CONCRETE COMPRESSIVE STRENGTH REGRESSION

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

VAIBHAV KOHLI

Registration number

12101119

Supervisor

Sajjad Manzoor Mir

for

CSM423

Machine Learning 2

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month 10 Year 2024

DECLARATION STATEMENT

I hereby declare that the research work reported in the dissertation/dissertation proposal entitled " CONCRETE COMPRESSIVE STRENGTH REGRESSION" in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mr. Sajjad Manzoor Mir. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University's Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

Signature of Candidate

Vaibhav Kohli

Reg No: 12101119

Roll No: 46

ABSTRACT

This project explores the prediction of concrete compressive strength using a regression model based on a dataset containing various concrete mixture components and their curing time. The aim is to understand the relationships between the ingredients—such as cement, slag, fly ash, water, and aggregates—and how they contribute to the overall strength of the concrete. By employing regression techniques, the project seeks to identify key predictors of strength and provide an accurate model to forecast compressive strength based on the mixture's composition and age. The results from this study could benefit both academic research and practical applications in the construction industry, offering insights into optimizing concrete formulations for improved durability and performance.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who supported me throughout the development of this concrete analysis notebook. The successful completion of this project would not have been possible without the kind assistance, encouragement, and courage of many individuals.

I extend my heartfelt thanks to my supervisor for their invaluable guidance and expertise, which significantly contributed to the depth and rigor of this analysis. I am also deeply appreciative of all those who shared their ideas and engaged in critical evaluations, enriching the quality of this project.

Finally, I would like to thank my parents and family for their unwavering support and encouragement throughout this journey. Their belief in my abilities and their continuous support have been a tremendous source of strength.

TABLE OF CONTENTS

INTRODUCTION

Concrete is a fundamental material used in construction, and its compressive strength is a critical property that determines its durability and suitability for various applications. The compressive strength of concrete is influenced by multiple factors, including the types and proportions of ingredients used, as well as the age of the concrete. Understanding how these variables interact and contribute to concrete strength is essential for optimizing concrete mixtures and ensuring the material meets the required performance standards.

This project focuses on developing a regression model to predict the compressive strength of concrete based on a dataset that includes the composition of various concrete mixtures and their curing times. The key ingredients examined in this study include cement, blast furnace slag, fly ash, water, superplasticizer, coarse and fine aggregates, as well as the age of the concrete. By analyzing these variables, the project aims to establish a robust predictive model that can accurately forecast concrete compressive strength and provide valuable insights into the relationship between the mixture's composition and its mechanical properties.

OBJECTIVE

The primary objective of this project is to develop a machine learning-based system to predict concrete compressive strength based on the mixture's ingredients and curing time. The goal is to enhance accuracy and efficiency in predicting concrete strength, helping optimize material formulations in construction. Specific objectives include:

Data Preparation: To preprocess the dataset by handling missing values, scaling numerical variables, and ensuring that all features, such as cement, slag, fly ash, water, aggregates, superplasticizer, and age, are properly formatted for model training.

Exploratory Data Analysis: To perform exploratory analysis to identify relationships between variables, such as the water-cement ratio and strength, and visualize data distributions for better understanding.

Model Training and Evaluation: To train various regression models, including Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regressor. These models will be evaluated based on performance metrics such as Root Mean Squared Error (RMSE) and R^2 using cross-validation to ensure generalization.

Hyperparameter Tuning: To fine-tune model parameters using GridSearchCV and RandomizedSearchCV to improve the accuracy and robustness of the predictive models.

Model Selection: To select the best-performing regression model based on its predictive accuracy, generalization ability, and consistency across cross-validation results.

Model Deployment: To save the selected model for future deployment in real-world applications where it can assist engineers in optimizing concrete mixtures for specific strength requirements.

KNOWING THE INPUT VARIABLES

1. Cement

Cement acts as the primary binder in concrete, initiating the hydration process when mixed with water, which leads to hardening and strength development. While more cement typically increases strength, excessive amounts can cause shrinkage and cracking.

2. Blast Furnace Slag

A byproduct of steel manufacturing, slag is used as a partial cement replacement. It improves concrete's workability, reduces heat during curing, and enhances long-term strength, although it may slow initial strength development.

3. Fly Ash

Fly ash, derived from coal combustion, is another cement substitute that enhances workability, reduces water demand, and improves long-term strength, but like slag, it can delay early strength gain.

4. Water

Water initiates the hydration process, binding the mix together. The water-cement ratio is crucial—lower ratios generally increase strength, but too little water can hinder workability.

5. Superplasticizer

Superplasticizers improve concrete's workability without the need for additional water. They allow for stronger mixes by reducing the water content, especially in high-performance concrete.

6. Coarse Aggregate

Coarse aggregates, such as gravel, provide bulk and strength to the concrete. The size, shape, and quality of the aggregate significantly influence the final strength and durability of the mixture.

7. Fine Aggregate

Fine aggregates, like sand, fill the gaps between coarse aggregates, contributing to the concrete's overall density and workability. A balanced mix of fine and coarse aggregates is essential for a smooth finish and durable concrete.

8. Age:

Age represents the curing time of concrete, which directly affects strength. As concrete cures, it gains strength, with most of it developing within the first 28 days, making this a key factor in strength prediction.

The balance of these ingredients is key to producing concrete that meets specific strength requirements. A mixture with the correct proportions of cement, aggregates, water, and additives will result in concrete that has good workability and durability. The challenge is to optimize these variables for the desired compressive strength, factoring in the age at which the strength is measured.

CONCEPTS APPLIED

1. Linear Regression:

A simple model that finds the linear relationship between input features and the target (compressive strength). It minimizes the difference between predicted and actual values.

2. Ridge Regression:

A regularized version of linear regression that reduces overfitting by penalizing large coefficients, especially useful when there is multicollinearity among features.

3. Lasso Regression:

Another regularization technique performs feature selection by driving some coefficients to zero, simplifying the model and focusing on important variables.

4. Random Forest Regressor:

An ensemble method that combines predictions from multiple decision trees to model complex, non-linear relationships. It's robust to overfitting and works well on large datasets.

5. GridSearchCV:

A hyperparameter tuning technique that automates the search for the best model parameters through cross-validation, optimizing model performance.

6. Cross-Validation:

A method to assess model performance by splitting the dataset into training and validation sets multiple times, reducing the chance of overfitting.

7. Feature Scaling:

Ensures features contribute equally to the model by normalizing their ranges, preventing any single feature from dominating the model due to differing scales.

8. Mean Squared Error (MSE):

A common evaluation metric for regression models, measuring the average squared difference between predicted and actual values. Lower MSE means better model performance.

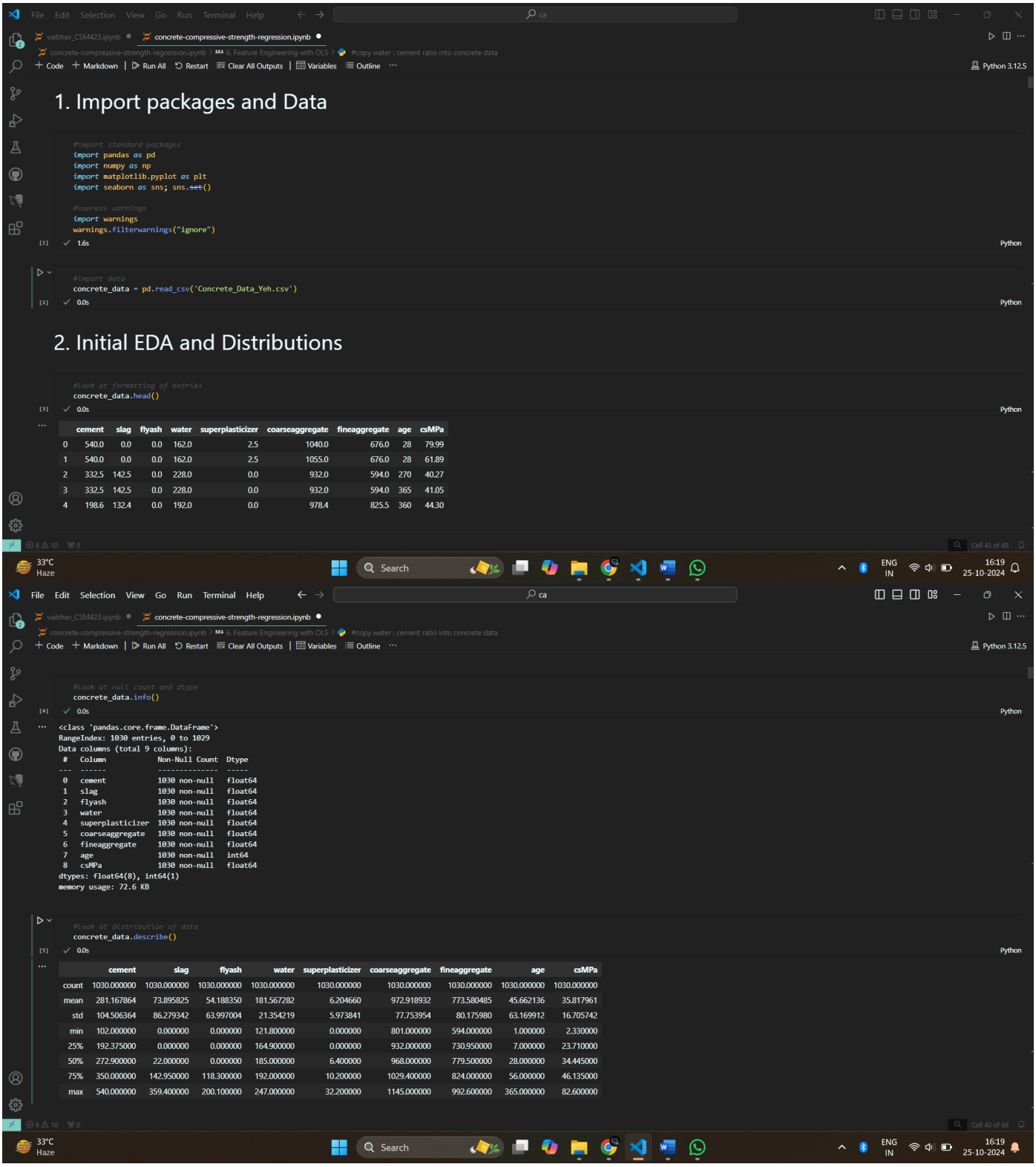
METHODOLOGY

1. **Data Collection**:
   - Gather the concrete dataset, which includes details of various concrete mixtures such as the quantities of cement, slag, fly ash, water, aggregates, superplasticizer, and the age of the concrete. This dataset serves as the foundation for predicting compressive strength.
2. **Data Preprocessing**:
   - **Handling Missing Values**: Identify any missing values in the dataset and handle them through techniques such as imputation (using mean or median) or removal of incomplete rows.
   - **Scaling Numerical Features**: Standardize the feature values (e.g., ingredient weights, age) using techniques like MinMaxScaler or StandardScaler to ensure uniformity during model training and avoid biases from features with larger magnitudes.
3. **Exploratory Data Analysis (EDA)**:
   - Perform EDA to visualize the relationships between the input variables and compressive strength. This includes plotting distributions, analyzing correlations between ingredients and strength, and identifying patterns in the data to guide model selection.
4. **Model Selection**:
   - Select appropriate regression models for predicting concrete compressive strength. Models considered include **Linear Regression**, **Ridge Regression**, **Lasso Regression**, and ensemble models like **Random Forest Regressor** and **Gradient Boosting Regressor**.
5. **Model Training and Evaluation**:
   - Split the dataset into training and test sets to evaluate model performance.
   - Train each selected model on the training data and assess their performance using metrics such as **Root Mean Squared Error (RMSE)**, **Mean Absolute Error (MAE)**, and $\mathbf{R^2}$.
   - Utilize **k-fold cross-validation** to ensure the model's generalization ability and reduce the risk of overfitting.
6. **Hyperparameter Tuning**:
   - Fine-tune model parameters using optimization techniques such as **GridSearchCV** or **RandomizedSearchCV** to find the best configuration for each model, enhancing performance and robustness.
7. **Model Deployment**:

- Save the best-performing model using serialization techniques like joblib or pickle for future deployment.
- Integrate the model into an interactive environment or an application that allows engineers to input concrete mixture details and predict compressive strength in real-time.

