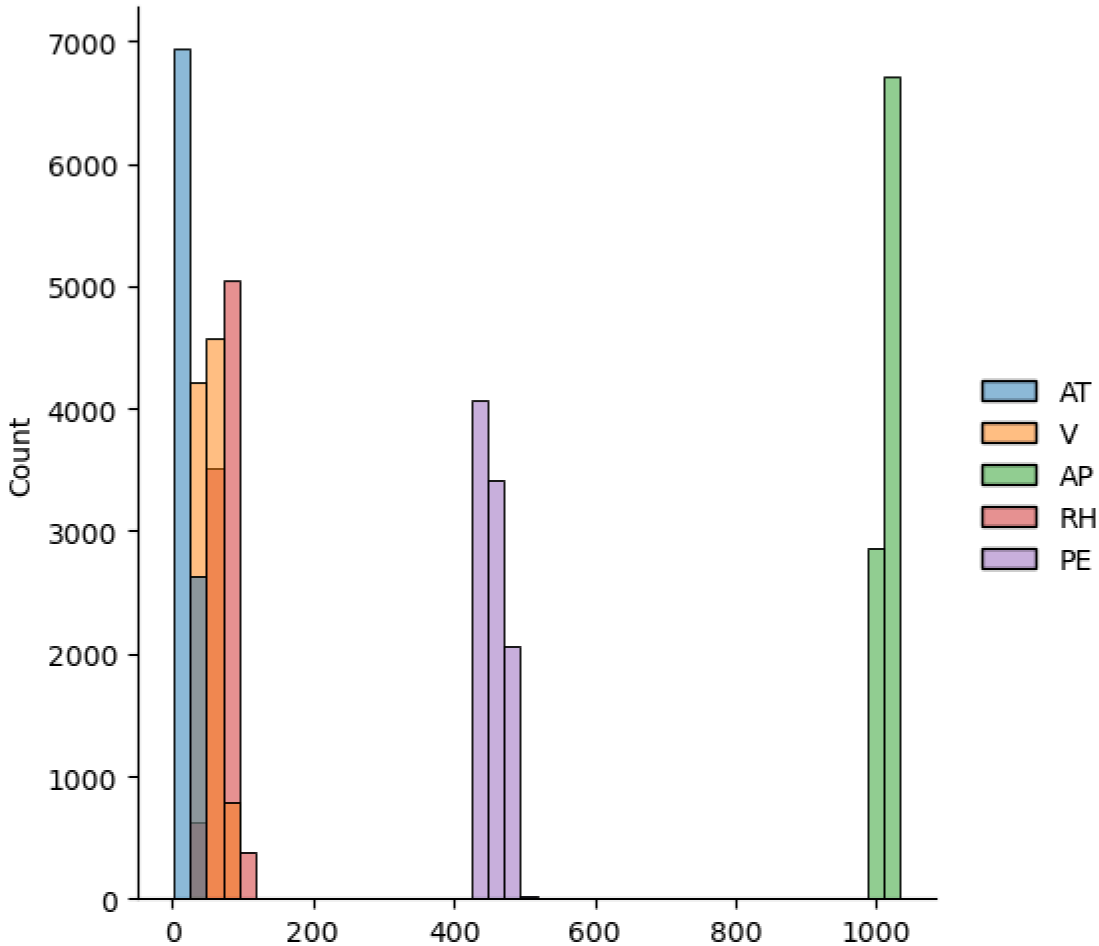
Out[20]: <Axes: >



In [65]:
```
sns.displot(data=df)
```

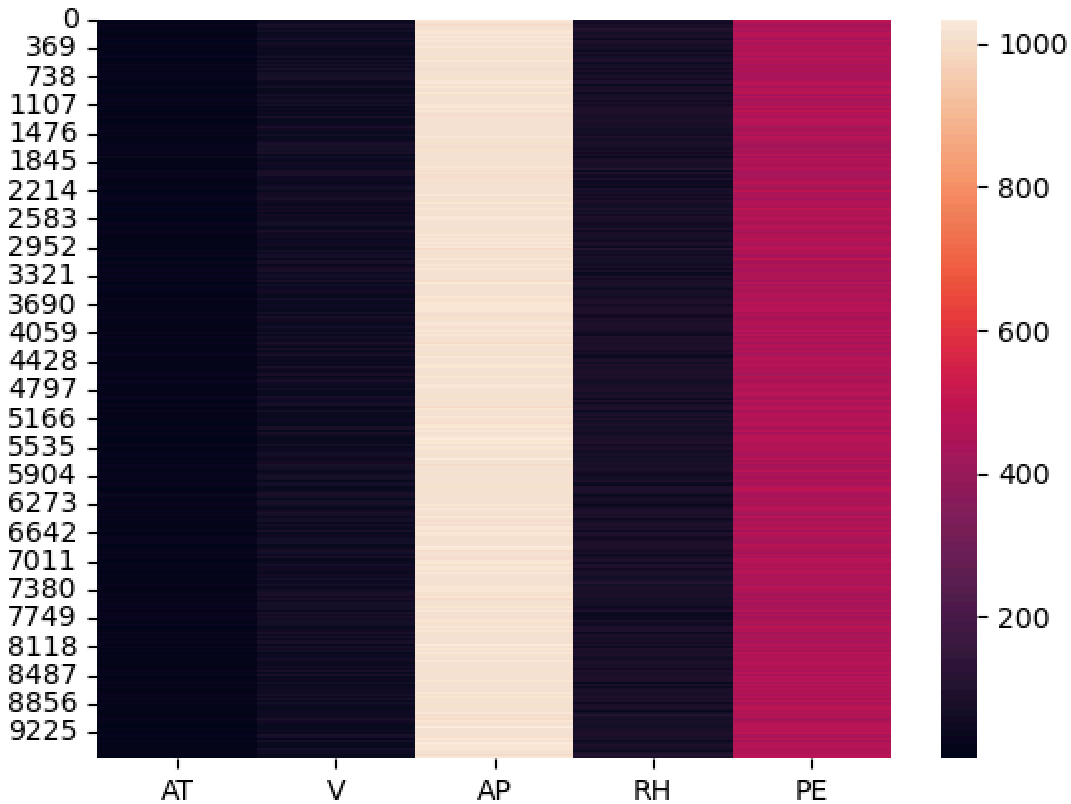
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[65]: <seaborn.axisgrid.FacetGrid at 0x26f4a0ad410>



