Data analysis

Preliminary analysis: descriptive statistics

Import the datafile CCPP_data.trt. Get familiar with the data and answer the questions:

1.How many observations are there? How many variables?

```
import pandas as pd
```

```
df = pd.read_csv('CCPP_data.txt', delimiter='\t')
```

```
# Determine the number of observations and variables
num_observations, num_variables = df.shape
print(f'Number of observations: {num_observations}')
print(f'Number of variables: {num_variables}')
```

```
Number of observations: 9568
Number of variables: 5
```

Number of Observations: 9568

Number of Variables: 5

2.Are there any missing values in the dataset? If you think it is appropriate, delete the variables concerning missing values.

```
# Check for missing values in the dataset
missing_values = df.isnull().sum()
print(missing_values)
```

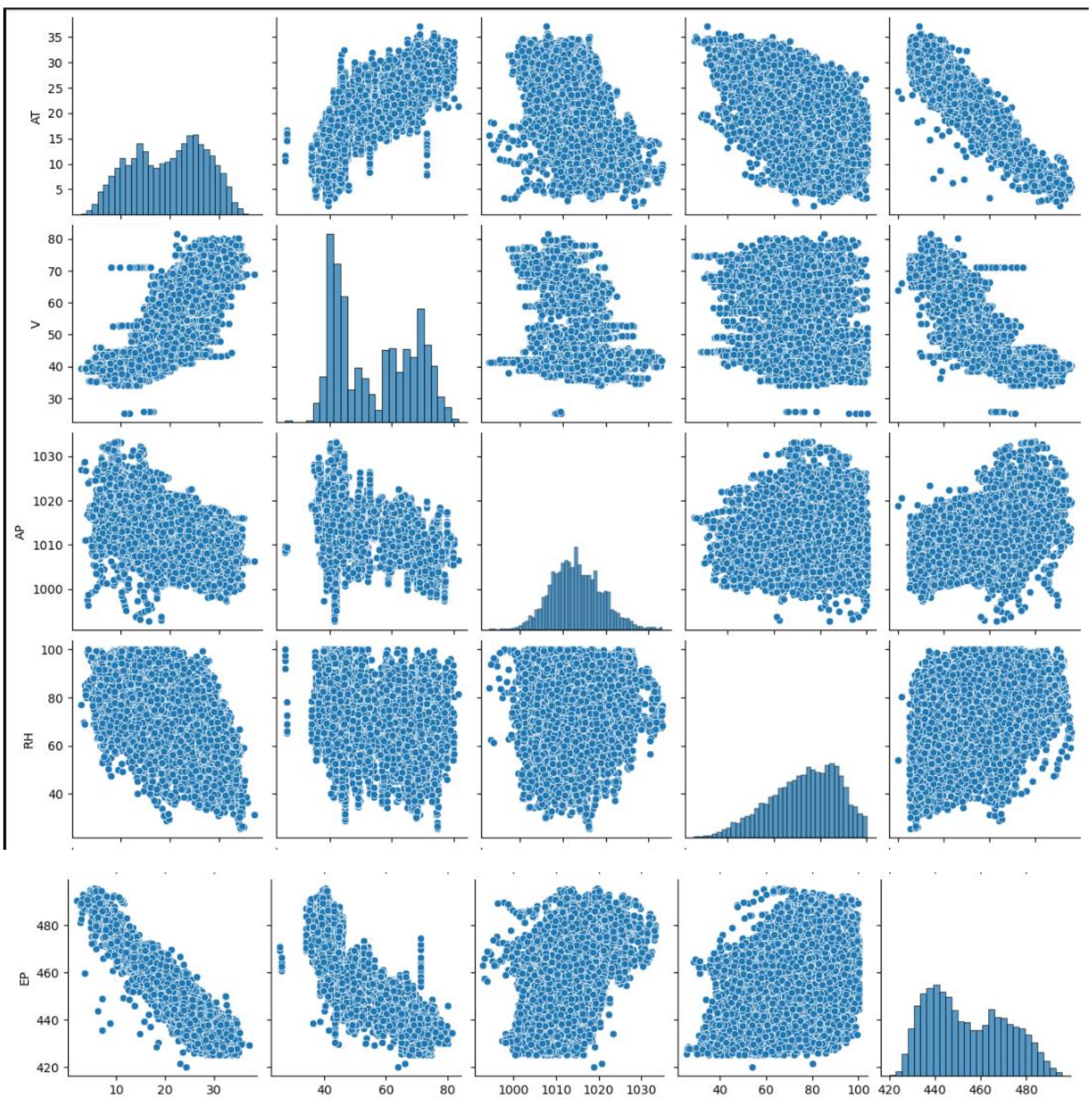
```
AT    0
V     0
AP    0
RH    0
EP    0
dtype: int64
```

There is no missing value so we don't need to delete the any values.