8. **Testing and Validation**:
   - Thoroughly test the deployed model to ensure its reliability in real-world applications. Validate the model predictions using a separate test set and compare its results against actual compressive strength outcomes.
9. **Documentation and Reporting**:
   - Document each phase of the project, including data preprocessing steps, EDA insights, model selection rationale, training procedures, hyperparameter tuning results, and the deployment process.
   - Compile a comprehensive report summarizing the methodology, findings, and potential applications for industry stakeholders.

CODE SNIPPETS



1. Import packages and Data

```
#import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

#supress warnings
import warnings
warnings.filterwarnings("ignore")
```

[1] ✓ 1.6s Python

```
#import data
concrete_data = pd.read_csv('Concrete_Data_Yeh.csv')
```

[2] ✓ 0.0s Python

2. Initial EDA and Distributions

```
#Look at formatting of entries
concrete_data.head()
```

[3] ✓ 0.0s Python

	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

```
#Look at null count and dtype
concrete_data.info()
```

[4] ✓ 0.0s Python

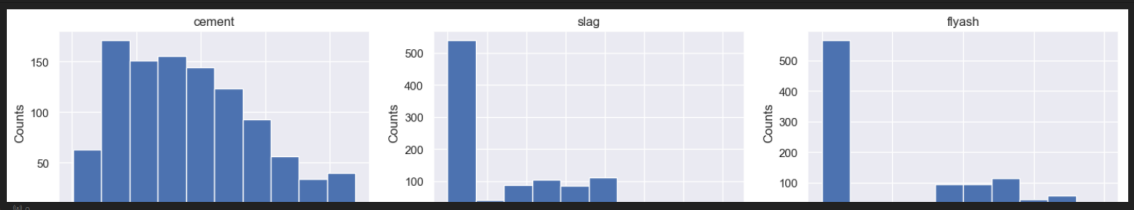
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   cement            1030 non-null   float64
 1   slag              1030 non-null   float64
 2   flyash            1030 non-null   float64
 3   water             1030 non-null   float64
 4   superplasticizer  1030 non-null   float64
 5   coarseaggregate   1030 non-null   float64
 6   fineaggregate     1030 non-null   float64
 7   age               1030 non-null   int64  
 8   csMPa             1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

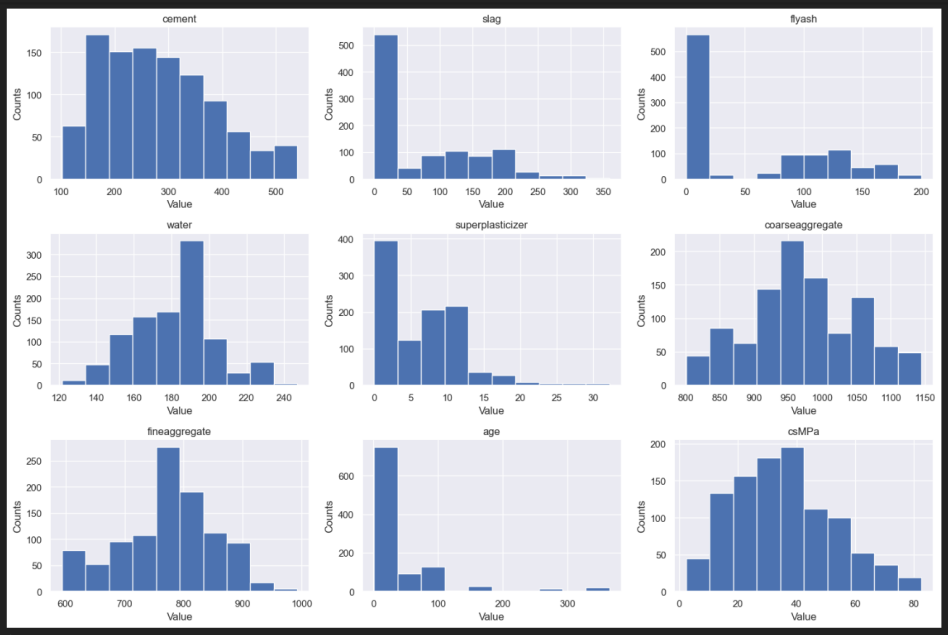
```
#Look at distribution of data
concrete_data.describe()
```

[5] ✓ 0.0s Python

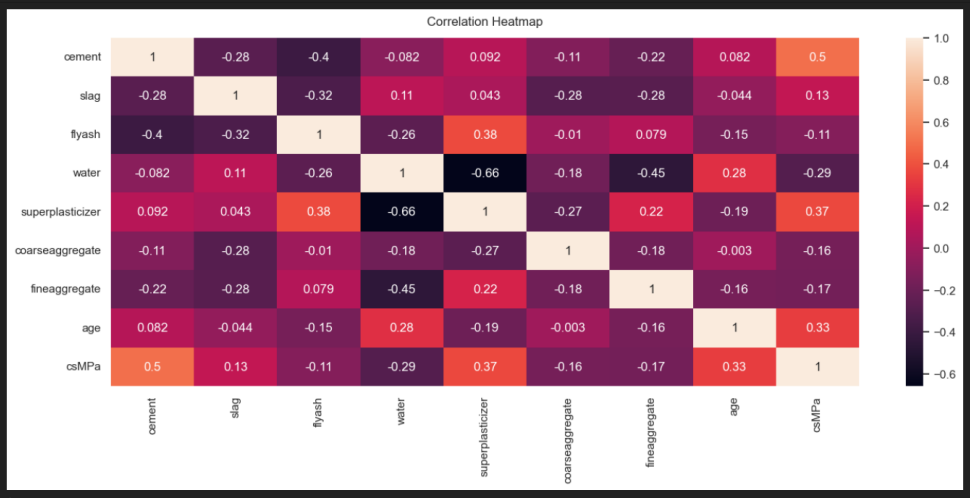
	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

```
import matplotlib.pyplot as plt

# Set the number of rows and columns for subplots
n_cols = 3  # Number of columns
n_rows = (len(concrete_data.columns) + n_cols - 1) // n_cols  # Calculate rows based on the number of columns

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 10))  # Adjust figsize for appropriate size

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

# Plot each column's histogram
for i, col in enumerate(concrete_data.columns):
    axes[i].hist(concrete_data[col])
    axes[i].set_title(col)
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Counts')

# Remove any empty subplots (in case there are more subplots than columns)
for i in range(len(concrete_data.columns), n_rows * n_cols):
    fig.delaxes(axes[i])

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



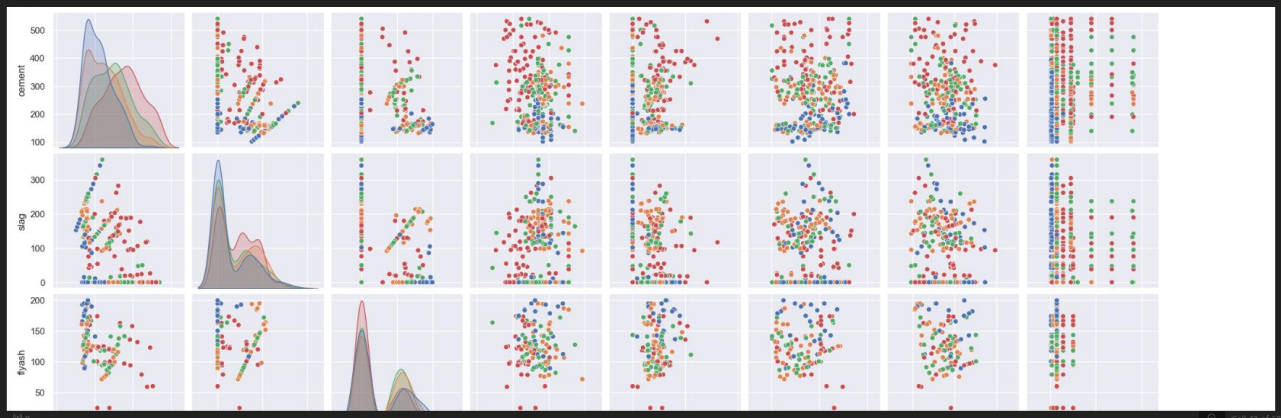


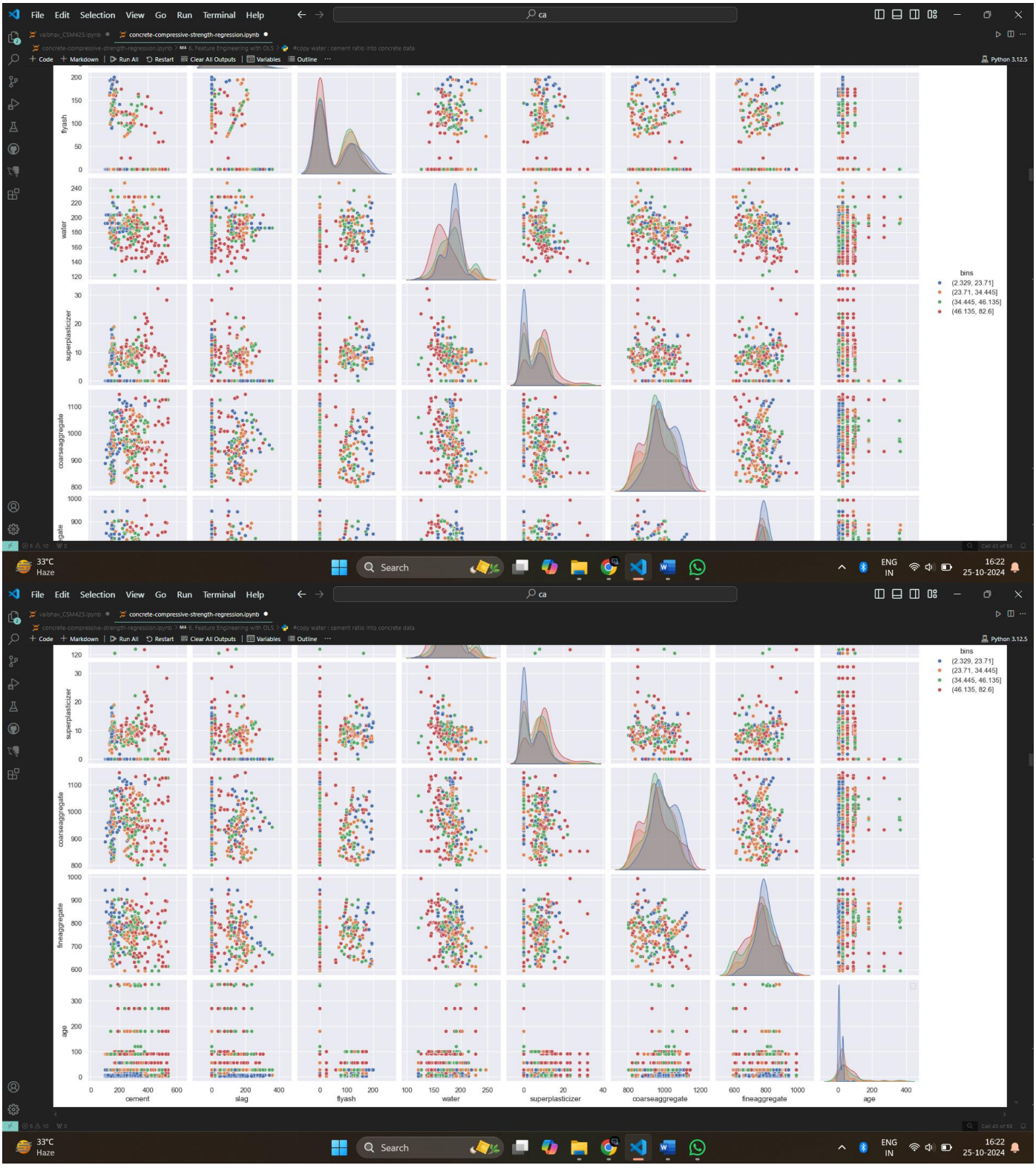
```
#heat map using Pearson's coefficient
plt.figure(figsize=(16, 6))
sns.heatmap(concrete_data.corr(), annot=True)
plt.title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```




```
#create bins from compressive strength
bins = pd.qcut(concrete_data['csMPa'], q=4)

#add bins to concrete df
concrete_data['bins']=bins
```

```
#look at how target is distributed among variables
sns.pairplot(concrete_data.loc[:, (concrete_data.columns != 'csMPa')], hue='bins')
plt.legend()
plt.show()
```

