- Name: Deep Salunkhe
- Roll No.:21102A0014
- SEM-7 ML Lab5 Github Link

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, mean_squared_error, r2_score
from sklearn.metrics import confusion_matrix, classification_report

# Load datasets
attributes_array = [
    'fixed acidity', 'volatile acidity', 'citric acid',
    'residual sugar', 'chlorides', 'free sulfur dioxide',
    'total sulfur dioxide', 'density', 'pH', 'sulphates',
    'alcohol', 'quality'
]

df_red = pd.read_csv('/content/winequality-red.csv', delimiter=";")
df_white = pd.read_csv('/content/winequality-white.csv',
delimiter=";")

df = pd.concat([df_red, df_white])

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6497,\n  \"fields\":
[\n    {\n      \"column\": \"fixed acidity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.2964337577998153,\n        \"min\": 3.8,\n        \"max\": 15.9,\n
\"num_unique_values\": 106,\n        \"samples\": [\n          7.15,\n
8.1,\n          7.3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"volatile acidity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.16463647408467877,\n        \"min\":
0.08,\n        \"max\": 1.58,\n        \"num_unique_values\": 187,\n
\"samples\": [\n          0.405,\n          0.21,\n          0.695\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"citric acid\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.14531786489759155,\n        \"min\": 0.0,\n        \"max\": 1.66,\n
\"num_unique_values\": 89,\n        \"samples\": [\n          0.1,\n
0.6,\n          0.37\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"residual sugar\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.757803743147418,\n        \"min\":
0.6,\n        \"max\": 65.8,\n        \"num_unique_values\": 316,\n
\"samples\": [\n          18.95,\n          3.2,\n          9.3\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"chlorides\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.03503360137245907,\n        \"min\": 0.009,\n        \"max\":
0.611,\n        \"num_unique_values\": 214,\n        \"samples\": [\n
0.089,\n          0.217,\n          0.1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"free sulfur dioxide\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
17.7493997720025,\n        \"min\": 1.0,\n        \"max\": 289.0,\n
\"num_unique_values\": 135,\n        \"samples\": [\n          77.5,\n
65.0,\n          128.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total sulfur dioxide\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 56.521854522630285,\n        \"min\":
6.0,\n        \"max\": 440.0,\n        \"num_unique_values\": 276,\n
\"samples\": [\n          14.0,\n          149.0,\n          227.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"density\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0029986730037190393,\n        \"min\": 0.98711,\n        \"max\":
1.03898,\n        \"num_unique_values\": 998,\n        \"samples\": [\
n          0.9918,\n          0.99412,\n          0.99484\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"pH\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.16078720210398764,\n
\"min\": 2.72,\n        \"max\": 4.01,\n        \"num_unique_values\":
108,\n        \"samples\": [\n          3.74,\n          3.17,\n
3.3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sulphates\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.14880587361449027,\n        \"min\":
0.22,\n        \"max\": 2.0,\n        \"num_unique_values\": 111,\n
\"samples\": [\n          1.11,\n          1.56,\n          0.46\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"alcohol\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.192711748870993,\n        \"min\": 8.0,\n        \"max\": 14.9,\n
\"num_unique_values\": 111,\n        \"samples\": [\n
10.9333333333333,\n          9.7,\n          10.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"quality\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 3,\n        \"max\": 9,\n        \"num_unique_values\": 7,\n
```

```
\"samples\": [\n          5,\n          6,\n          3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
df['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3, 9])
```