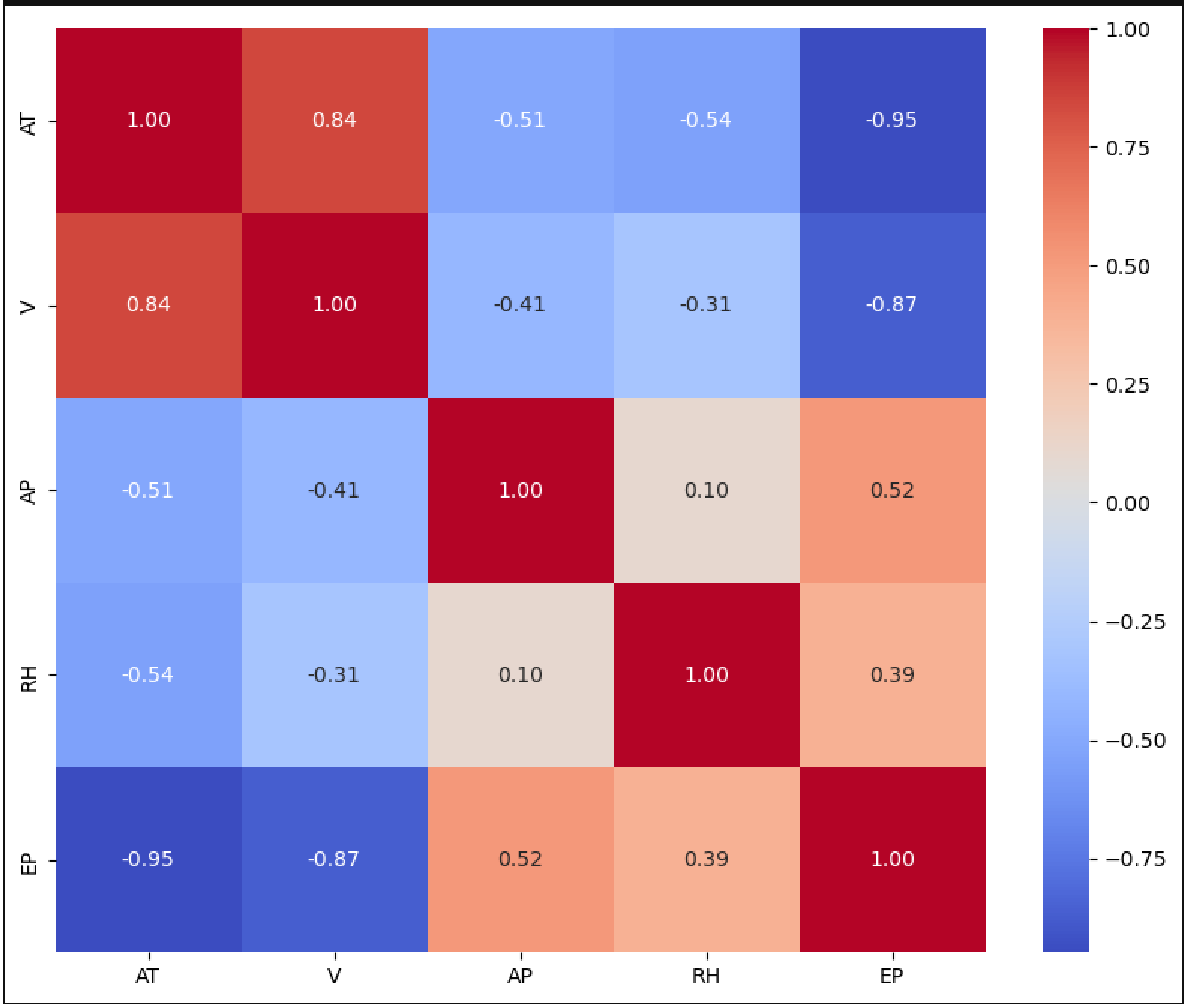
3.Calculate descriptive statistics for all the variables. You can use graphics of your choice to help you describe the data (boxplot, scatter plot, etc.). Interpret the results.

```python
# Part 3: Calculate descriptive statistics.
descriptive_stats = df.describe()
print(f"Descriptive Statistics:\n{descriptive_stats}")
```

```
Descriptive Statistics:
                AT            V           AP           RH           EP
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
```



```python
# Generate a heatmap to visualize the correlation between variables
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```



- The correlation heatmap reveals strong inverse relationships between Ambient Temperature (AT) and Exhaust Vacuum (V) with Electrical Energy Output (EP), suggesting that as AT and V increase, EP decreases, indicating more efficient power plant operation under these conditions. Atmospheric Pressure (AP) and Relative Humidity (RH) show weaker relationships with EP.
- The pairplot illustrates these relationships with scatterplots showing negative slopes for AT vs. EP and V vs. EP, reinforcing the correlation findings. The distributions on diagonal suggest that AT and V are roughly normally distributed, whereas EP, AP, and RH show some skewness. These visuals collectively suggest that AT and V are significant predictors for EP in the power plant dataset.

Principal Component Analysis (PCA)

Theoretical question

1. If two variables are perfectly correlated in the dataset, would it be suitable to include both in the analysis when performing PCA? Justify your answer. In contrast, what if the variables are completely uncorrelated?

- **Perfectly Correlated Variables:** Avoid including both in PCA due to redundancy and potential numerical instability.
- **Completely Uncorrelated Variables:** Include them in PCA as they provide unique and complementary information for dimensionality reduction.

PCA is most effective when dealing with uncorrelated or weakly correlated variables and as it aims to transform original variables into a set of orthogonal components capturing maximum variance. It is essential to consider correlation structure of variables before applying PCA to ensure meaningful and stable results.

Practical application: You are going to perform PCA with the CCPP dataset.

1.Calculate the variance of each variable and interpret the results. Do you think it is necessary to standardize the variables before performing PCA for this dataset? Why?