In [22]:
```
sns.heatmap(data=df)
```

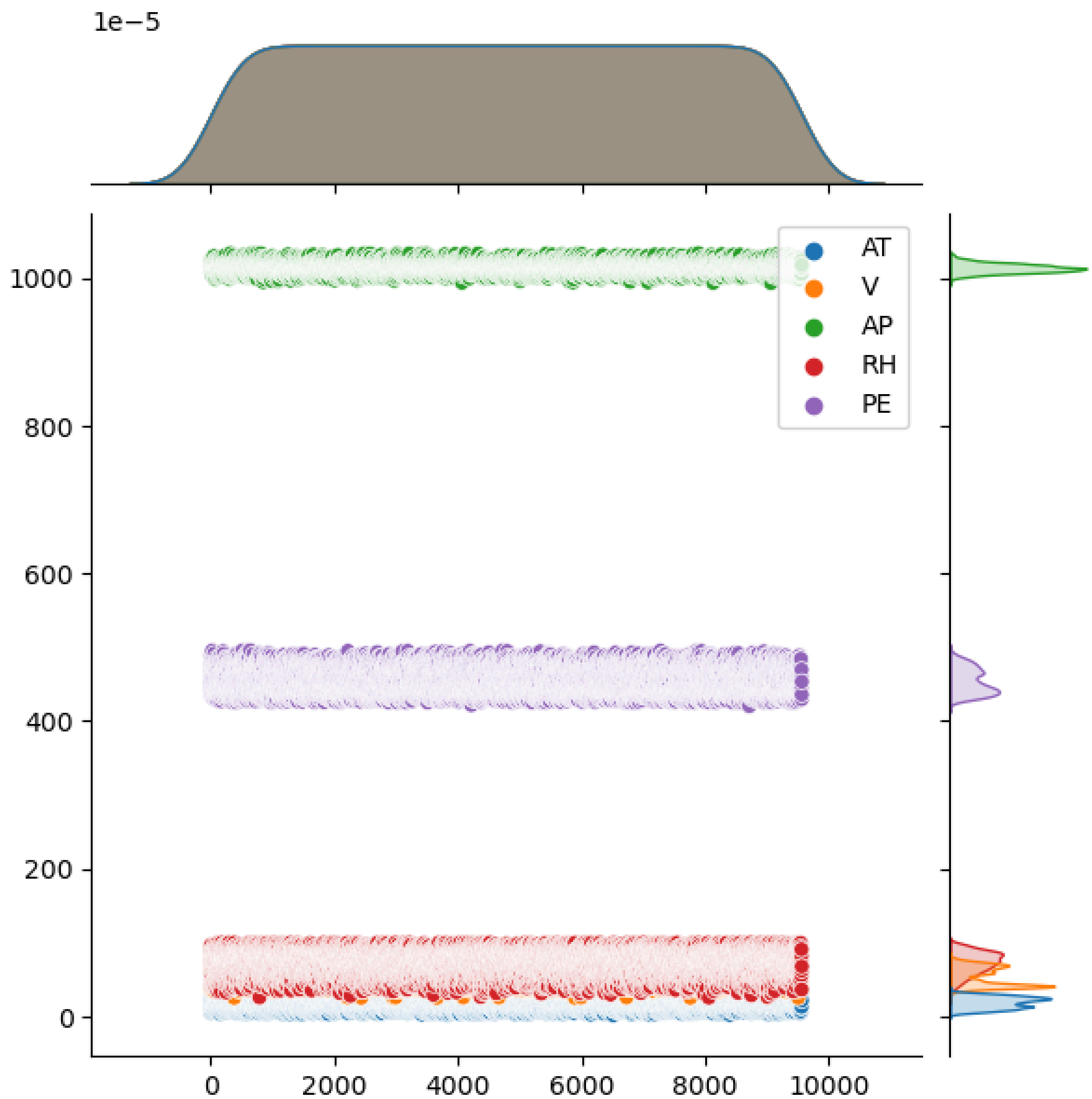
Out[22]: <Axes: >



In [71]:
```
sns.jointplot(data=df)
```

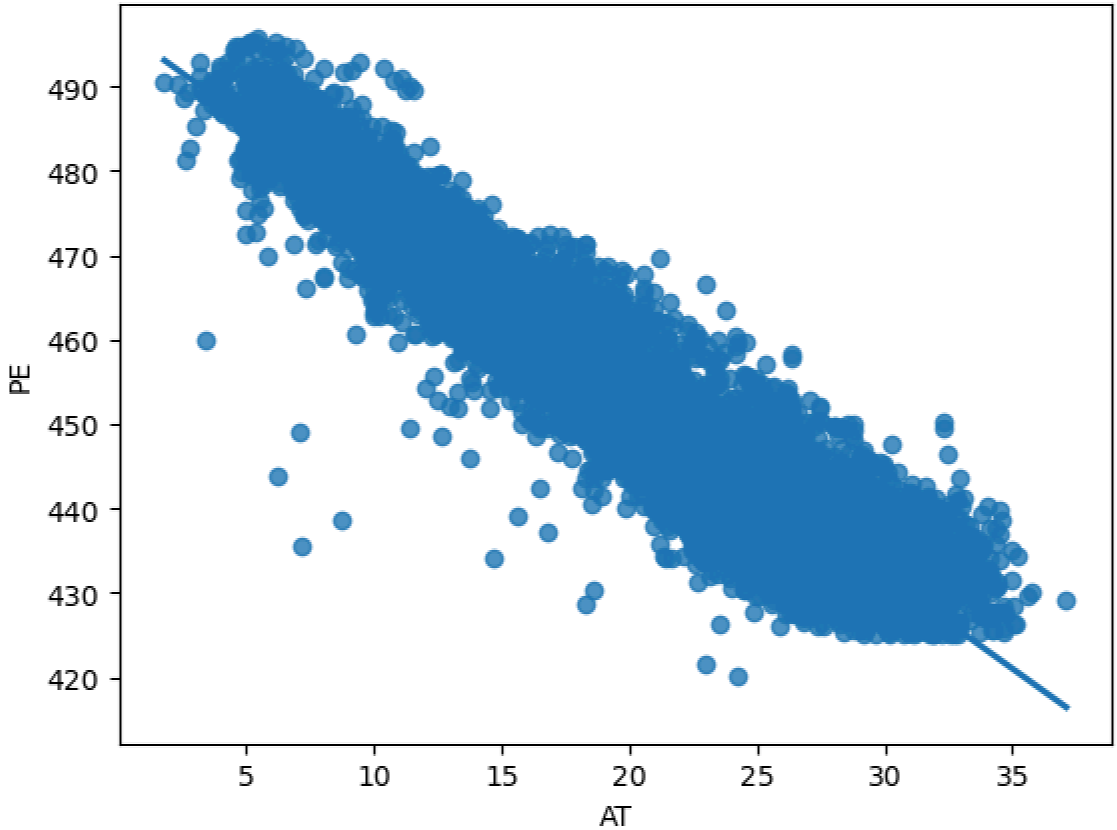
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[71]: <seaborn.axisgrid.JointGrid at 0x26f4eed0ed0>