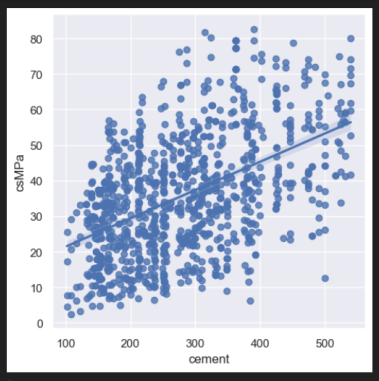



File Edit Selection View Go Run Terminal Help
ca
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
flyash
water
superplasticizer
coarseaggregate
bins
(2.329, 23.71]
(23.71, 34.445]
(34.445, 46.135]
(46.135, 82.6]
33°C
Haze
Search
ENG
IN
16:22
25-10-2024

File Edit Selection View Go Run Terminal Help
ca
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
superplasticizer
coarseaggregate
fineaggregate
age
cement
slag
flyash
water
superplasticizer
coarseaggregate
fineaggregate
age
bins
(2.329, 23.71]
(23.71, 34.445]
(34.445, 46.135]
(46.135, 82.6]
33°C
Haze
Search
ENG
IN
16:22
25-10-2024

```
#plot strongest linear correlation
sns.lmplot(x='cement', y='csMPa',data=concrete_data)
plt.show()
```

[10] ✓ 0.2s Python




```
#drop bins from concrete data
concrete_data = concrete_data.drop('bins', axis=1)
```

[11] ✓ 0.0s Python

```
#copy of variables and target
X = concrete_data.copy()
y = X.pop('csMPa')
```

[12] ✓ 0.0s Python

3. Mutual Information

```
#make a copy of features matrix for mutual information analysis
X_mi = X.copy()

#label encoding for categorical variables
for colname in X_mi.select_dtypes("object"):
    X_mi[colname], _ = X_mi[colname].factorize()

#all discrete features have int dtypes
discrete_features = X_mi.dtypes == object
```

[13] ✓ 0.0s Python

```
#some continuous variables also have int dtypes
discrete_features[X_mi.columns] = False
```

[14] ✓ 0.0s Python

```
#use regression since the target variable is continuous
from sklearn.feature_selection import mutual_info_regression

#define a function to produce mutual information scores
def make_mi_scores(X_mi, y, discrete_features):
    mi_scores = mutual_info_regression(X_mi, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X_mi.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores
```

[15]

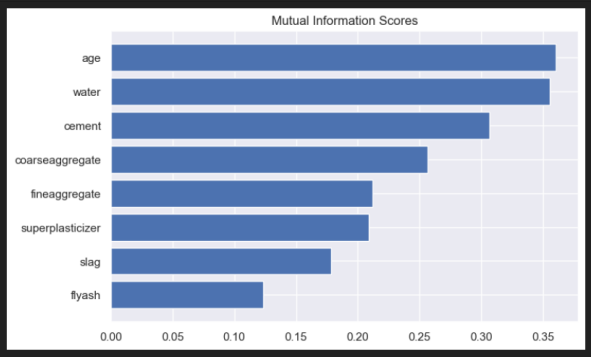
```python
#use regression since the target variable is continuous
from sklearn.feature_selection import mutual_info_regression

#define a function to produce mutual information scores
def make_mi_scores(X_mi, y, discrete_features):
    mi_scores = mutual_info_regression(X_mi, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X_mi.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

#compute mutual information scores
mi_scores = make_mi_scores(X_mi, y, discrete_features)
mi_scores
```

[15] ✓ 0.3s Python

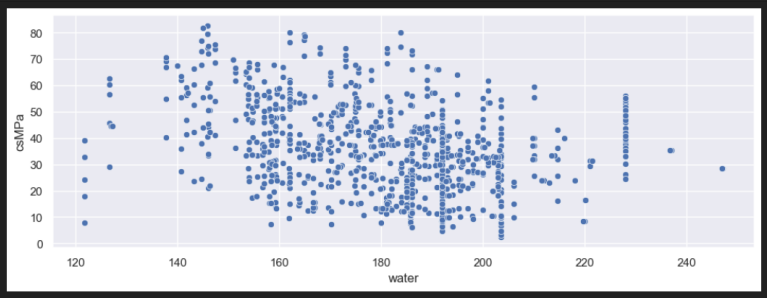
```
age                0.360557
water              0.355288
cement             0.306489
coarseaggregate    0.256595
fineaggregate      0.211687
superplasticizer   0.208601
slag               0.178566
flyash             0.123606
Name: MI Scores, dtype: float64
```

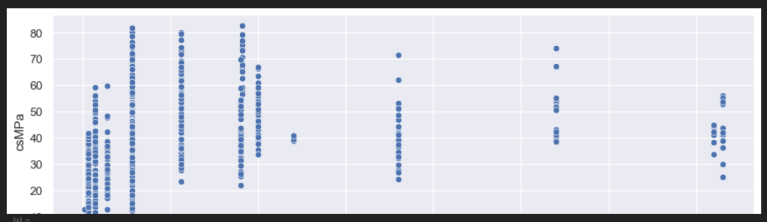
```python
#define a function to plot mutual information scores
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")

#plot the scores
plt.figure(dpi=100, figsize=(8, 5))
plot_mi_scores(mi_scores)
```

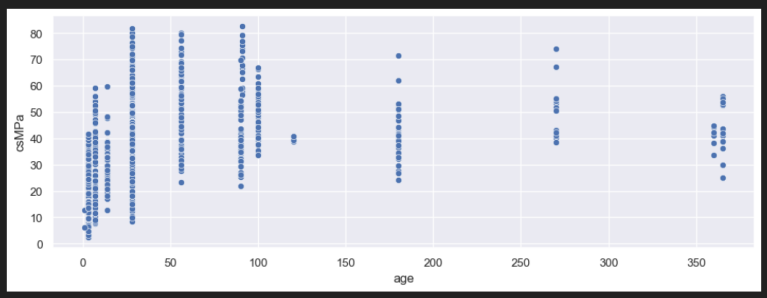
[16] ✓ 0.1s Python

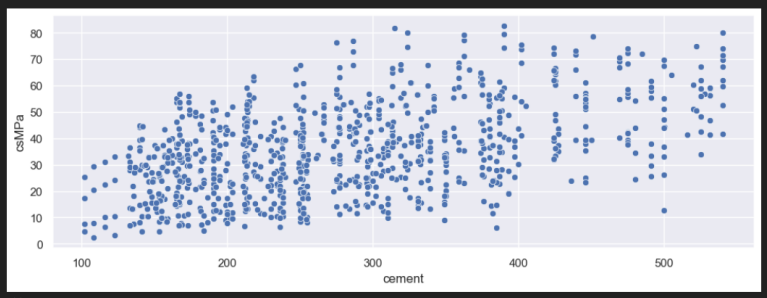


```
#plot top MI score predictors against target
for i in ['water', 'age', 'cement']:
    fig, ax = plt.subplots(figsize=(12,4))
    sns.scatterplot(x=X_mi[i], y=y, ax=ax)
    plt.show()
```









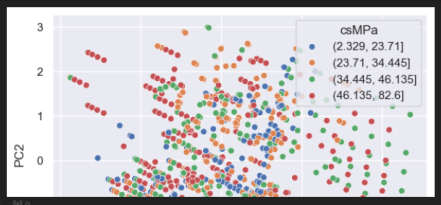
4. Principal Component Analysis

```
#copy features matrix for principal component analysis
X_for_PCA = X.copy()

#standardize
X_for_PCA_scaled = (X_for_PCA - X_for_PCA.mean(axis=0)) / X_for_PCA.std(axis=0)

from sklearn.decomposition import PCA

#create principal components
pca = PCA(len(X.columns))
X_pca = pca.fit_transform(X_for_PCA_scaled)

#convert to dataframe
component_names = [f"PC{i+1}" for i in range(X_pca.shape[1])]
X_pca = pd.DataFrame(X_pca, columns=component_names)
```

[18] ✓ 0.0s Python

```
#plot data using principal components
sns.scatterplot(x=X_pca.loc[:,'PC1'],y=X_pca.loc[:,'PC2'], hue=bins)
plt.show()
```

[19] ✓ 0.1s Python




```
#plot data using principal components
sns.scatterplot(x=X_pca.loc[:,'PC1'],y=X_pca.loc[:,'PC2'], hue=bins)
plt.show()
```

