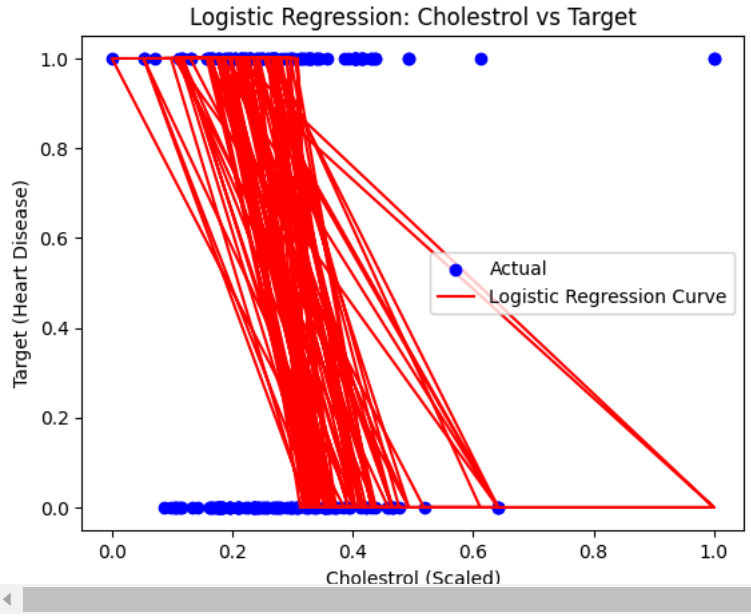
Simple Logistic Regression - Heart Disease - Scatter Plot cholestrol vs target

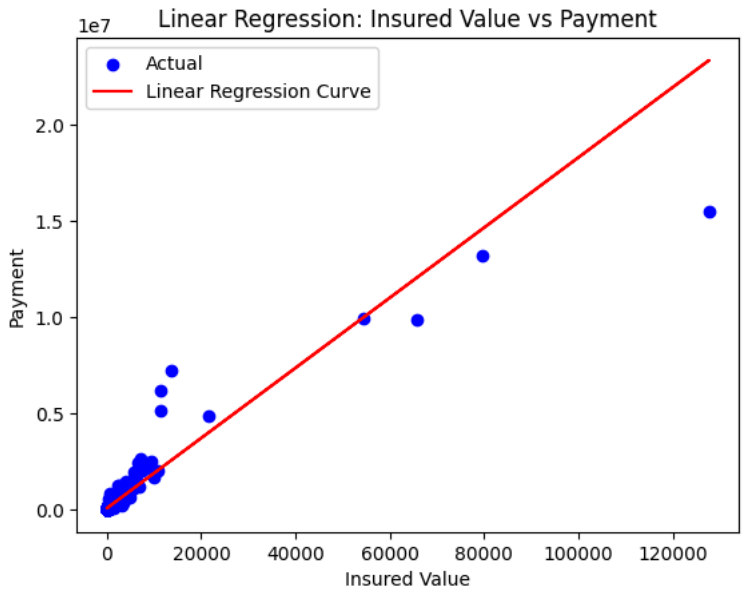
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('/content/Heart_Disease_Dataset.csv')

# Extract independent and target variables
X = data[['chol']]
y = data['target']

# Min-Max scaling for 'chol'
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train a simple logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Get predictions on the test set
y_pred = model.predict(X_test)