```
df.isnull().sum()
```

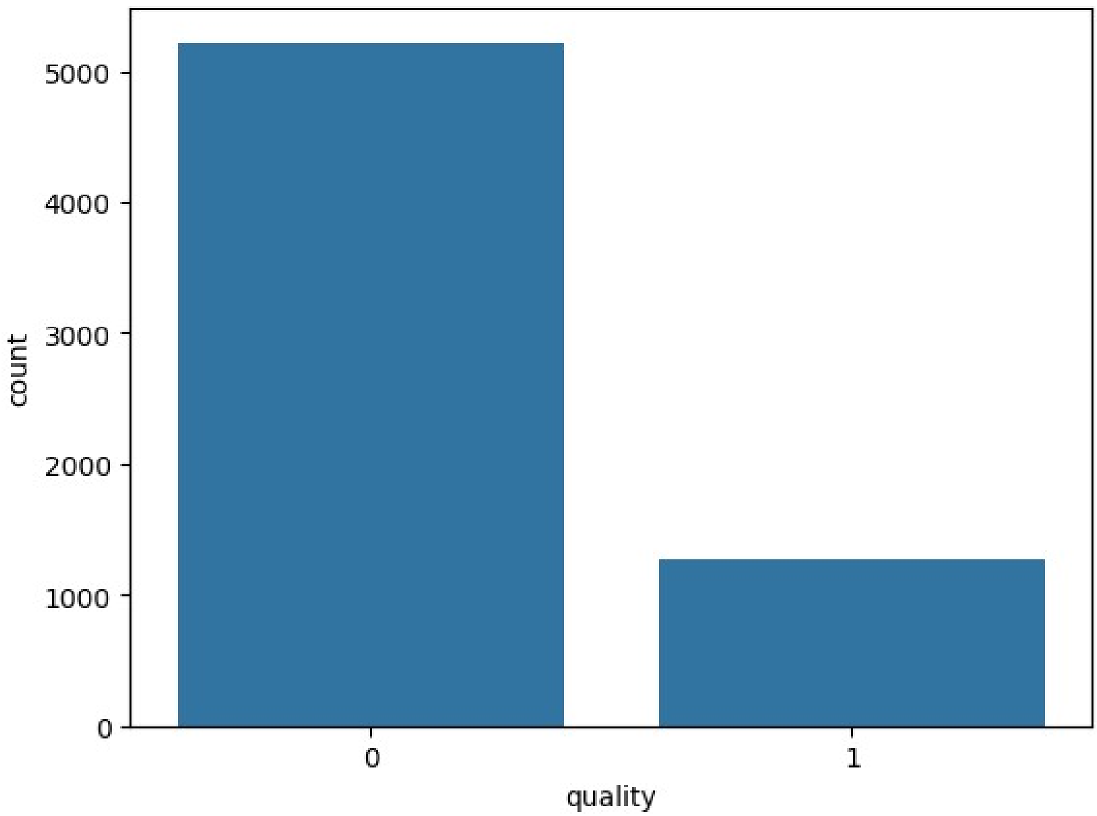
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
bins = (2, 6.5, 9)
group_names = ['bad', 'good']
df['quality'] = pd.cut(df['quality'], bins = bins, labels =
group_names)
df['quality'].unique()
```

```
['bad', 'good']
Categories (2, object): ['bad' < 'good']
```

```
label_quality = LabelEncoder()
df['quality'] = label_quality.fit_transform(df['quality'])
```

```
sns.countplot(x='quality', data=df)
plt.show()
```




```
#Now separate the dataset as response variable and feature variables
X = df.drop('quality', axis=1)
y = df['quality']

#Train and test splitting of data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 42)

#Applying Standard scaling to get optimized result
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Create an SVM model
svm_model = SVC(kernel='linear', C=1)

# Train the model
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average = 'weighted',
```

```
zero_division = 1)
recall = recall_score(y_test, y_pred, average = 'weighted')
f1 = f1_score(y_test, y_pred, average = 'weighted')

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

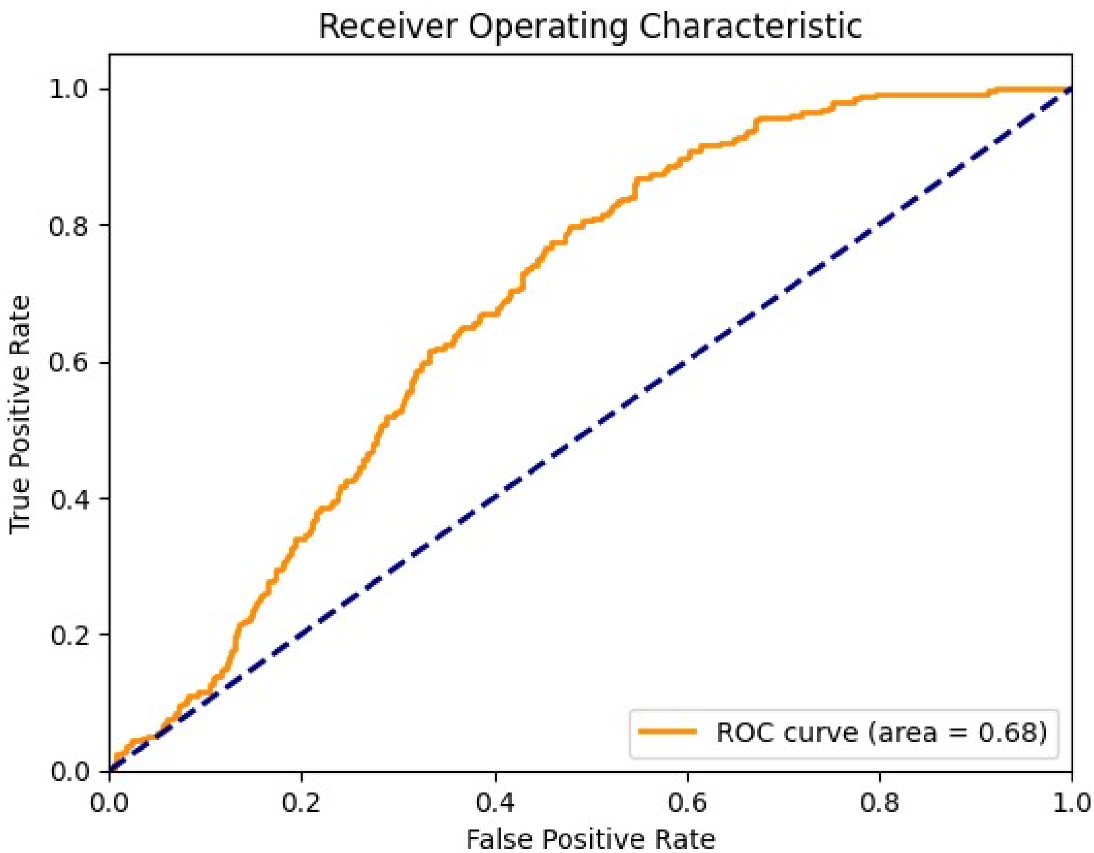
```
Accuracy: 0.8061538461538461
Precision: 0.843730177514793
Recall: 0.8061538461538461
F1-score: 0.7196330756126327
```