[19] ✓ 0.1s Python


```
#determine loadings
loadings = pd.DataFrame(
    pca.components_.T,  # transpose the matrix of loadings
    columns=component_names,  # so the columns are the principal components
    index=X.columns,  # and the rows are the original features
)
loadings
```

[20] ✓ 0.0s Python

```python
#determine loadings
loadings = pd.DataFrame(
    pca.components_.T,  # transpose the matrix of loadings
    columns=component_names,  # so the columns are the principal components
    index=X.columns,  # and the rows are the original features
)
loadings
```

[20] ✓ 0.0s — Python

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
cement	0.098401	-0.113737	0.814202	0.054297	-0.148206	-0.203142	-0.221844	0.446163
slag	0.177262	0.686053	-0.171794	0.362699	0.020932	0.304882	-0.228363	0.437384
flyash	-0.394662	-0.142948	-0.408221	-0.226751	-0.549631	-0.183267	-0.352463	0.381886
water	0.547004	0.053256	-0.213190	-0.296060	-0.070222	-0.365970	0.524275	0.388741
superplasticizer	-0.505945	0.282930	0.234597	0.037274	-0.354618	0.193294	0.664643	0.051750
coarseaggregate	0.037928	-0.629943	-0.174088	0.545805	0.033083	0.314559	0.226840	0.349320
fineaggregate	-0.401926	-0.019391	-0.004569	-0.385282	0.701237	0.092466	0.039026	0.433370
age	0.291479	-0.125981	0.100521	-0.527919	-0.228010	0.743908	-0.069367	0.012881

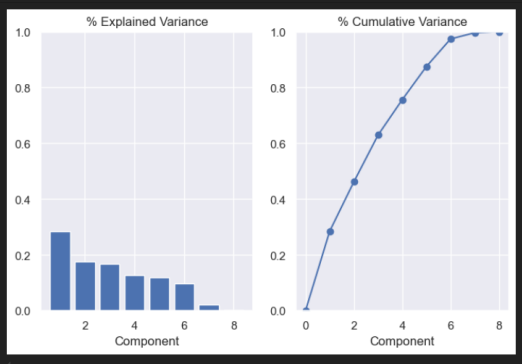
```python
#determine % explained variance and use % cumulative variance for elbow method to determine number of PCs

def plot_variance(pca, width=8, dpi=100):
    # Create figure
    fig, axs = plt.subplots(1, 2)
    n = pca.n_components_
    grid = np.arange(1, n + 1)
    # Explained variance
    evr = pca.explained_variance_ratio_
    axs[0].bar(grid, evr)
    axs[0].set(
        xlabel="Component", title="% Explained Variance", ylim=(0.0, 1.0)
    )
    # Cumulative Variance
    cv = np.cumsum(evr)
    axs[1].plot(np.r_[0, grid], np.r_[0, cv], "o-")
    axs[1].set(
        xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)
    )
```

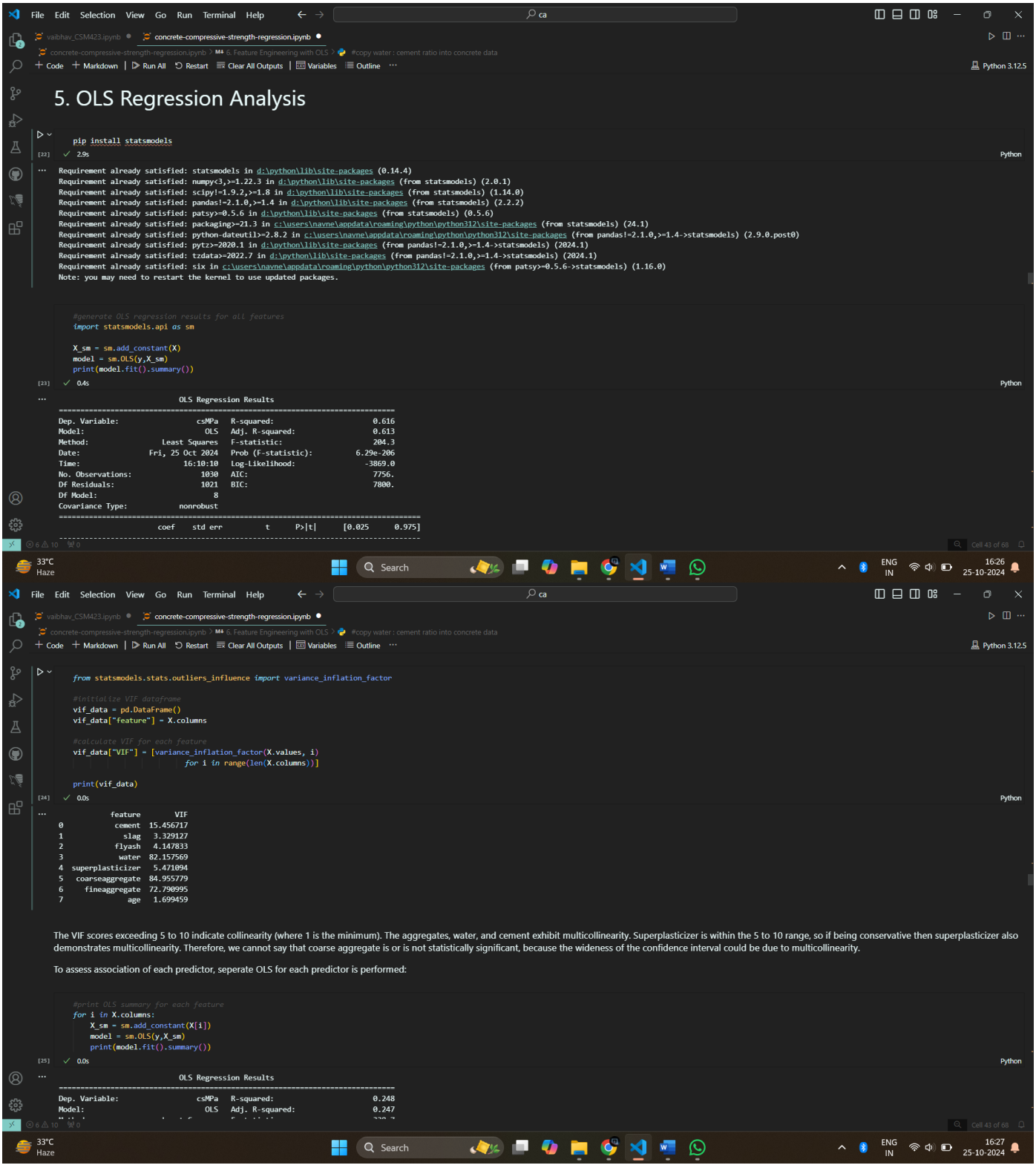
[21]


```python
    evr = pca.explained_variance_ratio_
    axs[0].bar(grid, evr)
    axs[0].set(
        xlabel="Component", title="% Explained Variance", ylim=(0.0, 1.0)
    )
    # Cumulative Variance
    cv = np.cumsum(evr)
    axs[1].plot(np.r_[0, grid], np.r_[0, cv], "o-")
    axs[1].set(
        xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)
    )
```


```python
    # Set up figure
    fig.set(figwidth=8, dpi=100)
    return axs

plot_variance(pca);
```

[21] ✓ 0.2s — Python





The VIF scores exceeding 5 to 10 indicate collinearity (where 1 is the minimum). The aggregates, water, and cement exhibit multicollinearity. Superplasticizer is within the 5 to 10 range, so if being conservative then superplasticizer also demonstrates multicollinearity. Therefore, we cannot say that coarse aggregate is or is not statistically significant, because the wideness of the confidence interval could be due to multicollinearity.

To assess association of each predictor, seperate OLS for each predictor is performed:

```
#print OLS summary for each feature
for i in X.columns:
    X_sm = sm.add_constant(X[i])
    model = sm.OLS(y,X_sm)
    print(model.fit().summary())
```

[25] ✓ 0.0s Python

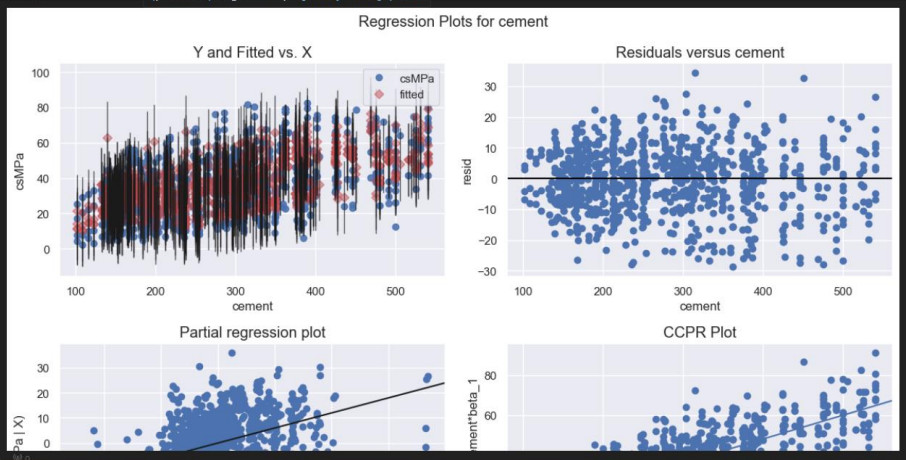
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  csMPa   R-squared:                       0.248
Model:                            OLS   Adj. R-squared:                  0.247
Method:                 Least Squares   F-statistic:                     338.7
Date:                Fri, 25 Oct 2024   Prob (F-statistic):           1.32e-65
Time:                        16:10:10   Log-Likelihood:                -4214.6
No. Observations:                1030   AIC:                             8433.
Df Residuals:                    1028   BIC:                             8443.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.4425      1.297     10.365      0.000      10.898      15.987
cement         0.0796      0.004     18.404      0.000       0.071       0.088
==============================================================================
Omnibus:                       19.696   Durbin-Watson:                   1.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.893
Skew:                           0.271   Prob(JB):                     0.000130
Kurtosis:                       2.649   Cond. No.                         861.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
...
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