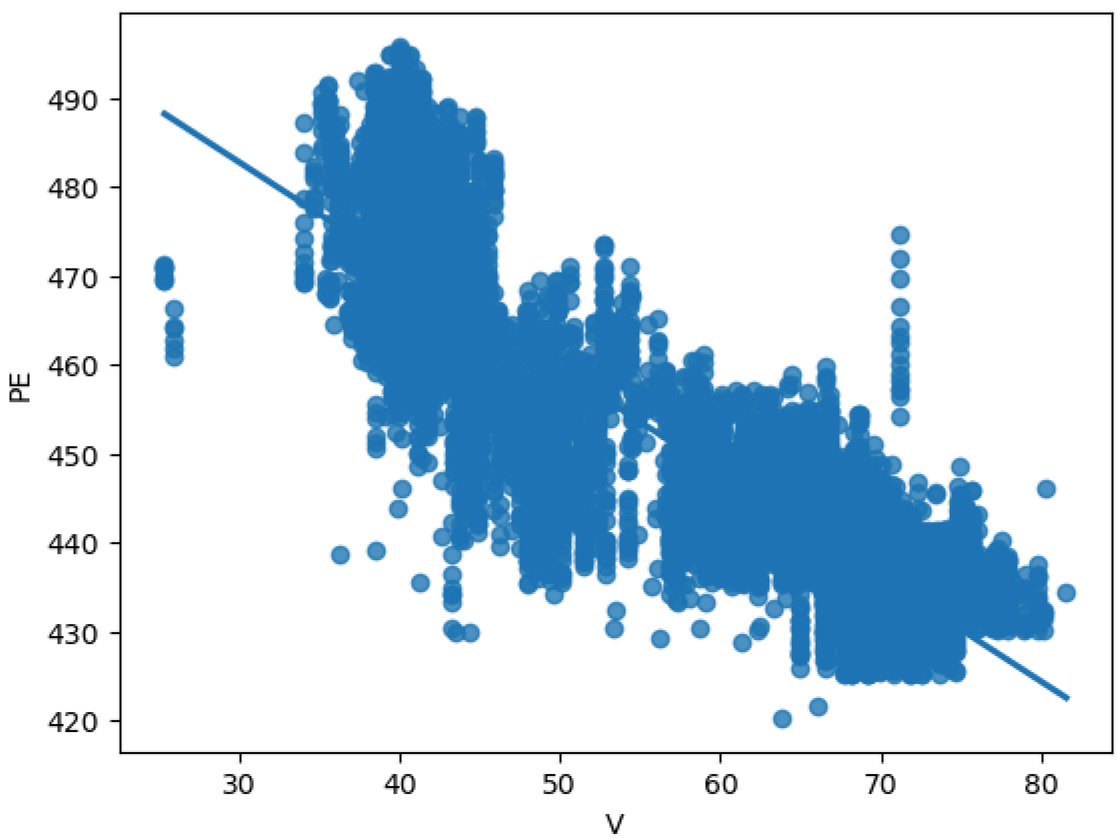
In [73]:
```
sns.regplot(x='AT', y='PE', data=df)
```

Out[73]: <Axes: xlabel='AT', ylabel='PE'>



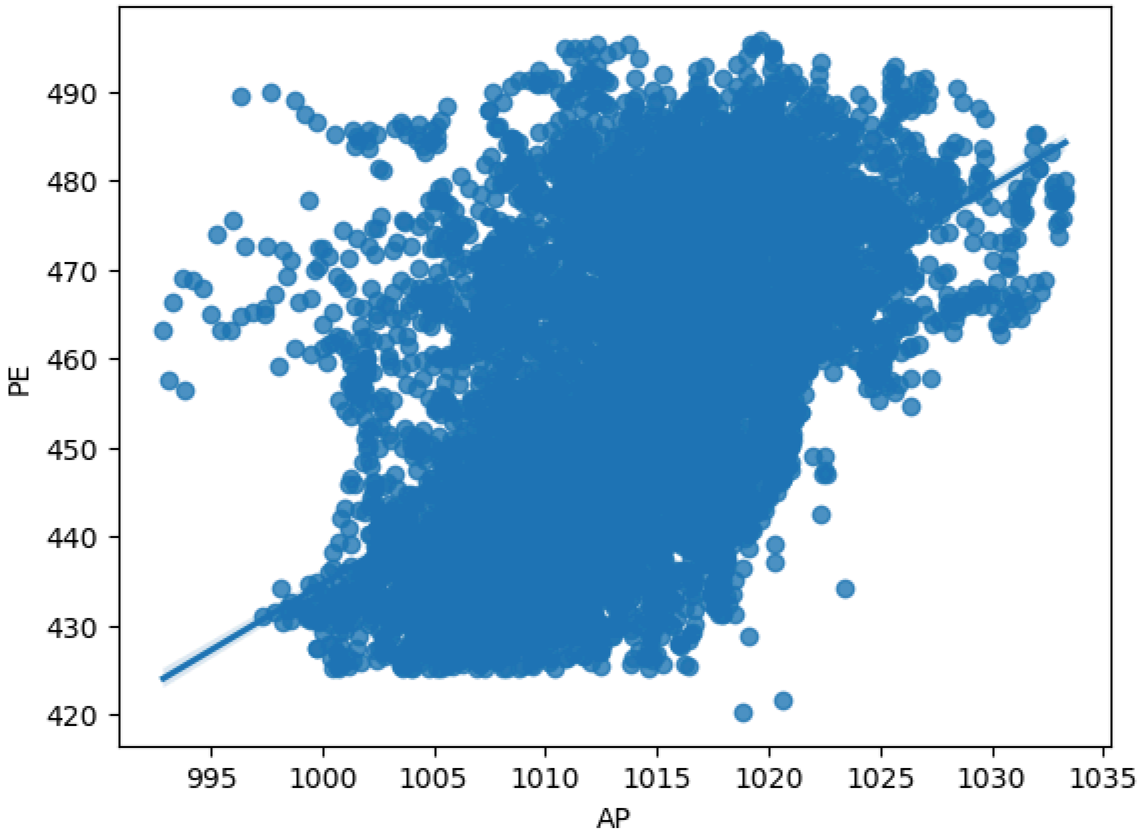
In [67]:
```
sns.regplot(x='AT', y='PE', data=df)
```