```
# Step 3: Calculate the percentage of variance explained by each component and plot
pve = pca.explained_variance_ratio_
cumulative_pve = np.cumsum(pve)
pve
```

```
array([0.66437362, 0.18291829, 0.11763241, 0.02793114, 0.00714454])
```

- Most of the variability in the dataset is captured by first two principal components (**around 84.73%).**
- Including more principal components increases cumulative explained variance, but beyond a certain point and additional components contribute relatively little to overall explanation of variability.
- The choice of how many principal components to retain depends on desired level of explained variance. In practice, a common approach is to choose number of components that collectively explain a sufficiently high percentage of total variance, often aiming for a threshold like 90% or 95%. So variability captured by all components will make more than 95% so we can take all components.

Standardize the variables

In most cases standardizing variables before PCA is considered good practice and as it helps overcome issues related to scale and provides equal weight to variables and improves interpretability and enhances numerical stability. However there might be cases where standardization is not necessary and such as when variables are already on similar scales and relative scales are not critical to analysis.

Before applying PCA it's advisable to assess the characteristics of your dataset and consider whether standardization is appropriate based on nature of your variables and goals of your analysis.

2.Perform PCA using the appropriate function with the appropriate arguments and options considering your answer to the previous question. Analyze the output of the function. Interpret the values of the two first principal component loading vectors.

```
# Standardize the variables before performing PCA
scaler = StandardScaler()
df_standardized = scaler.fit_transform(numeric_df)
```

```
# Perform PCA
pca = PCA()
pca.fit(df_standardized)
```

```
▾ PCA
PCA()
```

```
# Step 2: Analyze the output of the PCA - the first two principal component loading vectors
loadings = pca.components_
```

```
loadings
```

```
array([[ 0.5344631 ,  0.49018343, -0.33409283, -0.29337725, -0.52571967],
       [-0.08033939,  0.07824417, -0.5983973 ,  0.79026642, -0.0694484 ],
       [-0.07825384, -0.4503986 , -0.71253389, -0.48330099,  0.22300907],
       [-0.39882432,  0.74208676, -0.14887524, -0.18422501,  0.48388398],
       [-0.73668872, -0.00681151, -0.02083896, -0.14795294, -0.65948389]])
```

Variance Results:

Positive Loadings:

The first principal component has strong positive loadings on the first variable (0.5344631) and second variable (0.49018343).

This indicates that first principal component increases when there are increases in values of first and second variables.

Negative Loading:
The first principal component has a strong negative loading on fifth variable (-0.52571967). This suggests that first principal component decreases when there is an increase in fifth variable.
Overall Interpretation:
The first principal component represents a contrast between sets of variables.
It increases with first two variables and decreases with fifth variable.
In practical terms, this component captures commonality or pattern shared by first two variables while contrasting with pattern represented by fifth variable.

3. Calculate the percentage of variance explained (PVE) by each component? Plot the PVE explained by each component, as well as the cumulative PVE. How many components would you keep? Why?

```
# Step 2: Analyze the output of the PCA - the first two principal component loading v
loadings = pca.components_
```

```
loadings
```

```
array([[ 0.5344631 ,  0.49018343, -0.33409283, -0.29337725, -0.52571967],
       [-0.08033939,  0.07824417, -0.5983973 ,  0.79026642, -0.0694484 ],
       [-0.07825384, -0.4503986 , -0.71253389, -0.48330099,  0.22300907],
       [-0.39882432,  0.74208676, -0.14887524, -0.18422501,  0.48388398],
       [-0.73668872, -0.00681151, -0.02083896, -0.14795294, -0.65948389]])
```

```
######  First Principal Component (First Row):The first principal component has stron
```

```
# Step 3: Calculate the percentage of variance explained by each component and plot
pve = pca.explained_variance_ratio_
cumulative_pve = np.cumsum(pve)
pve
```