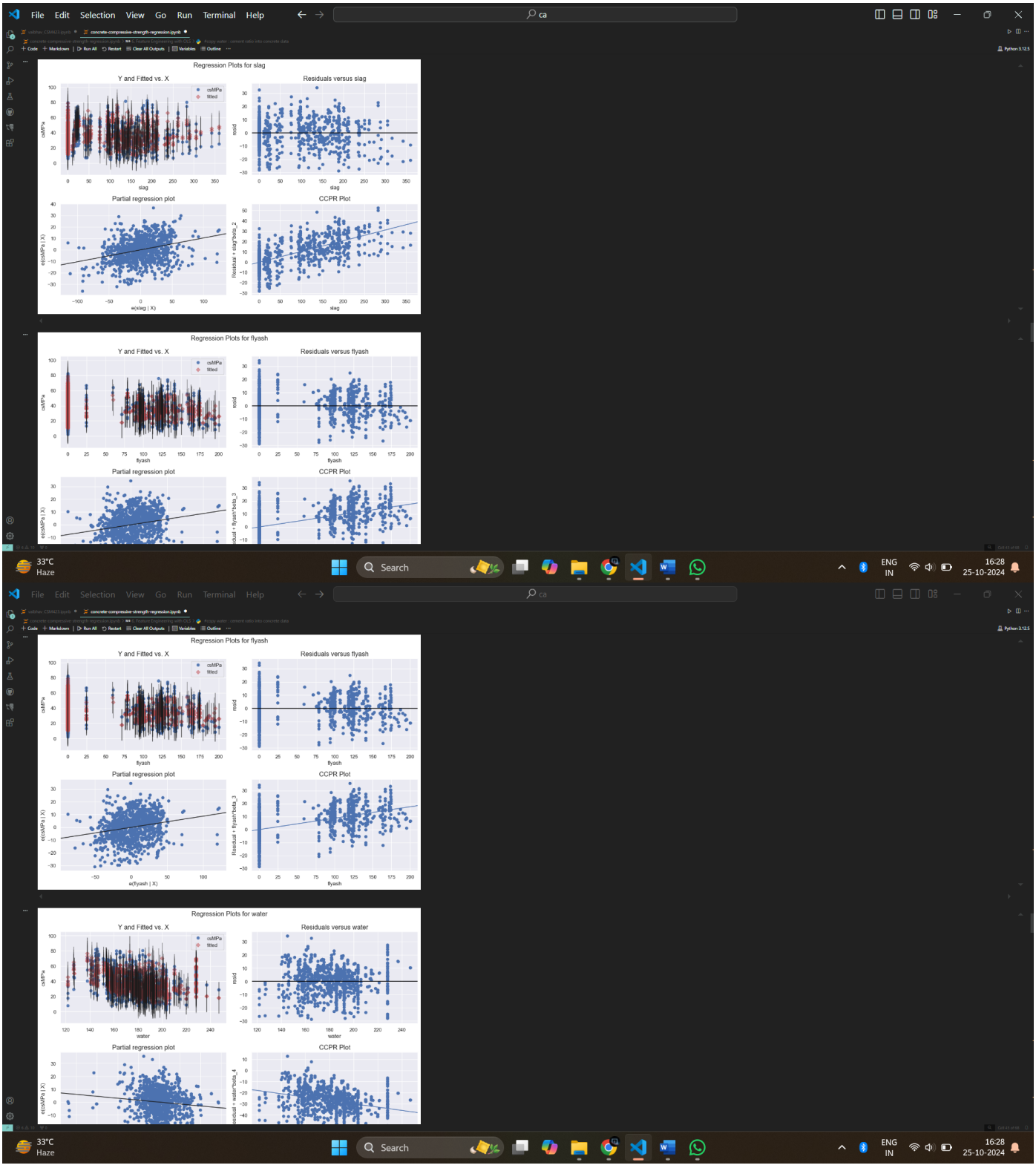
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

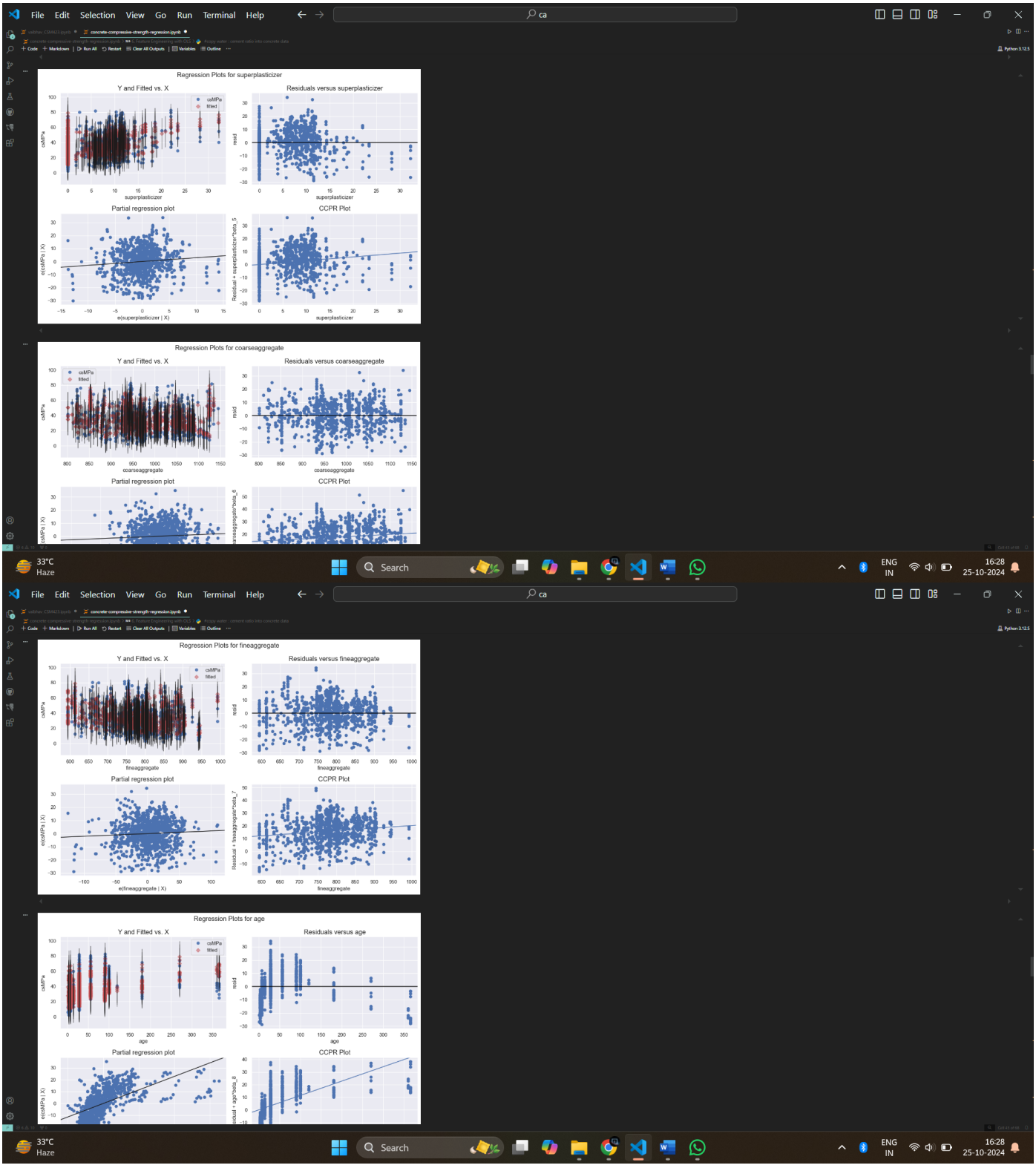
Looking at the p-value of the t-statistic: all variables have a strong association with compressive strength where fly ash has the weakest by 0.001.

```
from statsmodels.formula.api import ols

#fit multiple linear regression model
model = ols('csMPa ~ cement + slag + flyash + water + superplasticizer + coarseaggregate + fineaggregate + age', data=concrete_data).fit()

#create residual vs. predictor plot for 'assists'
for i in X.columns:
    fig = plt.figure(figsize=(12,8))
    fig = sm.graphics.plot_regress_exog(model, i, fig=fig)
    fig.show()
```

[26] ✓ 6.8s (parameter) figsize: tuple[float, float] | None Python




File Edit Selection View Go Run Terminal Help
Regression Plots for slag
Y and Fitted vs. X
Residuals versus slag
Partial regression plot
CCPR Plot
Regression Plots for flyash
Y and Fitted vs. X
Residuals versus flyash
Partial regression plot
CCPR Plot
33°C
Haze
Search
ENG
IN
16:28
25-10-2024
File Edit Selection View Go Run Terminal Help
Regression Plots for flyash
Y and Fitted vs. X
Residuals versus flyash
Partial regression plot
CCPR Plot
Regression Plots for water
Y and Fitted vs. X
Residuals versus water
Partial regression plot
CCPR Plot
33°C
Haze
Search
ENG
IN
16:28
25-10-2024


File Edit Selection View Go Run Terminal Help
Regression Plots for superplasticizer
Y and Fitted vs. X
Residuals versus superplasticizer
Partial regression plot
CCPR Plot
Regression Plots for coarseaggregate
Y and Fitted vs. X
Residuals versus coarseaggregate
Partial regression plot
CCPR Plot
33°C
Haze
Search
ENG
IN
16:28
25-10-2024

File Edit Selection View Go Run Terminal Help
Regression Plots for fineaggregate
Y and Fitted vs. X
Residuals versus fineaggregate
Partial regression plot
CCPR Plot
Regression Plots for age
Y and Fitted vs. X
Residuals versus age
Partial regression plot
CCPR Plot
33°C
Haze
Search
ENG
IN
16:28
25-10-2024

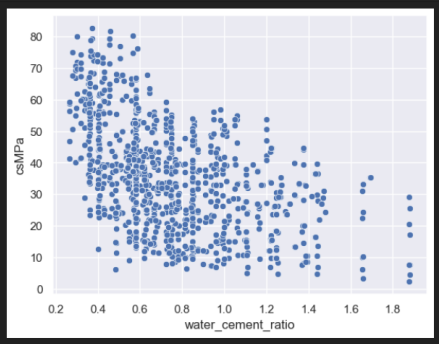
6. Feature Engineering with OLS

```
#feature engineering using knowledge that water:cement ratio is an important factor for concrete strength
X['water_cement_ratio'] = X['water']/X['cement']
```

[27] ✓ 0.0s Python

```
#plot water:cement ratio against compressive strength
sns.scatterplot(x=X['water_cement_ratio'], y=y)
plt.show()
```

[28] ✓ 0.1s Python




```
#generate OLS regression results with water : cement ratio
X_sm = sm.add_constant(X)
model = sm.OLS(y,X_sm)
print(model.fit().summary())
```