Out[67]: `<Axes: xlabel='V', ylabel='PE'>`



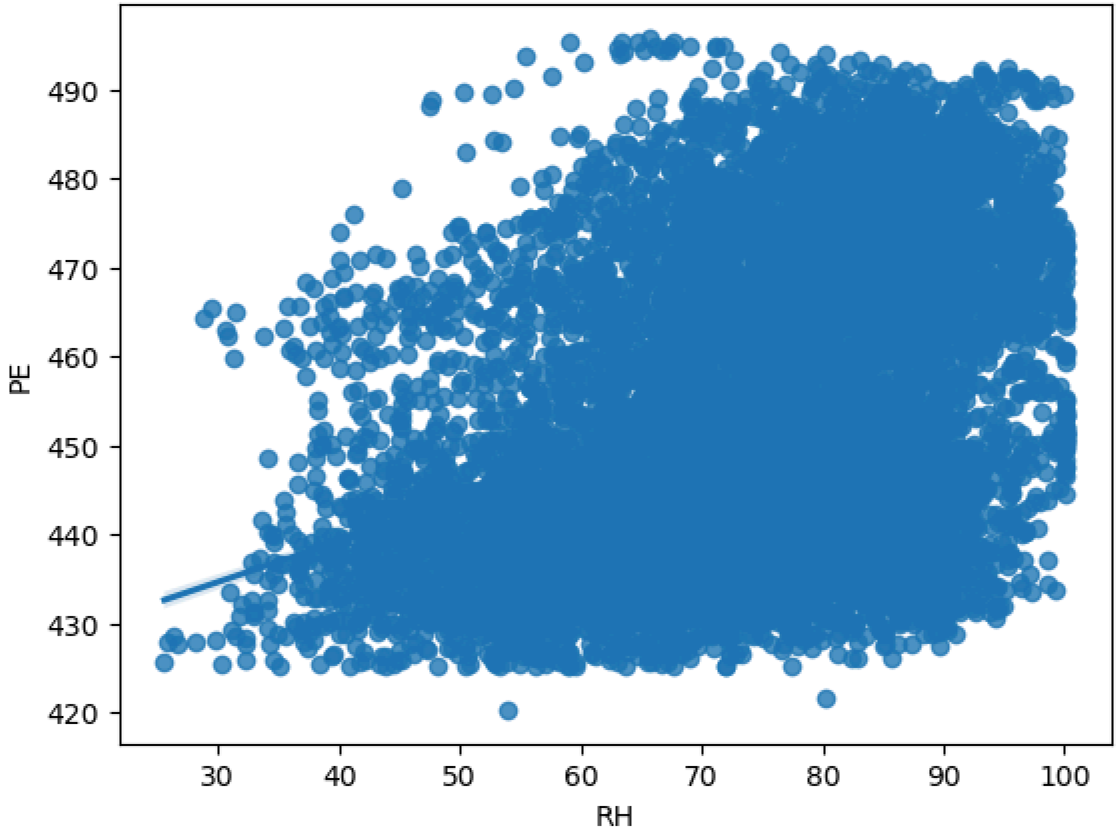
In [68]:
```
sns.regplot(x='AP', y='PE', data=df)
```

Out[68]: <Axes: xlabel='AP', ylabel='PE'>



In [27]:
```
sns.regplot(x='RH', y='PE', data=df)
```

Out[27]: <Axes: xlabel='RH', ylabel='PE'>



In [48]:
```python
scaler = StandardScaler()

# Feature
X = df[['AT', 'V', 'AP', 'RH']]
X_normalized = scaler.fit_transform(X)

# Target
y = df['PE']

# spliting the data set
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2,

print(f'X: {X.size}')
print(f'y: {y.size}')
print("Original :", X.shape)
print("Normalized :", X_normalized.shape)
```

```
X: 38272
y: 9568
Original : (9568, 4)
Normalized : (9568, 4)
```

In [46]:
```python
reg_model = LinearRegression()
print(reg_model)
```

```
LinearRegression()
```