```
array([0.66437362, 0.18291829, 0.11763241, 0.02793114, 0.00714454])
```

```
cumulative_pve
```

```
array([0.66437362, 0.84729191, 0.96492432, 0.99285546, 1.        ])
```

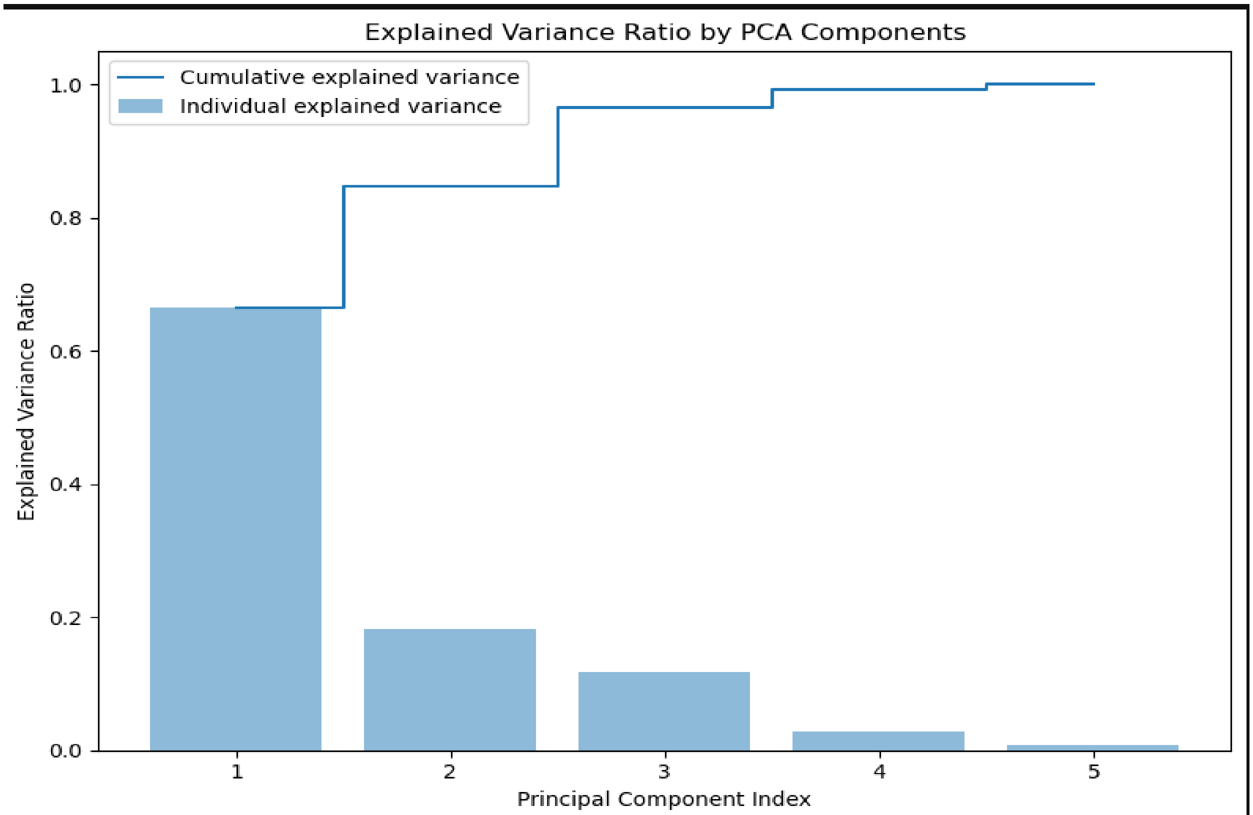
PVE:
First Component: 66.44%
Second Component: 18.29%
Third Component: 11.76%

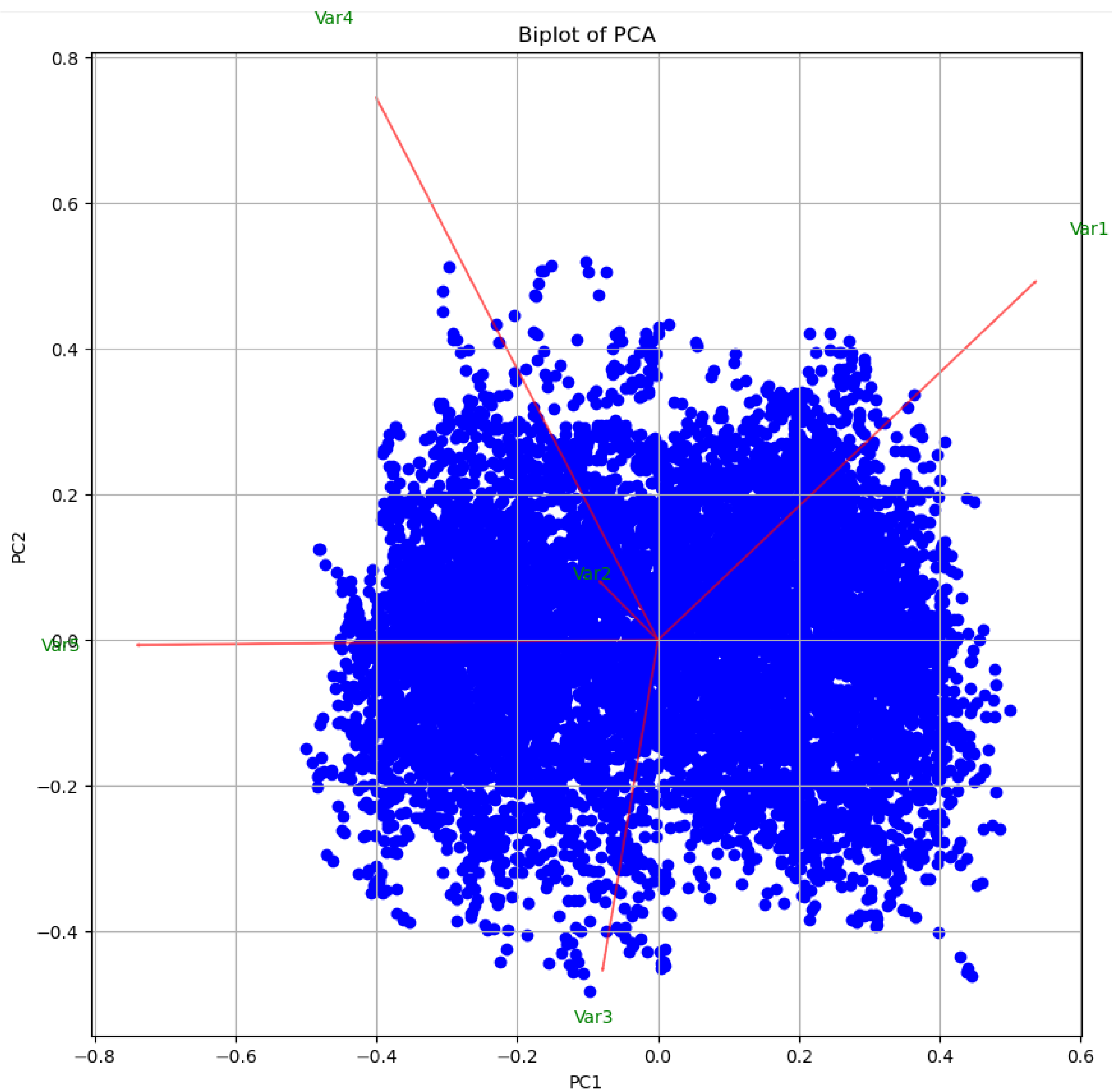
Fourth Component: 2.79%
Fifth Component: 0.71%

And the cumulative PVE:
After 1st Component: 66.44%
After 2nd Component: 84.73%
After 3rd Component: 96.49%
After 4th Component: 99.29%
After 5th Component: 100.00%

Given these results if we follow a common threshold of 95% for cumulative explained variance and we can see that first three components together account for approximately 96.49% of the variance. This surpasses the 95% threshold and which means that keeping first three components would be sufficient to retain most of information present in original data while reducing the dimensionality from five to three.
Therefore, you should keep all principal components because they allow you to reduce the data's complexity without losing significant information.

4.Use a biplot with a correlation circle to display both the principal component scores and the loading vectors in a single plot. Interpret the results.





Results:

- Variables represented by vectors that are close to each other and in same direction (like Var1 and Var2) are positively correlated.
- The first principal component (PC1) is strongly influenced by Var1 and Var2, and to a lesser extent by Var3, Var4, and Var5, given their projection lengths onto PC1.
- The second principal component (PC2) seems to be most strongly influenced by Var4, as indicated by length of its vector in the direction of PC2.