```
print(classification_report(y_test, y_pred, zero_division=1))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89      1048
           1       1.00      0.00      0.00       252

    accuracy                           0.81      1300
   macro avg       0.90      0.50      0.45      1300
weighted avg       0.84      0.81      0.72      1300

[[1048    0]
 [ 252    0]]
```

```
from sklearn.preprocessing import label_binarize

# Binarize the output labels for ROC curve calculation
y_test_bin = label_binarize(y_test, classes=[0, 1])
y_pred_prob = svm_model.decision_function(X_test)

# Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr, tpr, _ = roc_curve(y_test_bin, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

print("AUC:", roc_auc)
```




```
AUC: 0.684326911426148
```

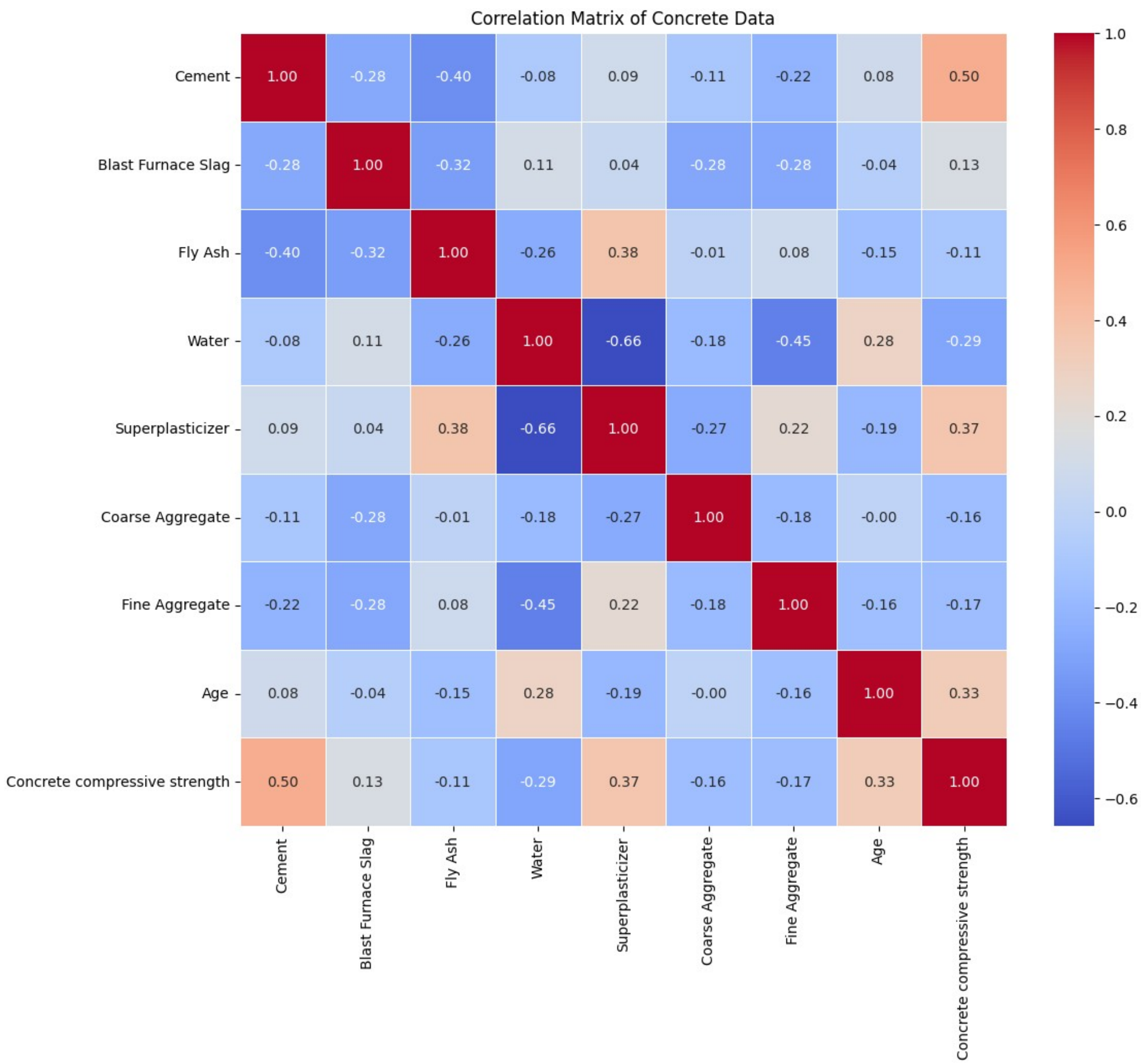
Regression