In [30]:
```python
reg_model.fit(X_train, y_train)
```

Out[30]:
```
▾ LinearRegression
LinearRegression()
```

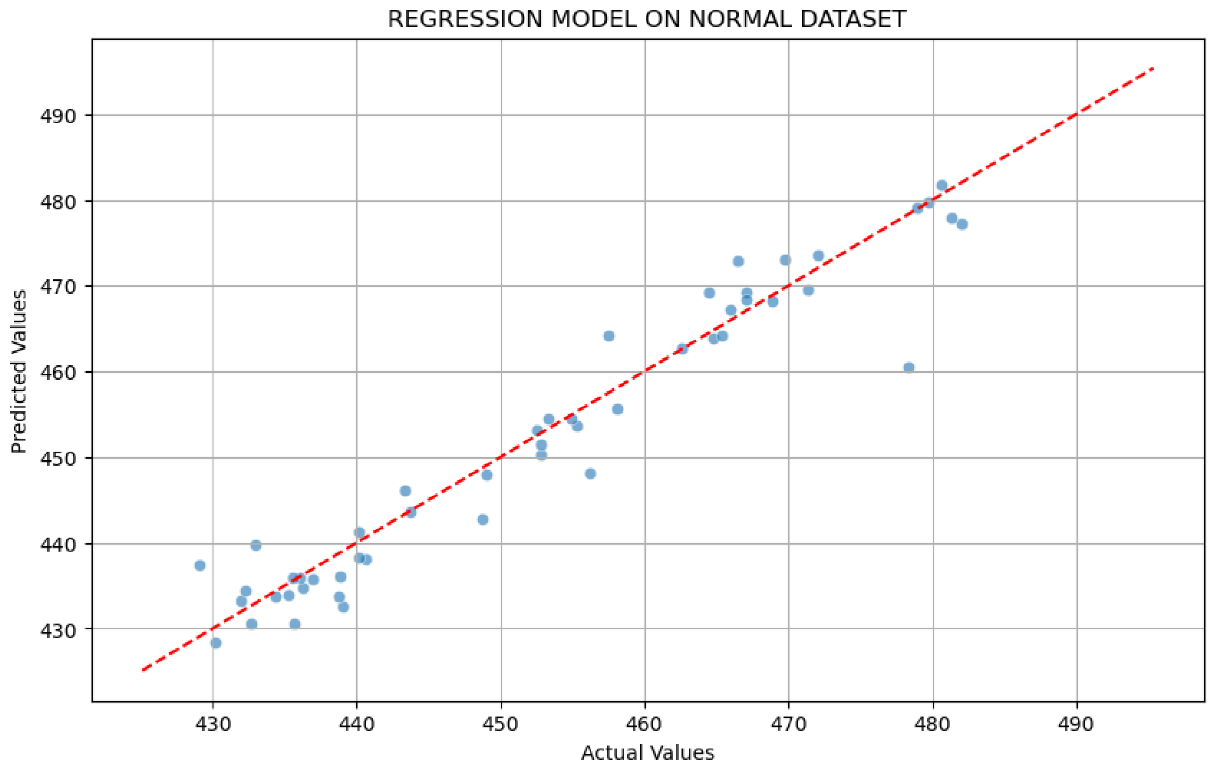
In [31]:
```
train_score = reg_model.score(X_train, y_train)
```

In [32]:
```
y_pred = reg_model.predict(X_test)
```

In [33]:
```
print(f'Training Score: {train_score}')
print(f'Predictions: {y_pred}')
print(f'Coefficients: {reg_model.coef_}')
print(f'Intercept: {reg_model.intercept_}')
print(f'Equation: y = {reg_model.coef_[0]} * AT + {reg_model.coef_[1]} * V + {reg_m
print(f'Accuracy: {reg_model.score(X_test, y_test)}')

r2 = r2_score(y_test, y_pred)
print(f'R² Score: {r2:.4f}')

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse:.4f}')

mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error (MAE): {mae:.4f}')
```

```
Training Score: 0.928331545565795
Predictions: [455.68020791 438.73212215 434.16444    ... 482.16817365 435.41524413
 458.76150613]
Coefficients: [-14.79909089  -2.94926621   0.3693725   -2.30844154]
Intercept: 454.37288174293514
Equation: y = -14.799090887518005 * AT + -2.9492662148344597 * V + 0.369372503902468
4 * AP + -2.3084415408470145 * RH + 454.37288174293514
Accuracy: 0.9301046431962188
R² Score: 0.9301
Mean Squared Error (MSE): 20.2737
Mean Absolute Error (MAE): 3.5959
```

In [58]:
```
plot_actual_vs_predicted('REGRESSION MODEL ON NORMAL DATASET', y_test, y_pred, 50)
```



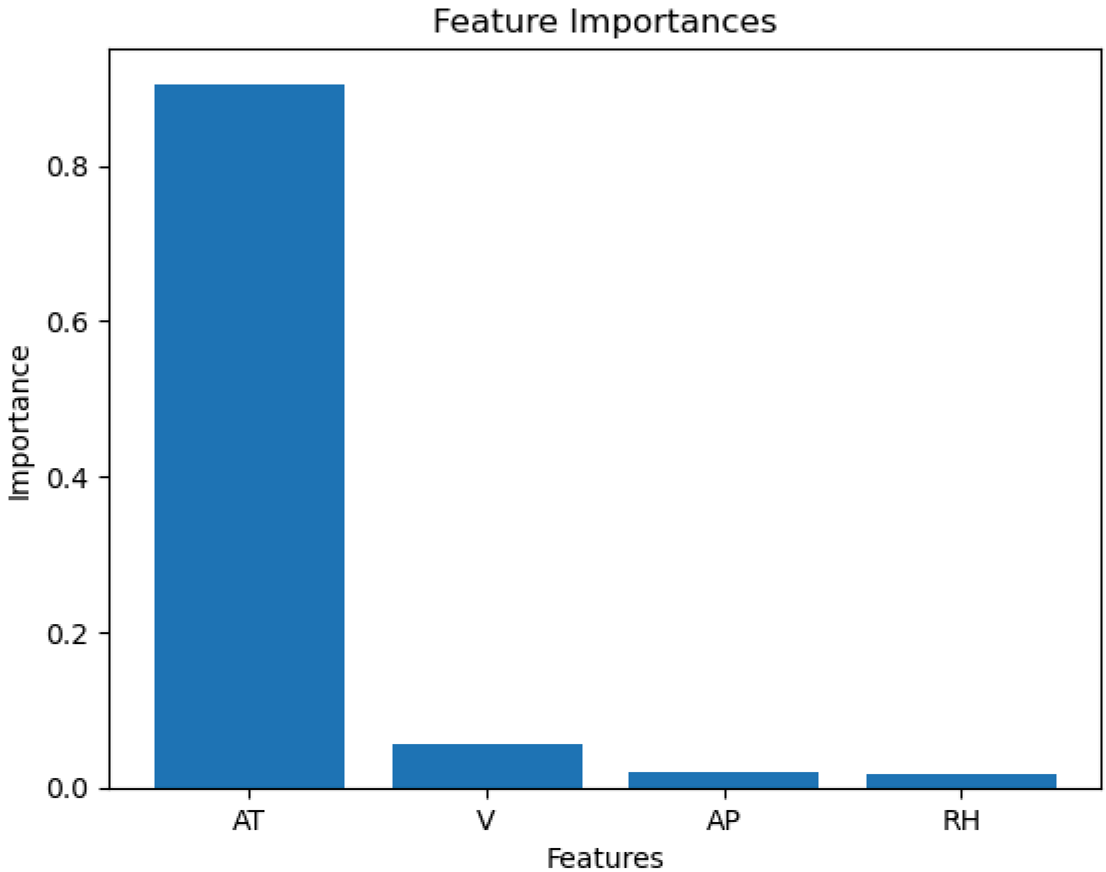
In [35]:
```
tree_model = DecisionTreeRegressor()
print(tree_model)
tree_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor()
```