- The distribution of data points suggests variability in the dataset, with PC1 and PC2 explaining a significant portion of this variability and as indicated by their respective PVEs.

Linear Regression

Theoretical question: Let us suppose that we fit a linear regression model to explain Y as a linear function of two variables X1 and X2. Let us denote R2 the associated coefficient of determination. Interpret R2 What is the range of values that can be taken by R2? If we denote r1 and r2 the coefficient of correlation between X and Y and the coefficient of correlation between X2 and Y respectively. What is the relationship between R2 and r1 and r2?

R^2 (Coefficient of Determination):
Interpretation: Measures the proportion of variance in dependent variable (Y) explained by independent variables (X_1 and X_2) in a linear regression model.
Range:
0≤R2≤1 (0 indicates no explanatory power, 1 indicates a perfect fit).
Relationship Between R^2, r_1, and r_2:
R2=r1^2+r2^2 - 2 . r1 .r2.px1.x2
Interpretation: Highlights individual explanatory power of X1 and X2(r1^2+r2^2) and accounts for shared variance (- 2 . r1 .r2.px1.x2)
Special Cases:
Uncorrelated(px1x2=0) :R^2 =r1^2 + r2^2
Perfectly correlated(px1x2=+-0) :R^2 =r1^2 + r2^2 +- 2.r1.r2
Conclusion:
R^2 provides insights into how well model captures variability in dependent variable.
The relationship formula highlights interplay between individual and shared explanatory power of independent variables in a multiple regression context.

Practical application
In this part, you are going to perform linear regression using the electrical power output EP as the target variable.
Calculate the correlation coefficient matrix for the whole dataset.Comment on the results. Which variable is the most correlated with the target EP?