[29] ✓ 0.0s Python

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  csMPa   R-squared:                       0.618
Model:                            OLS   Adj. R-squared:                  0.615
Method:                 Least Squares   F-statistic:                     183.6
Date:                Fri, 25 Oct 2024   Prob (F-statistic):          2.28e-206
Time:                        16:10:17   Log-Likelihood:                -3865.2
No. Observations:                1030   AIC:                             7750.
Df Residuals:                    1020   BIC:                             7800.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -16.1769     26.631     -0.607      0.544     -68.434      36.080
cement         0.1013      0.011      9.346      0.000       0.080       0.123
slag           0.1062      0.010     10.473      0.000       0.086       0.126
flyash         0.0881      0.013      7.023      0.000       0.063       0.113
water         -0.1226      0.041     -2.971      0.003      -0.204      -0.042
superplasticizer  0.2893      0.093      3.106      0.002       0.107       0.472
coarseaggregate  0.0164      0.009      1.750      0.081      -0.002       0.035
fineaggregate  0.0204      0.011      1.908      0.057      -0.001       0.041
age            0.1132      0.005     20.884      0.000       0.103       0.124
water_cement_ratio    -7.3785      2.699     -2.734      0.006     -12.675      -2.082
==============================================================================
...
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.07e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
#generate OLS summary with only water : cement ratio
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.07e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
#generate OLS summary with only water : cement ratio
X_sm = sm.add_constant(X['water_cement_ratio'])
model = sm.OLS(y,X_sm)
print(model.fit().summary())
```

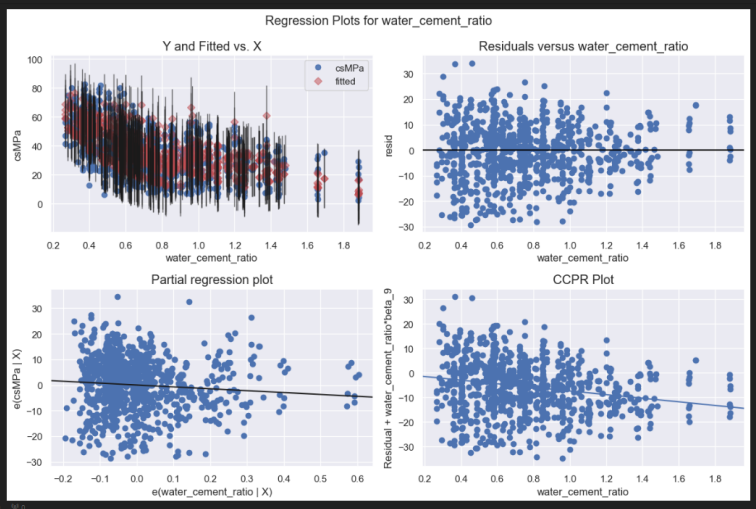
0.0s Python

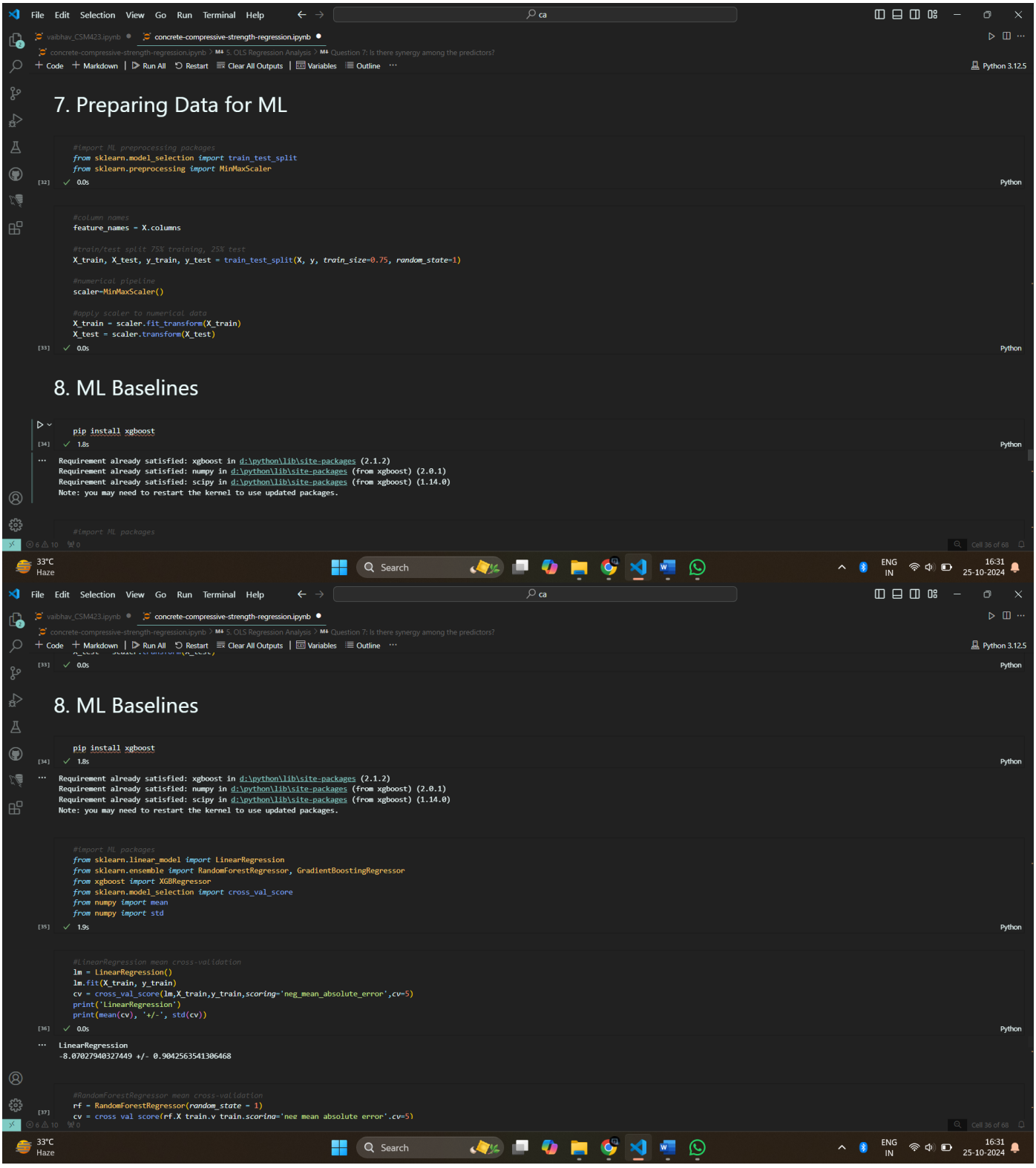
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  csMPa   R-squared:                       0.251
Model:                            OLS   Adj. R-squared:                  0.250
Method:                 Least Squares   F-statistic:                     343.9
Date:                Fri, 25 Oct 2024   Prob (F-statistic):           1.86e-66
Time:                        16:10:17   Log-Likelihood:                -4212.6
No. Observations:                1030   AIC:                             8429.
Df Residuals:                    1028   BIC:                             8439.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         55.7502      1.165     47.834      0.000      53.463      58.037
water_cement_ratio   -26.6379      1.436    -18.545      0.000     -29.456     -23.819
==============================================================================
Omnibus:                       29.143   Durbin-Watson:                   1.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.972
Skew:                           0.176   Prob(JB):                     0.000125
Kurtosis:                       2.456   Cond. No.                         5.09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Looking at the p-value of the t-statistic: water : cement ratio has a strong association with compressive strength.

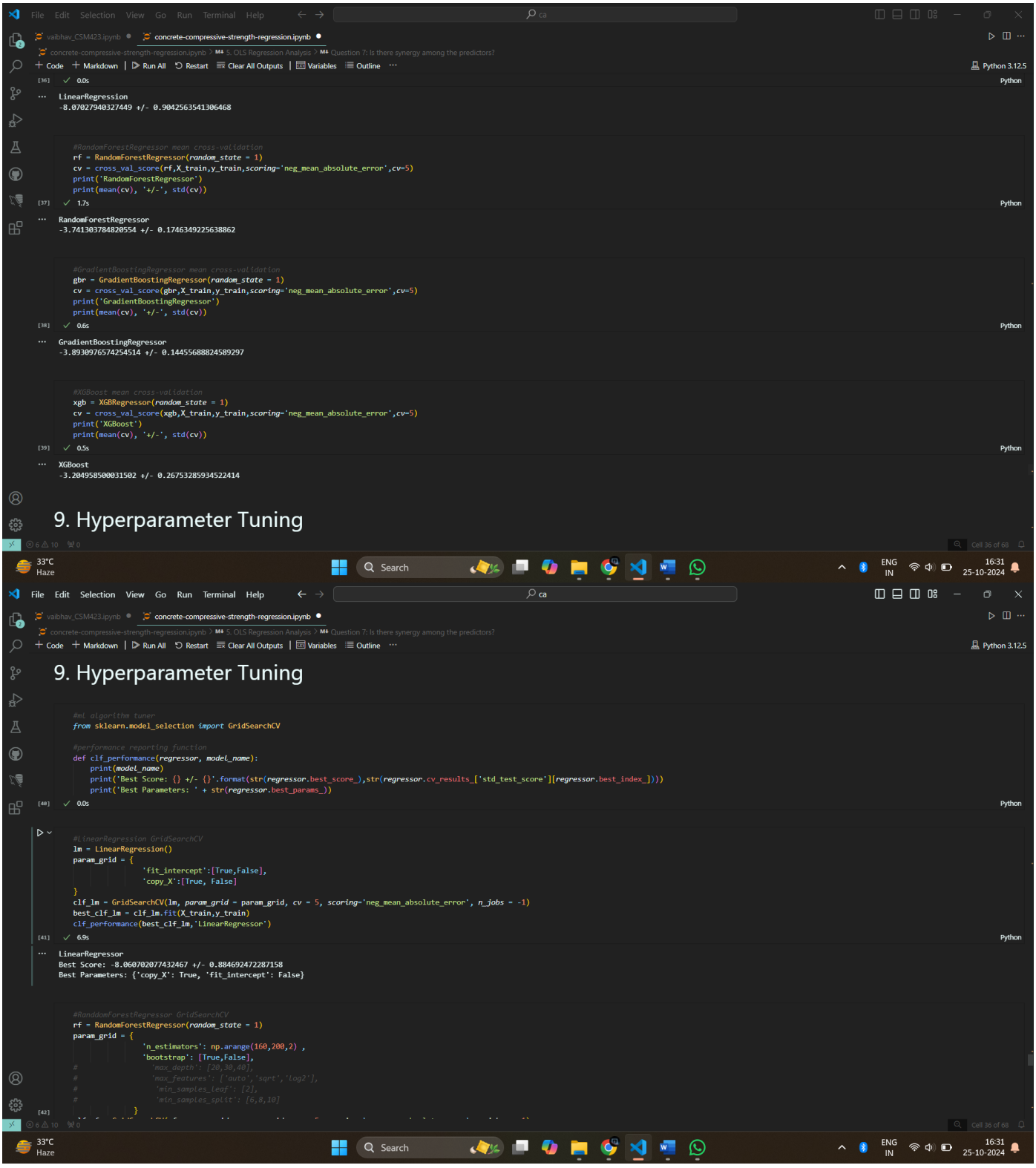
```
#copy water : cement ratio into concrete data
concrete_data['water_cement_ratio'] = X['water_cement_ratio']

#fit multiple linear regression model with water : cement ratio
model = ols('csMPa ~ cement + slag + flyash + water + superplasticizer + coarseaggregate + fineaggregate + age + water_cement_ratio', data=concrete_data).fit()