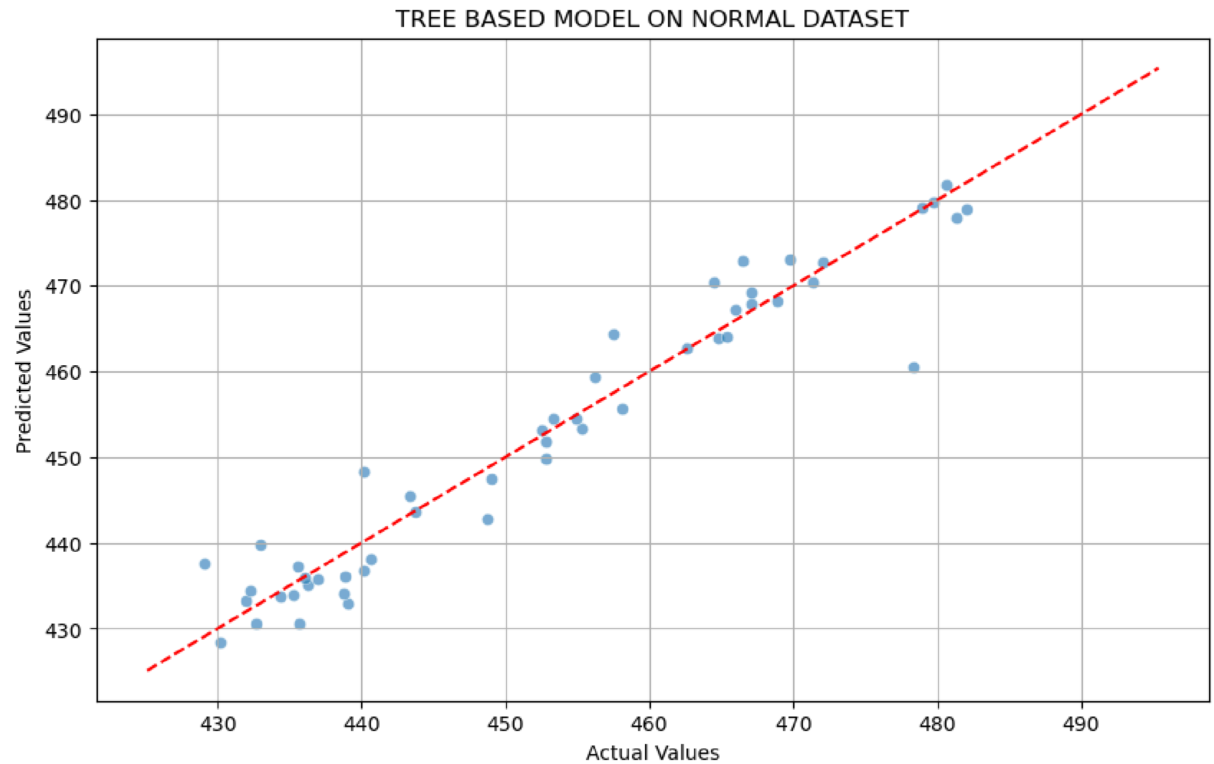
Out[35]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [36]:
```
train_score = tree_model.score(X_train, y_train)
y_pred = tree_model.predict(X_test)
print(f'Training Score: {train_score}')
print(f'Predictions: {y_pred}')
print(f'Feature Importances: {tree_model.feature_importances_}')

r2 = r2_score(y_test, y_pred)
print(f'R² Score: {r2:.4f}')

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse:.4f}')

mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error (MAE): {mae:.4f}')

# Plot feature importances
plt.bar(X.columns, tree_model.feature_importances_)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances')
plt.show()
```

```
Training Score: 1.0
Predictions: [456.57 436.96 431.73 ... 482.39 429.28 456.35]
Feature Importances: [0.90524682 0.05652531 0.02073968 0.0174882 ]
R² Score: 0.9308
Mean Squared Error (MSE): 20.0799
Mean Absolute Error (MAE): 3.0563
```



In [50]:
```
plot_actual_vs_predicted('TREE BASED MODEL ON NORMAL DATASET', y_test, y_pred, 50)
```