```python
import pandas as pd
# Load the dataset
df = pd.read_csv('CCPP_data.txt', delimiter='\t')

# Calculate the correlation coefficient matrix for the whole dataset
correlation_matrix = df.corr()

# Identify the variable that is most correlated with the target variable EP (assuming EP is the last column)
target_variable = 'EP'  # Assuming 'EP' is the name of the target variable column
most_correlated_variable = correlation_matrix.iloc[:-1][target_variable].abs().idxmax()
most_correlated_value = correlation_matrix.loc[most_correlated_variable, target_variable]

correlation_matrix, most_correlated_variable, most_correlated_value
```

```
(          AT         V        AP        RH        EP
 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
 EP -0.948128 -0.869780  0.518429  0.389794  1.000000,
 'AT',
 -0.9481284704167616)
```

The variable that is most correlated with target variable EP (Electrical Power Output) is Ambient Temperature (AT), with a correlation coefficient of approximately -0.948. This strong negative correlation suggests that as the ambient temperature increases and the electrical power output tends to decrease significantly.

Fit a simple linear regression model using as target variable EP, denoted Y, and as feature variable the most correlated variable to it that you identified in the previous question, denoted X:

Y = 3o+B2X +(1)

```python
# Import necessary libraries for linear regression and statistical analysis
import statsmodels.api as sm

# Prepare the data
X = df[['AT']]  # Independent variable (Ambient Temperature)
y = df['EP']    # Dependent variable (Electrical Power Output)

# Add a constant to the independent variable to represent the intercept
X_with_intercept = sm.add_constant(X)

# Fit the OLS model
model = sm.OLS(y, X_with_intercept).fit()

# Get the coefficient estimate (β1) and the intercept (β0)
coefficient_estimate = model.params['AT']
intercept_estimate = model.params['const']

# Calculate the 95% confidence interval for the slope (β1)
confidence_interval = model.conf_int(alpha=0.05).loc['AT']

# Perform the t-test for the slope coefficient and check if it is significantly different from 0
p_value = model.t_test([0, 1]).pvalue

# Get the R-squared value
r_squared = model.rsquared

# Compile the results
results = {
    'coefficient_estimate': coefficient_estimate,
    'intercept_estimate': intercept_estimate,
    '95%_confidence_interval': confidence_interval,
    'p_value_for_slope': p_value,
    'r_squared': r_squared
}

results
```

```
{'coefficient_estimate': -2.1713199585177896,
 'intercept_estimate': 497.03411989276725,
 '95%_confidence_interval': 0   -2.18591
 1   -2.15673
 Name: AT, dtype: float64,
 'p_value_for_slope': array(0.),
 'r_squared': 0.8989475964148236}
```

1. What are the coefficient estimates? Interpret coefficient estimate B1.
Coefficient Estimates:

Coefficient for AT (β1): -2.1713

Intercept (β0): 497.0341

2. Give the general expression of a 1-a confidence interval for the parameter B1. Calculate the 95% confidence interval for this coefficient. Interpret the results.
95% Confidence Interval for the Coefficient (β1):

Lower Bound: -2.1859
Upper Bound: -2.1567
This means we are 95% confident that the true value of the slope (β1) lies within this interval.

3. Elaborate the zero slope hypothesis test for coefficient 3, and conclude if there is an impact of the predictor on the number of shares. Is 3 significantly non zero?
Zero Slope Hypothesis Test (β1):
The p-value for the slope coefficient is 0.0, indicating that slope is significantly different from zero. Therefore, we can conclude that there is a statistically significant impact of Ambient Temperature on Electrical Power Output.

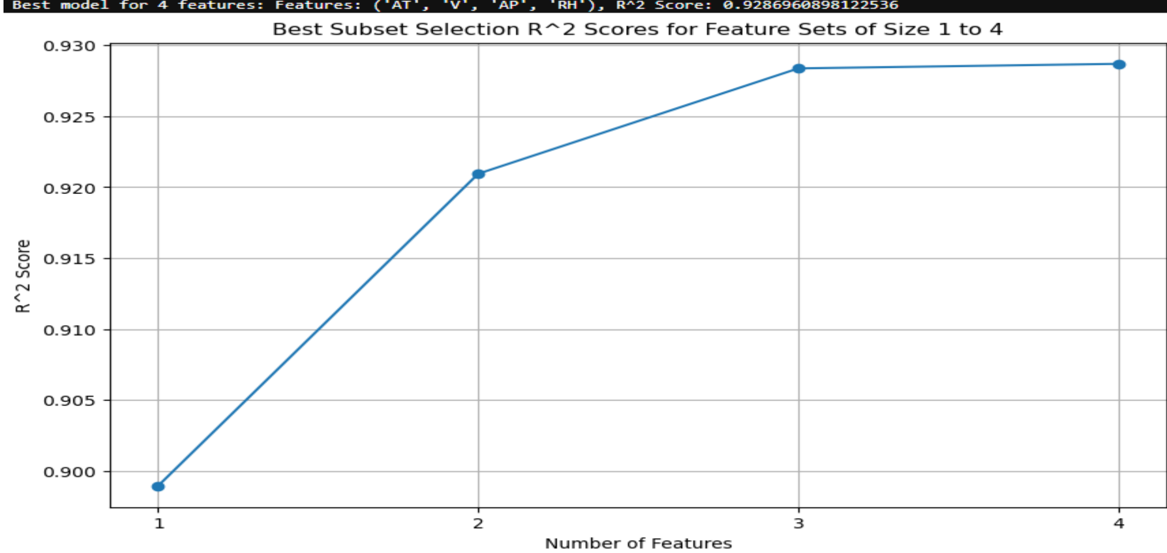
4. What is the value of the coefficient of determination R2? Interpret this result. Is this model suitable to predict the number of shares?
Coefficient of Determination (R^2):
R^2 value: 0.8989
This suggests that approximately 89.89% of variability in EP can be explained by model and which is a high degree of explanation and indicates that model fits the data well.
Given the high R^2 value and the significant p-value for slope and this model is suitable for predicting Electrical Power Output based on Ambient Temperature.

Feature selection for multiple linear regression
Now you are going to fit multiple linear regression models to predict the target variable EP as a function of two or more other predictors.
In some practical situations it is suitable to select only a subset of the predictors instead of considering all the available variables, since some variables can have no or just little statistical significance to predict the target. The best subset selection method consists in fitting a separate least squares regression for each possible combination of the available features1. Perform the following tasks and answer the questions:

1.Use Best Subset Selection method to select the best model for any possible number of features ranging from 1 to 4. Plot the curve R versus the number of features. Then, select the best model. That is, the model for which the adjusted coefficient of determination R2 is the highest.