```
# Load the dataset
df_concrete = pd.read_excel('/content/Concrete_Data.xls')


attributes_array = [
    'Cement',
    'Blast Furnace Slag',
    'Fly Ash',
    'Water',
```

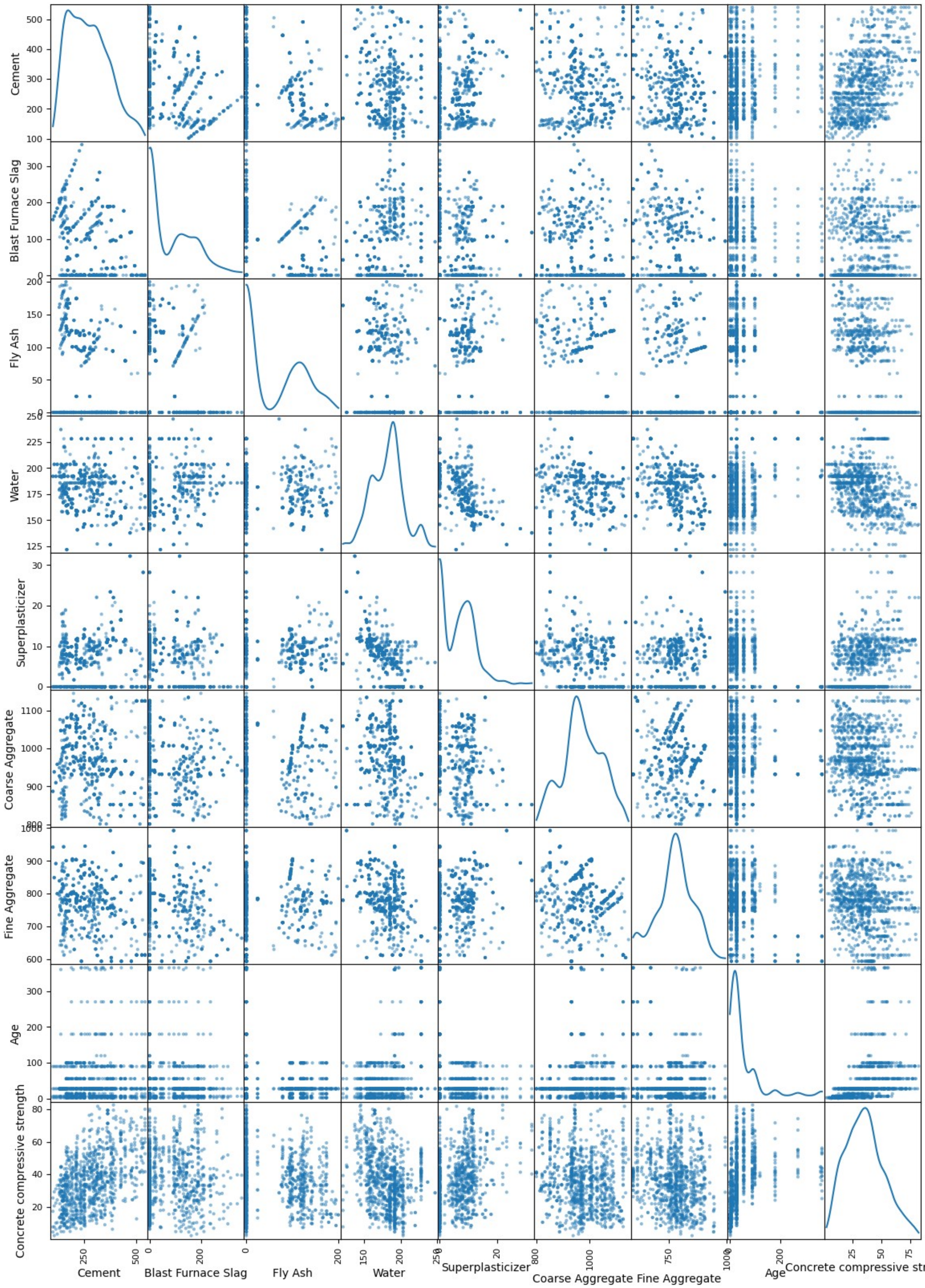
```
    'Superplasticizer',
    'Coarse Aggregate',
    'Fine Aggregate',
    'Age',
    'Concrete compressive strength'
]