In [52]:
```
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures

k_best_features = 2
selector = SelectKBest(score_func=f_regression, k=k_best_features)
X_selected = selector.fit_transform(X_normalized, y)
n_components_pca = 2
pca = PCA(n_components=n_components_pca)
X_pca = pca.fit_transform(X_normalized)

degree_poly = 2
poly = PolynomialFeatures(degree=degree_poly)

X_poly_scaled = poly.fit_transform(X_normalized)

feature_names = poly.get_feature_names_out(X.columns)
print("Polynomial Feature Names:")
print(feature_names)

print("\nFirst 5 rows of the transformed dataset:")
print(X_poly_scaled[:5])

print("Original Shape:", X.shape)
print("Normalized Shape:", X_normalized.shape)
print("Selected Shape:", X_selected.shape)
print("PCA Shape:", X_pca.shape)
print("Poly Scaled Shape:", X_poly_scaled.shape)

X_train, X_test, y_train, y_test = train_test_split(X_poly_scaled, y, test_size=0.2

print(f'X_train shape: {X_train.shape}')
```

```python
print(f'X_test shape: {X_test.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
Polynomial Feature Names:
['1' 'AT' 'V' 'AP' 'RH' 'AT^2' 'AT V' 'AT AP' 'AT RH' 'V^2' 'V AP' 'V RH'
 'AP^2' 'AP RH' 'RH^2']

First 5 rows of the transformed dataset:
[[ 1.00000000e+00 -6.29519384e-01 -9.87296587e-01  1.82048840e+00
  -9.51935258e-03  3.96294655e-01  6.21522339e-01 -1.14603273e+00
   5.99261697e-03  9.74754551e-01 -1.79736198e+00  9.39842431e-03
   3.31417800e+00 -1.73298709e-02  9.06180735e-05]
 [ 1.00000000e+00  7.41909107e-01  6.81045124e-01  1.14186280e+00
  -9.74620516e-01  5.50429122e-01  5.05273579e-01  8.47158411e-01
  -7.23079836e-01  4.63822461e-01  7.77660094e-01 -6.63760550e-01
   1.30385066e+00 -1.11288291e+00  9.49885151e-01]
 [ 1.00000000e+00 -1.95129733e+00 -1.17301765e+00 -1.85077563e-01
   1.28983970e+00  3.80756128e+00  2.28890620e+00  3.61141356e-01
  -2.51686077e+00  1.37597040e+00  2.17099247e-01 -1.51300473e+00
   3.42537044e-02 -2.38720389e-01  1.66368646e+00]
 [ 1.00000000e+00  1.62205478e-01  2.37203273e-01 -5.08392982e-01
   2.28159926e-01  2.63106172e-02  3.84756703e-02 -8.24641269e-02
   3.70087900e-02  5.62653925e-02 -1.20592479e-01  5.41202811e-02
   2.58463424e-01 -1.15994905e-01  5.20569519e-02]
 [ 1.00000000e+00 -1.18506869e+00 -1.32253884e+00 -6.78470364e-01
   1.59669940e+00  1.40438781e+00  1.56729937e+00  8.04033989e-01
  -1.89219848e+00  1.74910897e+00  8.97303406e-01 -2.11169697e+00
   4.60322035e-01 -1.08331323e+00  2.54944899e+00]]
Original Shape: (9568, 4)
Normalized Shape: (9568, 4)
Selected Shape: (9568, 2)
PCA Shape: (9568, 2)
Poly Scaled Shape: (9568, 15)
X_train shape: (7654, 15)
X_test shape: (1914, 15)
y_train shape: (7654,)
y_test shape: (1914,)
```

In [39]:

```python
reg_model.fit(X_train, y_train)
train_score = reg_model.score(X_train, y_train)
print(f'Training Score: {train_score}')
print(f'Predictions: {y_pred}')
print(f'Coefficients: {reg_model.coef_}')
print(f'Intercept: {reg_model.intercept_}')
print(f'Equation: y = {reg_model.coef_[0]} * AT + {reg_model.coef_[1]} * V + {reg_m
print(f'Accuracy: {reg_model.score(X_test, y_test)}')

r2 = r2_score(y_test, y_pred)
print(f'R² Score: {r2:.4f}')

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse:.4f}')

mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error (MAE): {mae:.4f}')
```

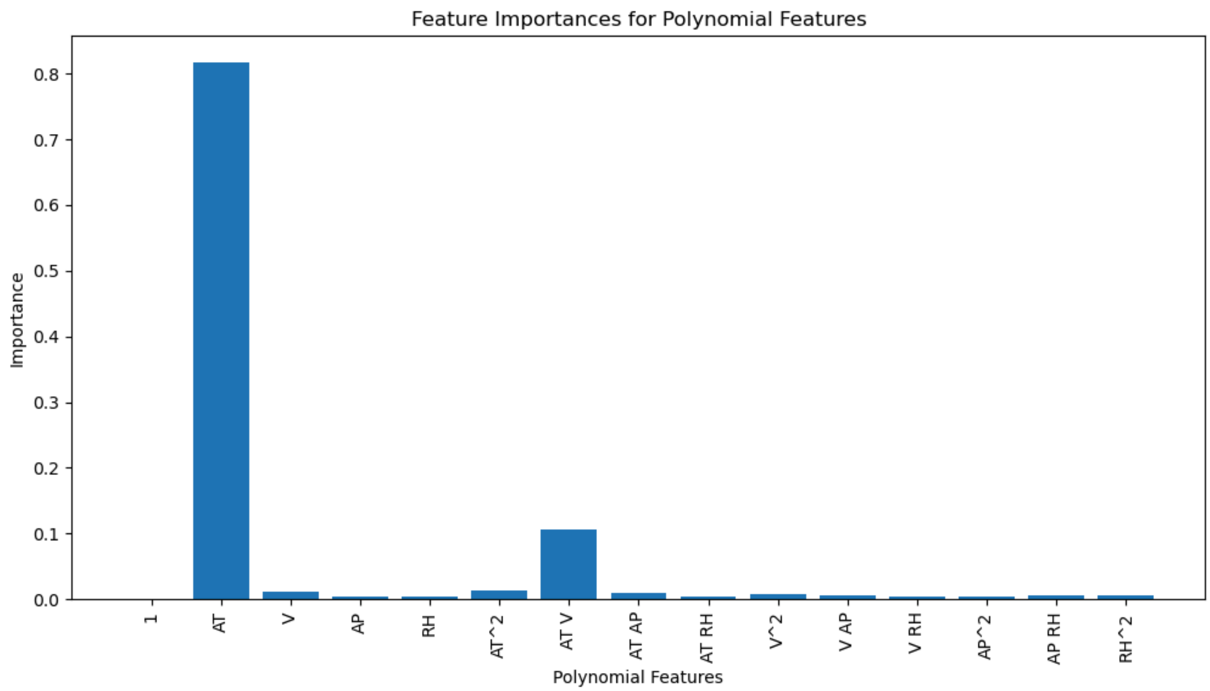
```
Training Score: 0.9377092268931407
Predictions: [456.57 436.96 431.73 ... 482.39 429.28 456.35]
Coefficients: [ 0.00000000e+00 -1.34240367e+01 -3.80720801e+00  7.60908552e-01
 -1.78462369e+00  1.00085045e+00  9.80509667e-01  1.39188923e-01
 -6.01410937e-01 -9.84665994e-02  1.71880579e-01  1.24857352e-02
 -2.64540651e-01 -3.15554253e-01 -4.09857299e-01]
Intercept: 453.1795000724065
Equation: y = 0.0 * AT + -13.424036734708139 * V + -3.8072080130992547 * AP + 0.7609
085522917225 * RH + 453.1795000724065
Accuracy: 0.9382776706597626
R² Score: 0.9308
Mean Squared Error (MSE): 20.0799
Mean Absolute Error (MAE): 3.0563
```