#plot regression plots for water : cement ratio
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'water_cement_ratio', fig=fig)
fig.show()
```

1.2s Python

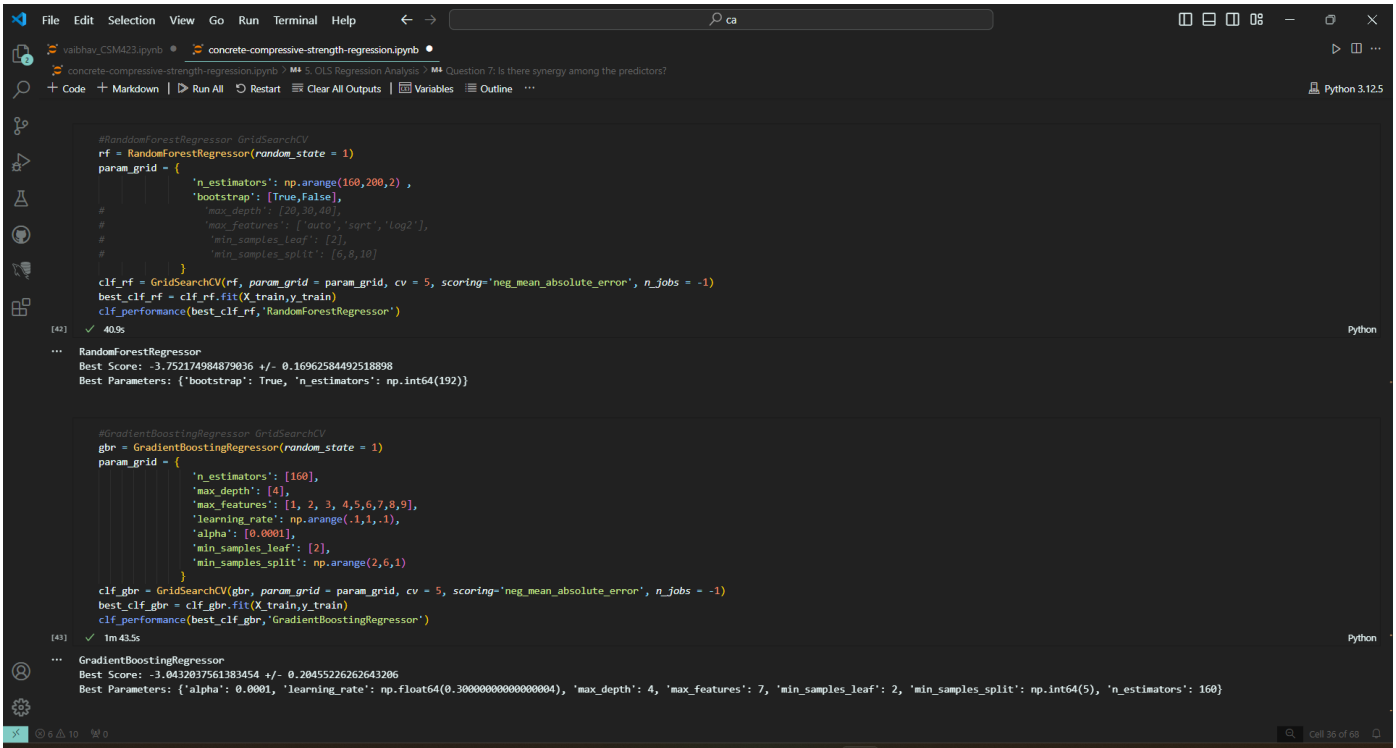


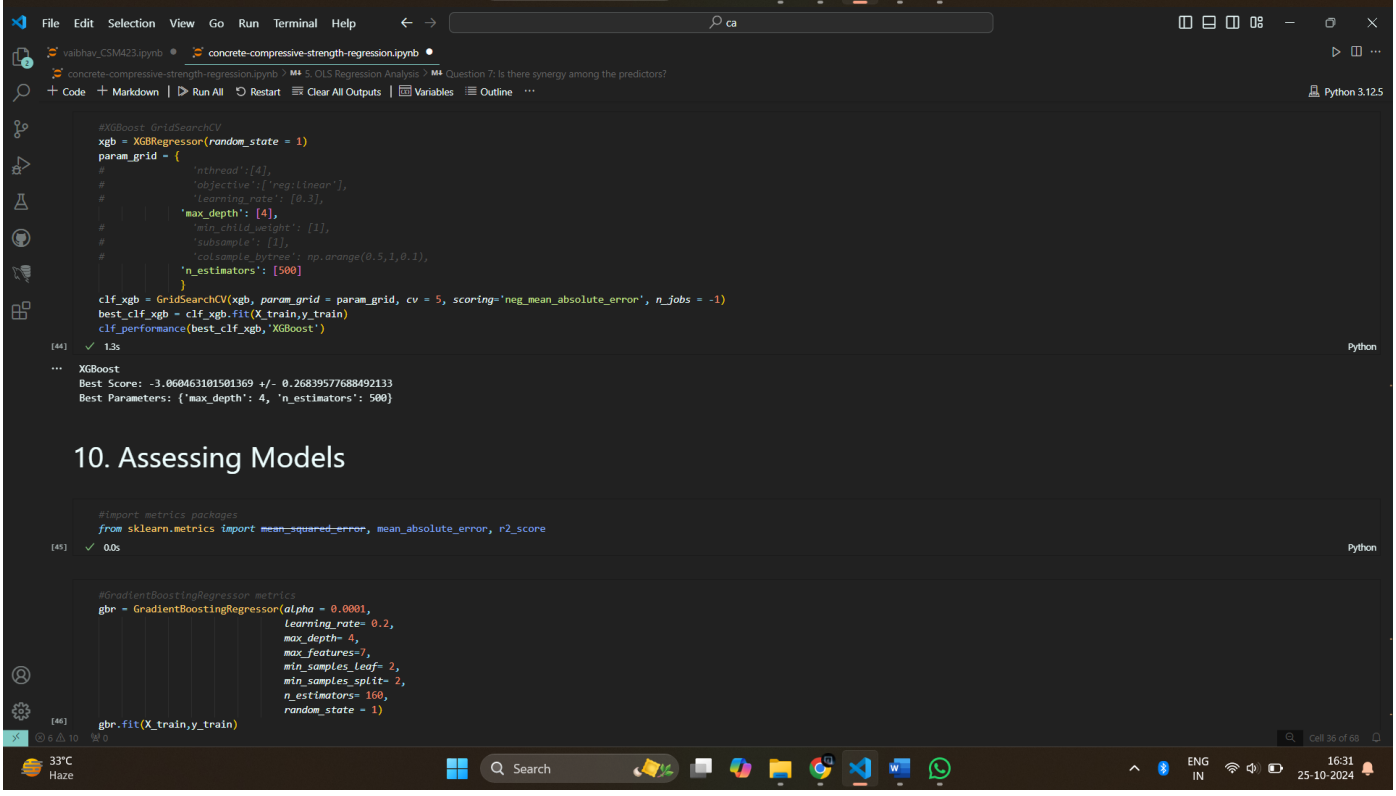

File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
7. Preparing Data for ML
#import ML preprocessing packages
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
[32] 0.0s
Python
#column names
feature_names = X.columns
#train/test split 75% training, 25% test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=1)
#numerical pipeline
scaler=MinMaxScaler()
#apply scaler to numerical data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
[33] 0.0s
Python
8. ML Baselines
pip install xgboost
[34] 1.8s
Python
Requirement already satisfied: xgboost in d:\python\lib\site-packages (2.1.2)
Requirement already satisfied: numpy in d:\python\lib\site-packages (from xgboost) (2.0.1)
Requirement already satisfied: scipy in d:\python\lib\site-packages (from xgboost) (1.14.0)
Note: you may need to restart the kernel to use updated packages.
#import ML packages
33°C Haze
Search
ENG IN
16:31 25-10-2024

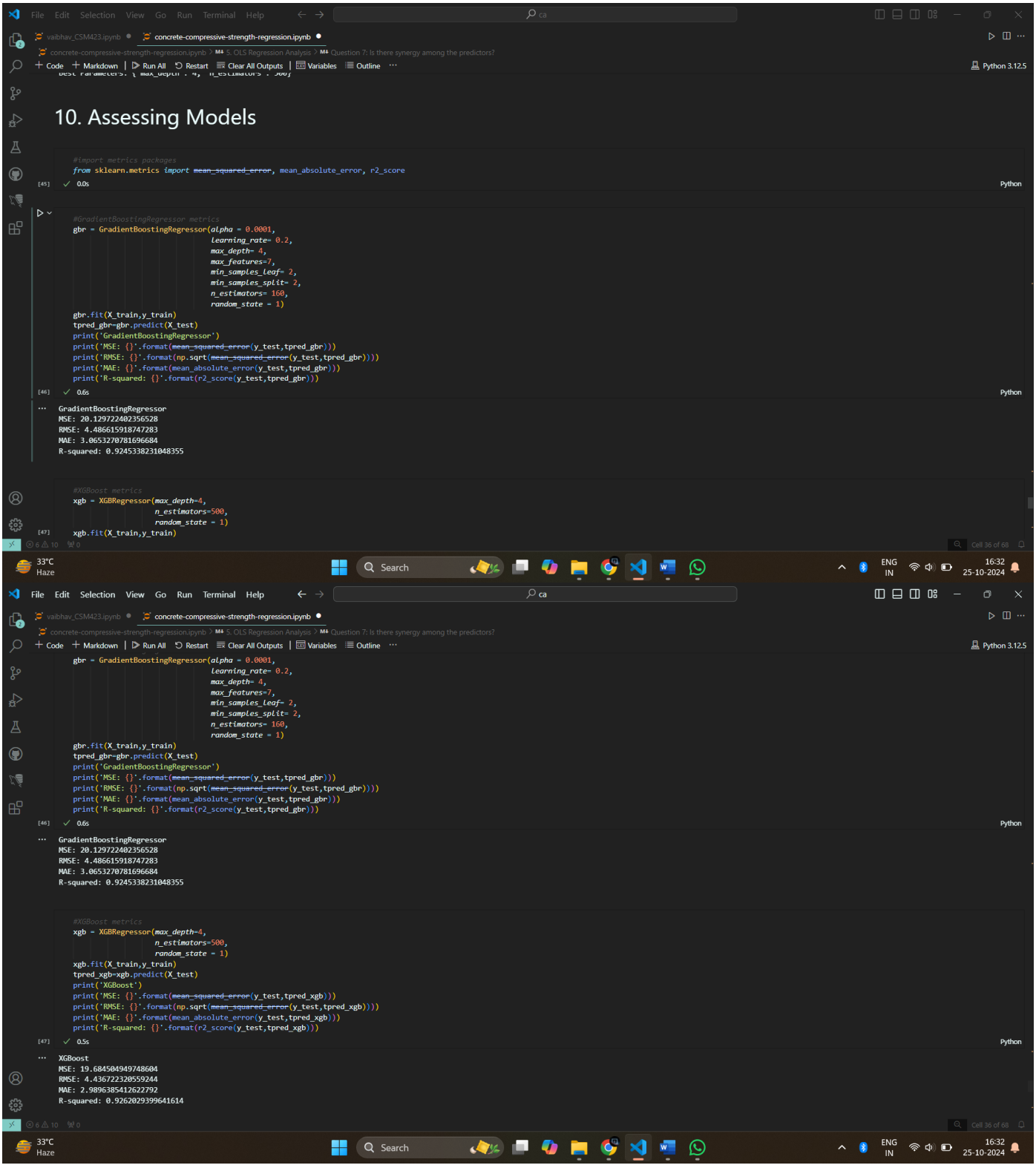
File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
[33] 0.0s
Python
8. ML Baselines
pip install xgboost
[34] 1.8s
Python
Requirement already satisfied: xgboost in d:\python\lib\site-packages (2.1.2)
Requirement already satisfied: numpy in d:\python\lib\site-packages (from xgboost) (2.0.1)
Requirement already satisfied: scipy in d:\python\lib\site-packages (from xgboost) (1.14.0)
Note: you may need to restart the kernel to use updated packages.
#import ML packages
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
[35] 1.9s
Python
#LinearRegression mean cross-validation
lm = LinearRegression()
lm.fit(X_train, y_train)
cv = cross_val_score(lm,X_train,y_train,scoring='neg_mean_absolute_error',cv=5)
print('LinearRegression')
print(mean(cv), '+/-', std(cv))
[36] 0.0s
Python
LinearRegression
-8.07027940327449 +/- 0.9042563541306468
#RandomForestRegressor mean cross-validation
rf = RandomForestRegressor(random_state = 1)
cv = cross_val_score(rf,X_train,y_train,scoring='neg_mean_absolute_error',cv=5)
33°C Haze
Search
ENG IN
16:31 25-10-2024


File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
LinearRegression
-8.07027940327449 +/- 0.9042563541306468
#RandomForestRegressor mean cross-validation
rf = RandomForestRegressor(random_state = 1)
cv = cross_val_score(rf,X_train,y_train,scoring='neg_mean_absolute_error',cv=5)
print('RandomForestRegressor')
print(mean(cv), '+/-', std(cv))
RandomForestRegressor
-3.741303784820554 +/- 0.1746349225638862
#GradientBoostingRegressor mean cross-validation
gbr = GradientBoostingRegressor(random_state = 1)
cv = cross_val_score(gbr,X_train,y_train,scoring='neg_mean_absolute_error',cv=5)
print('GradientBoostingRegressor')
print(mean(cv), '+/-', std(cv))
GradientBoostingRegressor
-3.893097657425451 4 +/- 0.14455688824589297
#XGBoost mean cross-validation
xgb = XGBRegressor(random_state = 1)
cv = cross_val_score(xgb,X_train,y_train,scoring='neg_mean_absolute_error',cv=5)
print('XGBoost')
print(mean(cv), '+/-', std(cv))
XGBoost
-3.204958500031502 +/- 0.26753285934522414
9. Hyperparameter Tuning
33°C Haze
Search
ENG IN
16:31 25-10-2024

File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
9. Hyperparameter Tuning
#ml algorithm tuner
from sklearn.model_selection import GridSearchCV
#performance reporting function
def clf_performance(regressor, model_name):
print(model_name)
print('Best Score: {} +/- {}'.format(str(regressor.best_score_),str(regressor.cv_results_['std_test_score'][regressor.best_index_])))
print('Best Parameters: ' + str(regressor.best_params_))
#LinearRegression GridSearchCV
lm = LinearRegression()
param_grid = {
'fit_intercept':[True,False],
'copy_X':[True, False]
}
clf_lm = GridSearchCV(lm, param_grid = param_grid, cv = 5, scoring='neg_mean_absolute_error', n_jobs = -1)
best_clf_lm = clf_lm.fit(X_train,y_train)
clf_performance(best_clf_lm,'LinearRegressor')
LinearRegressor
Best Score: -8.060702077432467 +/- 0.884692472287158
Best Parameters: {'copy_X': True, 'fit_intercept': False}
#RanddomForestRegressor GridSearchCV
rf = RandomForestRegressor(random_state = 1)
param_grid = {
'n_estimators': np.arange(160,200,2) ,
'bootstrap': [True,False],
# 'max_depth': [20,30,40],
# 'max_features': ['auto','sqrt','log2'],
# 'min_samples_leaf': [2],
# 'min_samples_split': [6,8,10]
}
33°C Haze
Search
ENG IN
16:31 25-10-2024


File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
#RanddomForestRegressor GridSearchCV
rf = RandomForestRegressor(random_state = 1)
param_grid = {
'n_estimators': np.arange(160,200,2) ,
'bootstrap': [True,False],
# 'max_depth': [20,30,40],
# 'max_features': ['auto','sqrt','log2'],
# 'min_samples_leaf': [2],
# 'min_samples_split': [6,8,10]
}
clf_rf = GridSearchCV(rf, param_grid = param_grid, cv = 5, scoring='neg_mean_absolute_error', n_jobs = -1)
best_clf_rf = clf_rf.fit(X_train,y_train)
clf_performance(best_clf_rf,'RandomForestRegressor')
[42] 40.9s
Python
RandomForestRegressor
Best Score: -3.752174984879036 +/- 0.16962584492518898
Best Parameters: {'bootstrap': True, 'n_estimators': np.int64(192)}
#GradientBoostingRegressor GridSearchCV
gbr = GradientBoostingRegressor(random_state = 1)
param_grid = {
'n_estimators': [160],
'max_depth': [4],
'max_features': [1, 2, 3, 4,5,6,7,8,9],
'learning_rate': np.arange(.1,1,.1),
'alpha': [0.0001],