df_concrete.columns = attributes_array
df_concrete.to_csv('concrete_data.csv', index=False)
df = pd.read_csv('concrete_data.csv')
df.head()
```

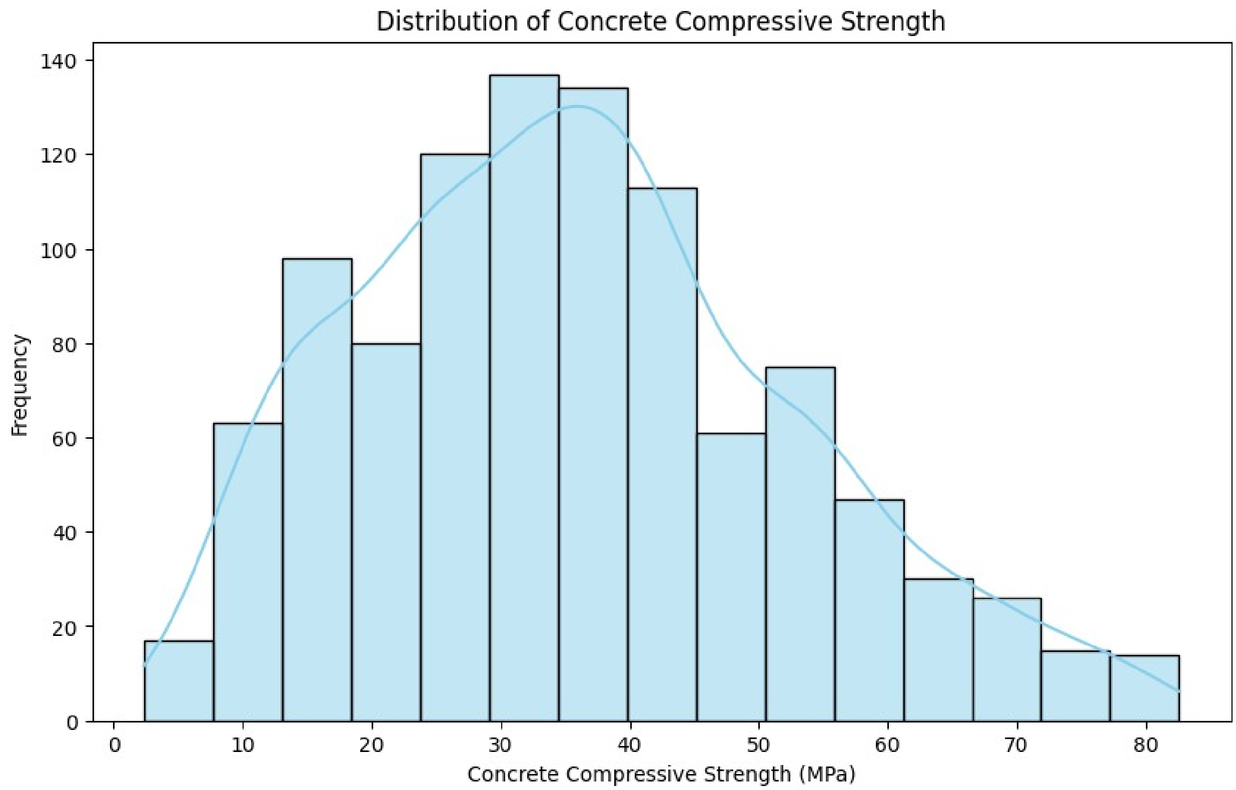

{"summary":"{\n \"name\": \"df\",\n \"rows\": 1030,\n \"fields\":
[\n   {\n   \"column\": \"Cement\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 104.5071416428718,\n
\"min\": 102.0,\n   \"max\": 540.0,\n
\"num_unique_values\": 280,\n   \"samples\": [\n
194.68,\n   480.0,\n   145.4\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Blast Furnace Slag\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
86.27910364316895,\n   \"min\": 0.0,\n   \"max\": 359.4,\n
\"num_unique_values\": 187,\n   \"samples\": [\n   186.7,\
n   212.0,\n   26.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Fly Ash\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
63.99646938186508,\n   \"min\": 0.0,\n   \"max\": 200.1,\n
\"num_unique_values\": 163,\n   \"samples\": [\n   81.8,\n
137.9,\n   107.5\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Water\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 21.355567066911522,\n   \"min\":
121.75,\n   \"max\": 247.0,\n   \"num_unique_values\": 205,\
n   \"samples\": [\n   164.9,\n   181.1,\n
185.7\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Superplasticizer\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 5.973491650590111,\n   \"min\":
0.0,\n   \"max\": 32.2,\n   \"num_unique_values\": 155,\n
\"samples\": [\n   4.14,\n   9.8,\n   6.13\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Coarse Aggregate\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
77.75381809178927,\n   \"min\": 801.0,\n   \"max\": 1145.0,\
n   \"num_unique_values\": 284,\n   \"samples\": [\n
852.1,\n   913.9,\n   914.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Fine Aggregate\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":