```python
from itertools import combinations
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'df' is the DataFrame with the dataset loaded and 'EP' is the target variable
X = df.drop(columns='EP')  # Features
y = df['EP']               # Target variable

# Initialize dictionary to store the best model for each number of features
best_models = {}
r2_scores = []

# Try every possible combination of features from 1 to 4
for num_features in range(1, 5):
    best_r2 = -np.inf
    best_model = None
    for combo in combinations(X.columns, num_features):
        # Create a model with the given combination of features
        model = LinearRegression()
        model.fit(X[list(combo)], y)
        r2 = r2_score(y, model.predict(X[list(combo)]))

        # If the new model is better, update the best model for this number of features
        if r2 > best_r2:
            best_r2 = r2
            best_model = (combo, model)
    r2_scores.append(best_r2)
    best_models[num_features] = best_model

# Print the best models for each number of features
for num_features, (features, model) in best_models.items():
    print(f"Best model for {num_features} features: Features: {features}, R^2 Score: {r2_scores[num_features - 1]}")

# Plot the curve of R^2 versus the number of features
plt.figure(figsize=(10, 6))
plt.plot(range(1, 5), r2_scores, marker='o')
plt.xlabel('Number of Features')
plt.ylabel('R^2 Score')
plt.title('Best Subset Selection R^2 Scores for Feature Sets of Size 1 to 4')
plt.xticks(range(1, 5))
plt.grid(True)
plt.show()
```

```
Best model for 1 features: Features: ('AT',), R^2 Score: 0.8989475964148236
Best model for 2 features: Features: ('AT', 'RH'), R^2 Score: 0.9209480760095283
Best model for 3 features: Features: ('AT', 'V', 'RH'), R^2 Score: 0.928374821658302
Best model for 4 features: Features: ('AT', 'V', 'AP', 'RH'), R^2 Score: 0.9286960898122536
```



The model with highest R^2 score is one that uses 4 features. This model has an R^2 score of approximately 0.929, which is the highest on graph indicating that it explains most variance in Electrical Power Output (EP) within your dataset. Therefore, you should select the model with 4 features for the best predictive performance based on R^2.

2.How many features did you keep? Which ones?
With all 4 features
['AT', 'V', 'AP', 'RH']

With one target variable
['EP']

3.Why is it more appropriate to use the adjusted coefficient of determination R2 instead of the coefficient of determination R2 when comparing two models with different numbers of predictors?
Penalty for Complexity:
R2: Will always increase or stay same as more predictors are added, even with weak relationships. This can lead to overfitting.
Adj R2: Includes a penalty for number of predictors, preventing automatic increase with new predictors.
Comparability:
Adj R2: More comparable across models with different predictor counts due to penalty. Reliable for model selection with varying complexities.
Bias Correction:
R2: Can be biased towards models with more predictors, potentially favoring them.
Adj R2: Corrects for this bias and offering a balanced view of model performance relative to number of predictors.

4. For the selected model, what are the values of the coefficient estimates? Interpret them. What is the value of the coefficient of determination R2? Interpret this value.
The R^2 value of 0.929 suggests that approximately 92.9% of variability in Electrical Power Output (EP) can be explained by combined variation of 'AT', 'V', 'AP', and 'RH'. This is a high R^2 value, indicating a strong fit of the model to data. In other words and the model explains a large proportion of the variance in the electrical power output and which suggests that the predictors included in the model are relevant and provide a good predictive capability.