'min_samples_leaf': [2],
'min_samples_split': np.arange(2,6,1)
}
clf_gbr = GridSearchCV(gbr, param_grid = param_grid, cv = 5, scoring='neg_mean_absolute_error', n_jobs = -1)
best_clf_gbr = clf_gbr.fit(X_train,y_train)
clf_performance(best_clf_gbr,'GradientBoostingRegressor')
[43] 1m 43.5s
Python
GradientBoostingRegressor
Best Score: -3.0432037561383454 +/- 0.20455226262643206
Best Parameters: {'alpha': 0.0001, 'learning_rate': np.float64(0.3000000000000004), 'max_depth': 4, 'max_features': 7, 'min_samples_leaf': 2, 'min_samples_split': np.int64(5), 'n_estimators': 160}
33°C Haze
Search
ENG IN
16:31 25-10-2024


File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
#XGBoost GridSearchCV
xgb = XGBRegressor(random_state = 1)
param_grid = {
# 'nthread':[4],
# 'objective':['reg:linear'],
# 'learning_rate': [0.3],
'max_depth': [4],
'min_child_weight': [1],
'subsample': [1],
'colsample_bytree': np.arange(0.5,1,0.1),
'n_estimators': [500]
}
clf_xgb = GridSearchCV(xgb, param_grid = param_grid, cv = 5, scoring='neg_mean_absolute_error', n_jobs = -1)
best_clf_xgb = clf_xgb.fit(X_train,y_train)
clf_performance(best_clf_xgb,'XGBoost')
[44] 1.3s
Python
XGBoost
Best Score: -3.060463101501369 +/- 0.26839577688492133
Best Parameters: {'max_depth': 4, 'n_estimators': 500}
10. Assessing Models
#import metrics packages
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
[45] 0.0s
Python
#GradientBoostingRegressor metrics
gbr = GradientBoostingRegressor(alpha = 0.0001,
learning_rate= 0.2,
max_depth= 4,
max_features=7,
min_samples_leaf= 2,
min_samples_split= 2,
n_estimators= 160,
random_state = 1)
[46] gbr.fit(X_train,y_train)
33°C Haze
Search
ENG IN
16:31 25-10-2024


File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
10. Assessing Models
#import metrics packages
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
[45] 0.0s
Python
#GradientBoostingRegressor metrics
gbr = GradientBoostingRegressor(alpha = 0.0001,
learning_rate= 0.2,
max_depth= 4,
max_features=7,
min_samples_leaf= 2,
min_samples_split= 2,
n_estimators= 160,
random_state = 1)
gbr.fit(X_train,y_train)
tpred_gbr=gbr.predict(X_test)
print('GradientBoostingRegressor')
print('MSE: {}'.format(mean_squared_error(y_test,tpred_gbr)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(y_test,tpred_gbr))))
print('MAE: {}'.format(mean_absolute_error(y_test,tpred_gbr)))
print('R-squared: {}'.format(r2_score(y_test,tpred_gbr)))
[46] 0.6s
Python
GradientBoostingRegressor
MSE: 20.129722402356528
RMSE: 4.486615918747283
MAE: 3.0653270781696684
R-squared: 0.9245338231048355
#XGBoost metrics
xgb = XGBRegressor(max_depth=4,
n_estimators=500,
random_state = 1)
[47] xgb.fit(X_train,y_train)
33°C Haze
Search
ENG IN
16:32 25-10-2024
File Edit Selection View Go Run Terminal Help
vaibhav_CSM423.ipynb
concrete-compressive-strength-regression.ipynb
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.12.5
gbr = GradientBoostingRegressor(alpha = 0.0001,
learning_rate= 0.2,
max_depth= 4,
max_features=7,
min_samples_leaf= 2,
min_samples_split= 2,
n_estimators= 160,
random_state = 1)
gbr.fit(X_train,y_train)
tpred_gbr=gbr.predict(X_test)
print('GradientBoostingRegressor')
print('MSE: {}'.format(mean_squared_error(y_test,tpred_gbr)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(y_test,tpred_gbr))))
print('MAE: {}'.format(mean_absolute_error(y_test,tpred_gbr)))
print('R-squared: {}'.format(r2_score(y_test,tpred_gbr)))
[46] 0.6s
Python
GradientBoostingRegressor
MSE: 20.129722402356528
RMSE: 4.486615918747283
MAE: 3.0653270781696684
R-squared: 0.9245338231048355
#XGBoost metrics
xgb = XGBRegressor(max_depth=4,
n_estimators=500,
random_state = 1)
xgb.fit(X_train,y_train)
tpred_xgb=xgb.predict(X_test)
print('XGBoost')
print('MSE: {}'.format(mean_squared_error(y_test,tpred_xgb)))
print('RMSE: {}'.format(np.sqrt(mean_squared_error(y_test,tpred_xgb))))
print('MAE: {}'.format(mean_absolute_error(y_test,tpred_xgb)))
print('R-squared: {}'.format(r2_score(y_test,tpred_xgb)))
[47] 0.5s
Python
XGBoost
MSE: 19.684504949748604
RMSE: 4.436722320559244
MAE: 2.9896385412622792
R-squared: 0.9262029399641614
33°C Haze
Search
ENG IN
16:32 25-10-2024

CONCLUSION

This project successfully developed a machine learning-based approach for predicting concrete compressive strength. By employing models like Linear Regression, Ridge Regression, Lasso, and Random Forest, we could evaluate how different components of concrete mixtures, such as cement, aggregates, water, and admixtures, influence its strength.

Through detailed data preprocessing, including handling missing values and scaling features, and rigorous model training with hyperparameter tuning (GridSearchCV), we identified Random Forest as the most effective model for capturing the complex interactions within the dataset. The high accuracy of this model highlights its suitability for practical applications in concrete design, providing a data-driven method for optimizing material usage and improving construction quality.

This project not only demonstrates the potential of machine learning in automating material strength predictions but also opens pathways for further research. Future work could explore integrating environmental factors or advanced materials to further enhance prediction accuracy and practical applicability.

BIBLIOGRAPHY

1. Dataset: https://www.kaggle.com/code/michaelbryantds/concrete-compressive-strength-regression/input
2. Upgrad ML Course
3. GeekForGeeks
4. Github Link: https://github.com/kohli2113/CSM423-Machine-Learning