```
80.1754273990239,\n        \"min\": 594.0,\n        \"max\": 992.6,\n
\"num_unique_values\": 304,\n        \"samples\": [\n          698.0,\
n          613.0,\n          689.3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 63,\n        \"min\": 1,\n
\"max\": 365,\n        \"num_unique_values\": 14,\n
\"samples\": [\n          91,\n          100,\n          28\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Concrete compressive strength\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 16.705679174867946,\n
\"min\": 2.331807832,\n        \"max\": 82.5992248,\n
\"num_unique_values\": 938,\n        \"samples\": [\n
33.39821744,\n          56.63355864,\n          25.559564796\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}

# Increase the figure size for better readability
plt.figure(figsize=(12, 10))

# Create the heatmap with annotations and a title
sns.heatmap(df.corr(), annot=True, linewidths=.5, fmt=".2f",
cmap="coolwarm")
plt.title("Correlation Matrix of Concrete Data")
plt.show()

print()

from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize=(14, 20), diagonal='kde')
plt.show()

print()

# Create a distribution plot with increased figure size and a
descriptive title
plt.figure(figsize=(10, 6))
sns.histplot(df['Concrete compressive strength'], bins=15, kde=True,
color='skyblue')
plt.xlabel("Concrete Compressive Strength (MPa)")
plt.ylabel("Frequency")
plt.title('Distribution of Concrete Compressive Strength')
plt.show()
```



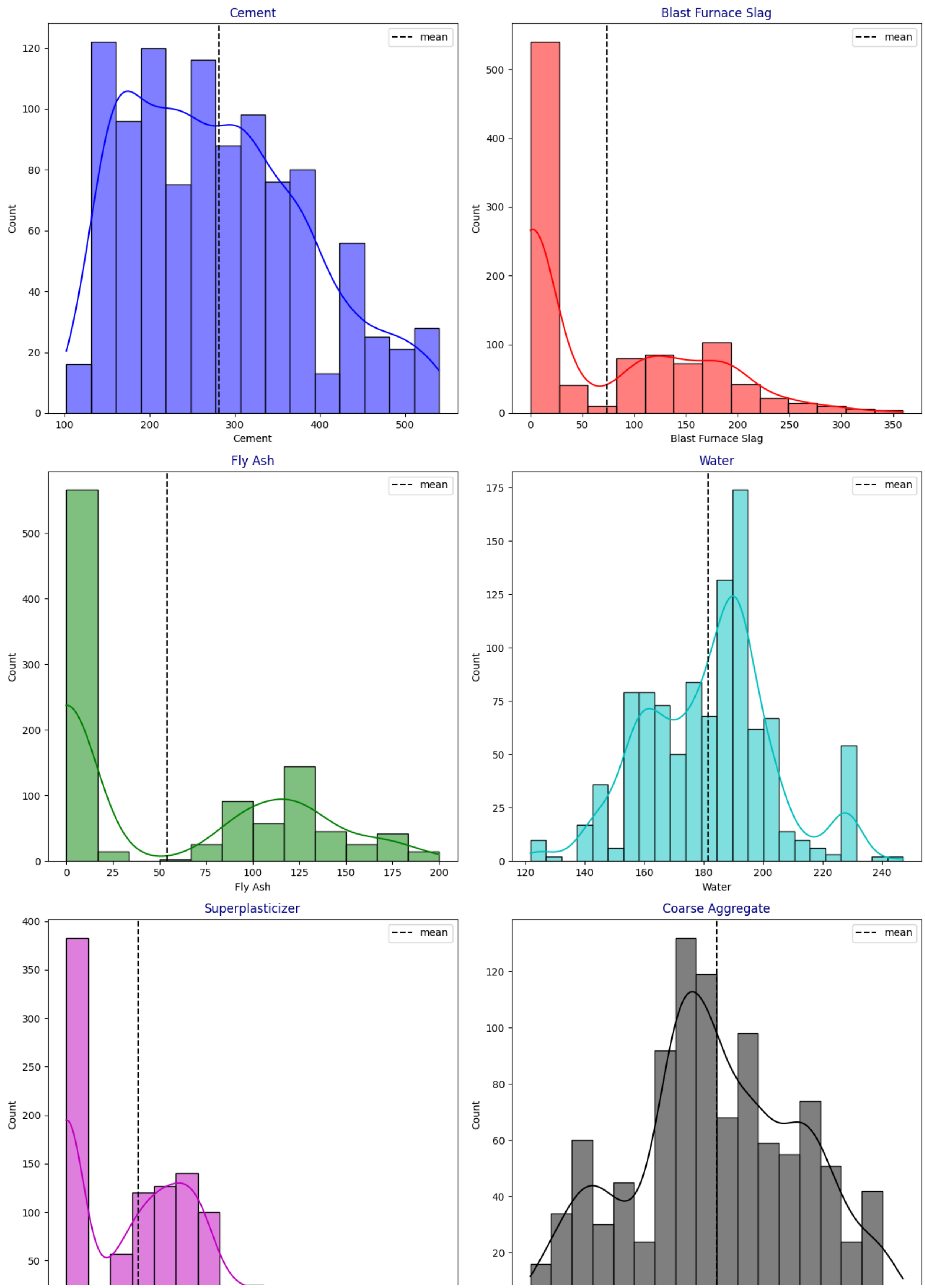
Correlation Matrix of Concrete Data
Cement
Blast Furnace Slag
Fly Ash
Water
Superplasticizer
Coarse Aggregate
Fine Aggregate
Age
Concrete compressive strength
1.00 -0.28 -0.40 -0.08 0.09 -0.11 -0.22 0.08 0.50
-0.28 1.00 -0.32 0.11 0.04 -0.28 -0.28 -0.04 0.13
-0.40 -0.32 1.00 -0.26 0.38 -0.01 0.08 -0.15 -0.11
-0.08 0.11 -0.26 1.00 -0.66 -0.18 -0.45 0.28 -0.29
0.09 0.04 0.38 -0.66 1.00 -0.27 0.22 -0.19 0.37
-0.11 -0.28 -0.01 -0.18 -0.27 1.00 -0.18 -0.00 -0.16
-0.22 -0.28 0.08 -0.45 0.22 -0.18 1.00 -0.16 -0.17
0.08 -0.04 -0.15 0.28 -0.19 -0.00 -0.16 1.00 0.33
0.50 0.13 -0.11 -0.29 0.37 -0.16 -0.17 0.33 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


Cement
Blast Furnace Slag
Fly Ash
Water
Superplasticizer
Coarse Aggregate
Fine Aggregate
Age
Concrete compressive strength
Concrete compressive stre