```python
import statsmodels.api as sm

# Assuming 'df' is your DataFrame with the dataset already loaded
X = df[['AT', 'V', 'AP', 'RH']]  # Features
y = df['EP']                     # Target variable

# Add a constant to the model (the intercept term)
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Get the coefficient estimates and R-squared value
coefficients = model.params
r_squared = model.rsquared

# Output the results
print(coefficients)
print(f'R-squared: {r_squared}')
```

```
const    454.609274
AT        -1.977513
V         -0.233916
AP         0.062083
RH        -0.158054
dtype: float64
R-squared: 0.9286960898122537
```

5.For the selected model, perform the zero slope hypothesis test for all the coefficients except Bo and conclude.

```python
import statsmodels.api as sm

# Assuming 'df' is your DataFrame with the dataset already loaded
X = df[['AT', 'V', 'AP', 'RH']]  # Features
y = df['EP']                     # Target variable

# Add a constant to the model (the intercept term)
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Hypothesis testing for each coefficient (excluding the intercept)
for i, feature in enumerate(X.columns[1:]):  # Exclude the first column (intercept) from testing
    hypothesis_test_result = model.t_test([0, 0, 0, 0, 1], use_t=True)  # Testing the i-th coefficien
    p_value = hypothesis_test_result.pvalue
    print(f'Hypothesis test for {feature}: p-value = {p_value}')

    # Check if p-value is less than the significance level (e.g., 0.05)
    significance_level = 0.05
    if p_value < significance_level:
        print(f'Reject the null hypothesis for {feature}')
    else:
        print(f'Fail to reject the null hypothesis for {feature}')

# Print the summary of the model (includes coefficients, standard errors, t-values, and p-values)
print(model.summary())
```

```
Hypothesis test for AT: p-value = 3.104584420261387e-293
Reject the null hypothesis for AT
Hypothesis test for V: p-value = 3.104584420261387e-293
Reject the null hypothesis for V
Hypothesis test for AP: p-value = 3.104584420261387e-293
Reject the null hypothesis for AP
Hypothesis test for RH: p-value = 3.104584420261387e-293
Reject the null hypothesis for RH
                            OLS Regression Results
==============================================================================
Dep. Variable:                     EP   R-squared:                       0.929
Model:                            OLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                 3.114e+04
Date:                Tue, 16 Jan 2024   Prob (F-statistic):               0.00
Time:                        00:14:27   Log-Likelihood:                -28088.
No. Observations:                9568   AIC:                         5.619e+04
Df Residuals:                    9563   BIC:                         5.622e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        454.6093      9.749     46.634      0.000     435.500     473.718
AT            -1.9775      0.015   -129.342      0.000      -2.007      -1.948
V             -0.2339      0.007    -32.122      0.000      -0.248      -0.220
AP             0.0621      0.009      6.564      0.000       0.044       0.081
RH            -0.1581      0.004    -37.918      0.000      -0.166      -0.150
==============================================================================
Omnibus:                      892.002   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4086.777
Skew:                          -0.352   Prob(JB):                         0.00
Kurtosis:                       6.123   Cond. No.                     2.13e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.13e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation:
The model is a good fit and explains a high percentage of the variance.
All predictor variables (AT, V, AP, RH) are statistically significant in predicting the dependent variable (EP).
The coefficients provide insights into the direction and magnitude of relationships between predictors and dependent variables.
Residual analysis suggests some departure from normality and there might be autocorrelations in residuals. Further investigation may be needed.
Note:
The extremely low p-values in the hypothesis tests suggest strong evidence against null hypothesis and supporting the inclusion of all predictors in the model.

6.For the selected model, make a prediction of the electrical energy production given the following conditions: temperature of 22°C, atmospheric pressure 1010 mbar, relative humidity 80% and exhaust vacuum 75.

```
import pandas as pd
from sklearn.linear_model import LinearRegression


df = pd.read_csv('CCPP_data.txt', delimiter='\t')

# Split the dataset into features (X) and target variable (y)
X = df[['AT', 'V', 'AP', 'RH']]  # Features
y = df['EP']                      # Target variable

# Initialize and train the linear regression model on the entire dataset
model = LinearRegression()
model.fit(X, y)

# Given conditions for prediction
new_data = {'AT': 22, 'V': 75, 'AP': 1010, 'RH': 80}
new_data_df = pd.DataFrame([new_data])  # Convert the dictionary to a DataFrame

# Make a prediction
predicted_energy_production = model.predict(new_data_df)

# Output the prediction
print(f'Predicted Electrical Energy Production: {predicted_energy_production[0]}')
```

```
Predicted Electrical Energy Production: 443.61969926099573
```

The Predicted Electrical Energy Production Is 443.61969926099573.