In [51]:
```python
plot_actual_vs_predicted('REGRESSION MODEL ON FEATURE ENGINEERED DATASET', y_test,
```

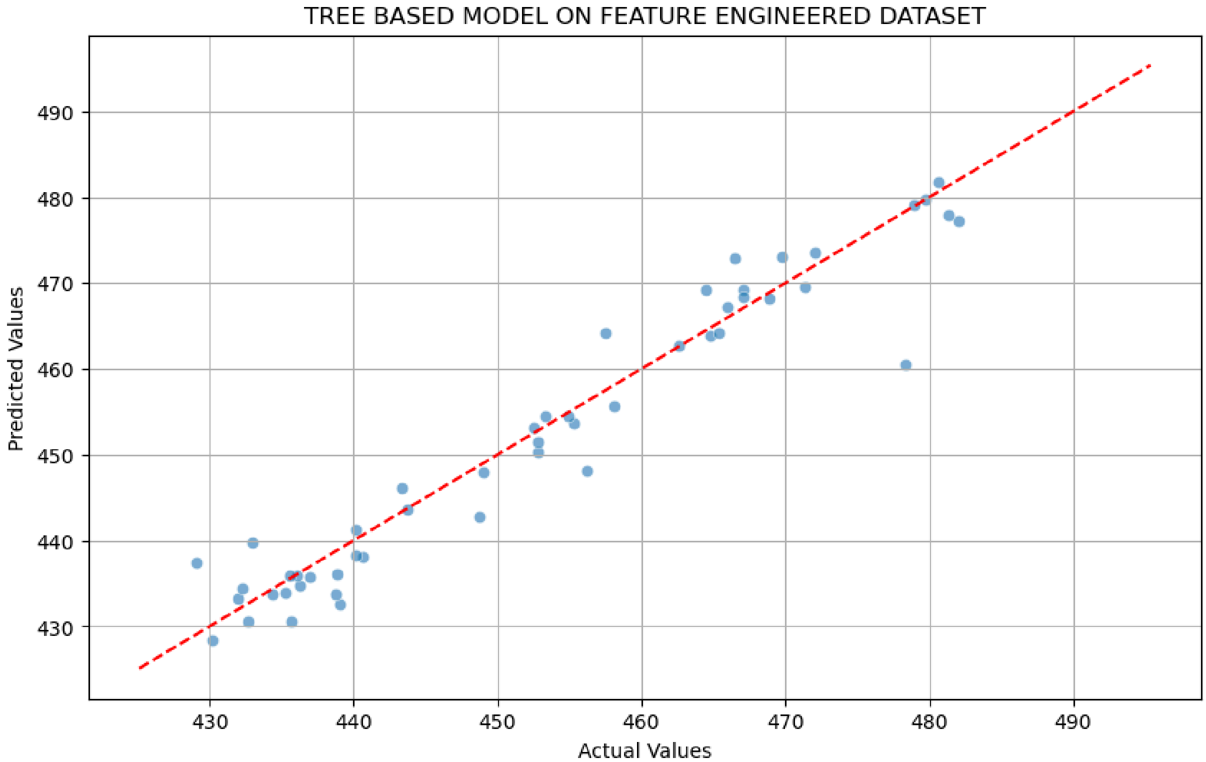
In [55]:
```python
tree_model.fit(X_train, y_train)
train_score = tree_model.score(X_train, y_train)
y_pred = tree_model.predict(X_test)
print(f'Training Score: {train_score}')
print(f'Predictions: {y_pred}')
print(f'Feature Importances: {tree_model.feature_importances_}')

r2 = r2_score(y_test, y_pred)
print(f'R² Score: {r2:.4f}')

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse:.4f}')

mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error (MAE): {mae:.4f}')
```

```
feature_names = poly.get_feature_names_out(X.columns)
assert len(feature_names) == len(tree_model.feature_importances_)

plt.figure(figsize=(12, 6))
plt.bar(feature_names, tree_model.feature_importances_)
plt.xlabel('Polynomial Features')
plt.ylabel('Importance')
plt.title('Feature Importances for Polynomial Features')
plt.xticks(rotation=90)
plt.show()
```

```
Training Score: 1.0
Predictions: [453.62 434.77 438.04 ... 485.31 439.03 456.59]
Feature Importances: [0.         0.81631505 0.01140105 0.00427166 0.00345139 0.01384
155
 0.10530994 0.00911107 0.0044146  0.00719858 0.00563522 0.00476022
 0.00369013 0.0052123  0.00538725]
R² Score: 0.9242
Mean Squared Error (MSE): 21.9877
Mean Absolute Error (MAE): 3.2316
```



In [57]:
```
plot_actual_vs_predicted('TREE BASED MODEL ON FEATURE ENGINEERED DATASET', y_test,
```


TREE BASED MODEL ON FEATURE ENGINEERED DATASET
Predicted Values
490
480
470
460
450
440
430
430
440
450
460
470
480
490
Actual Values