```
# Select columns for visualization (excluding 'Concrete compressive
strength')
cols = [col for col in df.columns if col != 'Concrete compressive
strength']

# Create a figure with subplots
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(13, 25))
axes = axes.flatten()

# Define colors for plots
colors = ['b', 'r', 'g', 'c', 'm', 'k', 'lime', 'c']

# Iterate through columns and create distribution plots
for i, col in enumerate(cols):
    sns.histplot(df[col], color=colors[i], kde=True, ax=axes[i])
    axes[i].set_facecolor("w")
    axes[i].axvline(df[col].mean(), linestyle="dashed", label="mean",
color="k")
    axes[i].set_title(col, color="navy")
    axes[i].legend()

# Adjust layout and show plot
```

```
plt.tight_layout()
plt.show()
```


Cement
mean
Count
0
20
40
60
80
100
120
100
200
300
400
500
Cement
Blast Furnace Slag
mean
Count
0
100
200
300
400
500
0
50
100
150
200
250
300
350
Blast Furnace Slag
Fly Ash
mean
Count
0
100
200
300
400
500
0
25
50
75
100
125
150
175
200
Fly Ash
Water
mean
Count
0
25
50
75
100
125
150
175
120
140
160
180
200
220
240
Water
Superplasticizer
mean
Count
50
100
150
200
250
300
350
400
Coarse Aggregate
mean
Count
20
40
60
80
100
120

```
# Split the data into features (X) and target (y)
X_concrete = df.drop('Concrete compressive strength', axis=1)
y_concrete = df['Concrete compressive strength']

# Split the data into training and testing sets
X_train_concrete, X_test_concrete, y_train_concrete, y_test_concrete =
train_test_split(
    X_concrete, y_concrete, test_size=0.3, random_state=0
)

# Scale the features using StandardScaler
scaler_concrete = StandardScaler()
X_train_concrete = scaler_concrete.fit_transform(X_train_concrete)
X_test_concrete = scaler_concrete.transform(X_test_concrete)

# Create an SVR model
svr_model = SVR(kernel='linear', C=1)

# Train the model
svr_model.fit(X_train_concrete, y_train_concrete)

# Make predictions on the test set
y_pred_concrete = svr_model.predict(X_test_concrete)

# Evaluate the model
mse = mean_squared_error(y_test_concrete, y_pred_concrete)
r2 = r2_score(y_test_concrete, y_pred_concrete)