# Plot the scatter plot with the regression curve
plt.figure()
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Logistic Regression Curve')
plt.xlabel('Cholestrol (Scaled)')
plt.ylabel('Target (Heart Disease)')
plt.title('Logistic Regression: Cholestrol vs Target')
plt.legend()
plt.show()
```



Simple Linear Regression - Car Insurance - Scatter Plot - Insured value vs payment

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
```

```
data = pd.read_csv('/content/Car_Insurance_Dataset.csv')

# Extract independent and target variables
X = data[['Insured']]
y = data['Payment']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a simple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Get predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Plot the scatter plot with the regression curve
plt.figure()
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Linear Regression Curve')
plt.xlabel('Insured Value')
plt.ylabel('Payment')
plt.title('Linear Regression: Insured Value vs Payment')
plt.legend()
plt.show()

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

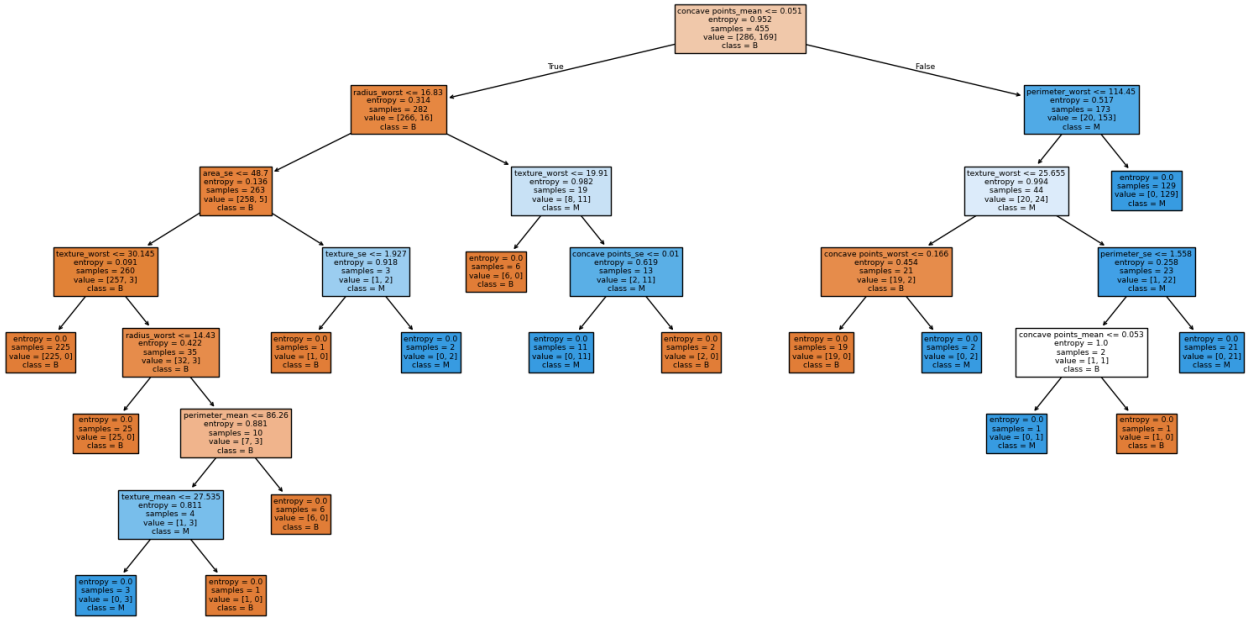


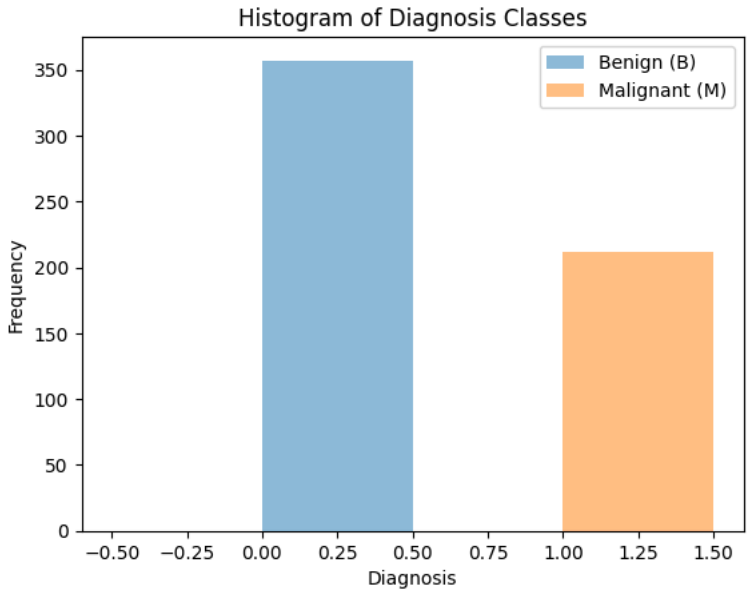