print("SVR Model Results:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

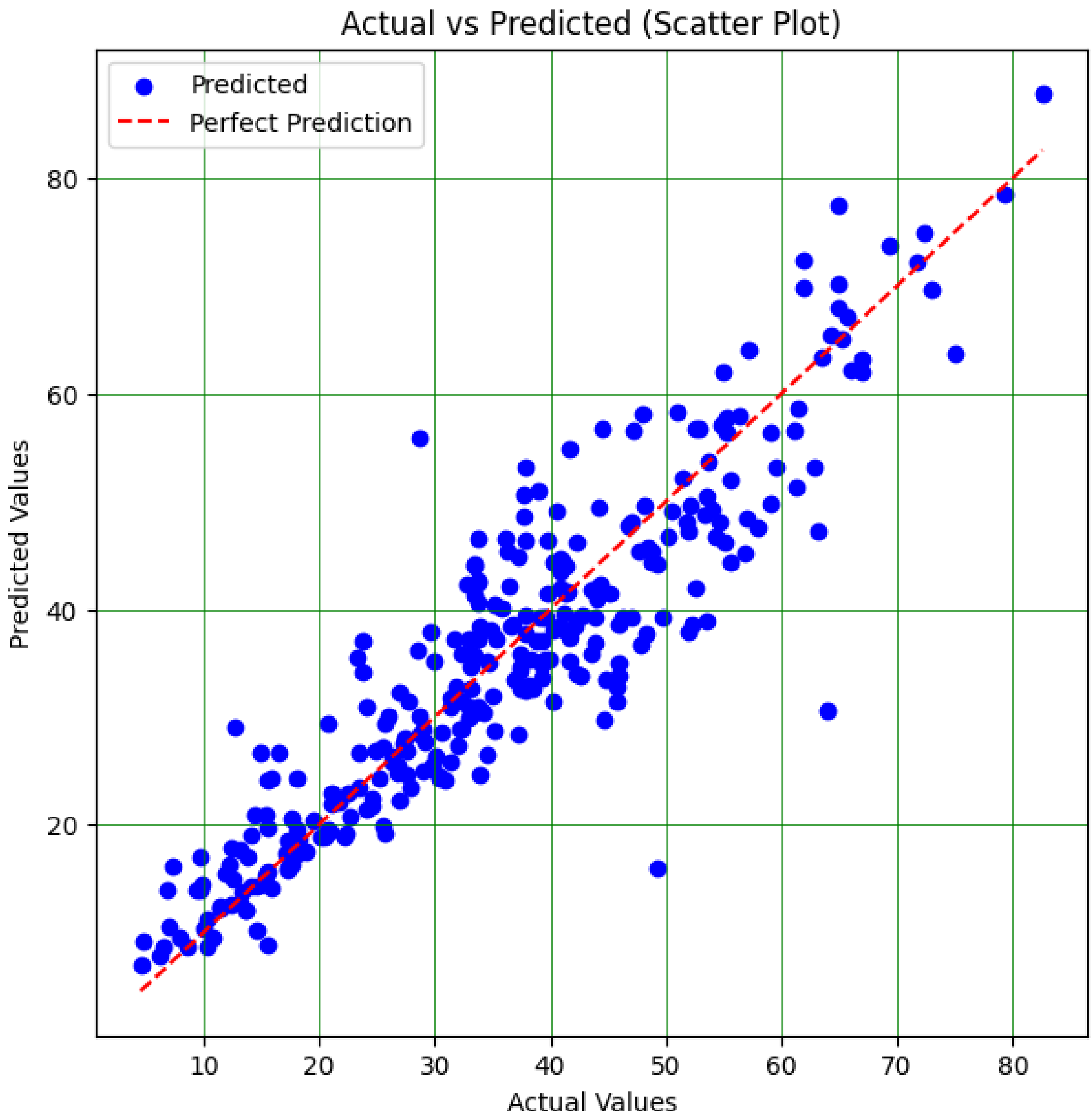
```
SVR Model Results:
Mean Squared Error: 93.07450744225504
R-squared: 0.6374191118739791
```

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=3)  # Try different degrees
X_train_concrete_poly = poly.fit_transform(X_train_concrete)
X_test_concrete_poly = poly.transform(X_test_concrete)

svr_model.fit(X_train_concrete_poly, y_train_concrete)
y_pred_concrete = svr_model.predict(X_test_concrete_poly)

# Evaluate the model
mse = mean_squared_error(y_test_concrete, y_pred_concrete)
r2 = r2_score(y_test_concrete, y_pred_concrete)

print("SVR Model Results:")
```

```python
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
SVR Model Results:
Mean Squared Error: 44.55026320014946
R-squared: 0.8264500727293151
```

```python
# Create a DataFrame from the actual and predicted values
dat = pd.DataFrame({'Actual': y_test_concrete, 'Predicted':
y_pred_concrete})

# Sort the DataFrame by the 'Actual' column
dat_sorted = dat.sort_values(by=['Actual'])

# Scatter plot of Actual vs Predicted
plt.figure(figsize=(7, 7))

plt.scatter(dat_sorted['Actual'], dat_sorted['Predicted'],
color='blue', label='Predicted')

# Plot the perfect prediction line (Actual = Predicted)
plt.plot(dat_sorted['Actual'], dat_sorted['Actual'], color='red',
label='Perfect Prediction', linestyle='--')

# Add grid, labels, and title
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.title('Actual vs Predicted (Scatter Plot)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Add a legend
plt.legend()

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



Actual vs Predicted (Scatter Plot)
Predicted
Perfect Prediction
80
60
40
20
Predicted Values
10
20
30
40
50
60
70
80
Actual Values