```
Mean Squared Error: 290225809666.27216
R-squared: 0.827746780432404
```

Decision Tree - Breast Cancer - Visualize the tree - Histogram

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import plot_tree
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('/content/Breast_Cancer_Dataset.csv')

# Drop the 'id' column
data = data.drop('id', axis=1)

# Encode the 'diagnosis' column
label_encoder = LabelEncoder()
data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])
# 'B' will be 0, 'M' will be 1
```

```
19
20 # Separate features (X) and target (y)
21 X = data.drop('diagnosis', axis=1)
22 y = data['diagnosis']
23
24 # Split data into training and testing sets
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
26
27 # Create a Decision Tree Classifier with 'entropy' criterion
28 model = DecisionTreeClassifier(criterion='entropy', random_state=42)
29
30 # Train the model
31 model.fit(X_train, y_train)
32
33 # Visualize the decision tree
34 plt.figure(figsize=(20, 10))
35 plot_tree(model, feature_names=X.columns, class_names=['B', 'M'], filled=True)
36 plt.show()
37
38 # Plot the histogram for the classes 'B' and 'M'
39 plt.figure()
40 plt.hist(data[data['diagnosis'] == 0]['diagnosis'], bins=2, alpha=0.5, label='Benign (B)')
41 plt.hist(data[data['diagnosis'] == 1]['diagnosis'], bins=2, alpha=0.5, label='Malignant (M)')
42 plt.xlabel('Diagnosis')
43 plt.ylabel('Frequency')
44 plt.title('Histogram of Diagnosis Classes')
45 plt.legend()
46 plt.show()
```

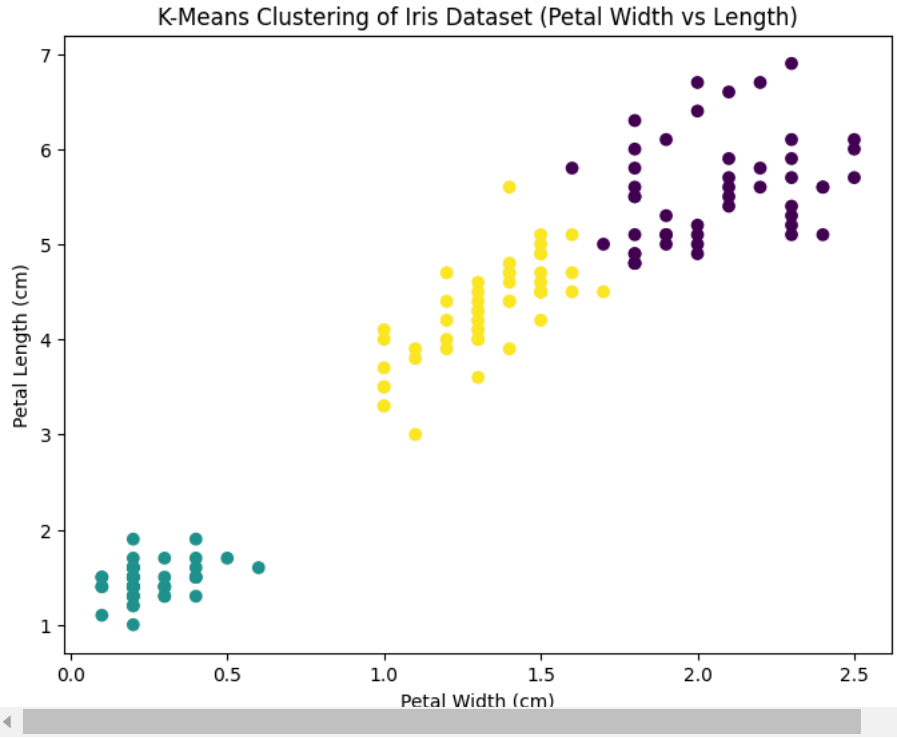




K-means - Iris - Scatter Plot - Petal width vs Length

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
data = pd.read_csv('/content/Iris_Dataset.csv')

# Extract features for clustering
X = data[['petal.width', 'petal.length']]

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-Means clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
```

```
19
20 # Get cluster labels
21 cluster_labels = kmeans.labels_
22
23 # Add cluster labels to the DataFrame
24 data['cluster'] = cluster_labels
25
26 # Plot the scatter plot of Petal Width vs Length, colored by cluster
27 plt.figure(figsize=(8, 6))
28 plt.scatter(data['petal.width'], data['petal.length'], c=data['cluster'], cmap='viridis')
29 plt.xlabel('Petal Width (cm)')
30 plt.ylabel('Petal Length (cm)')
31 plt.title('K-Means Clustering of Iris Dataset (Petal Width vs Length)')
32 plt.show()
```

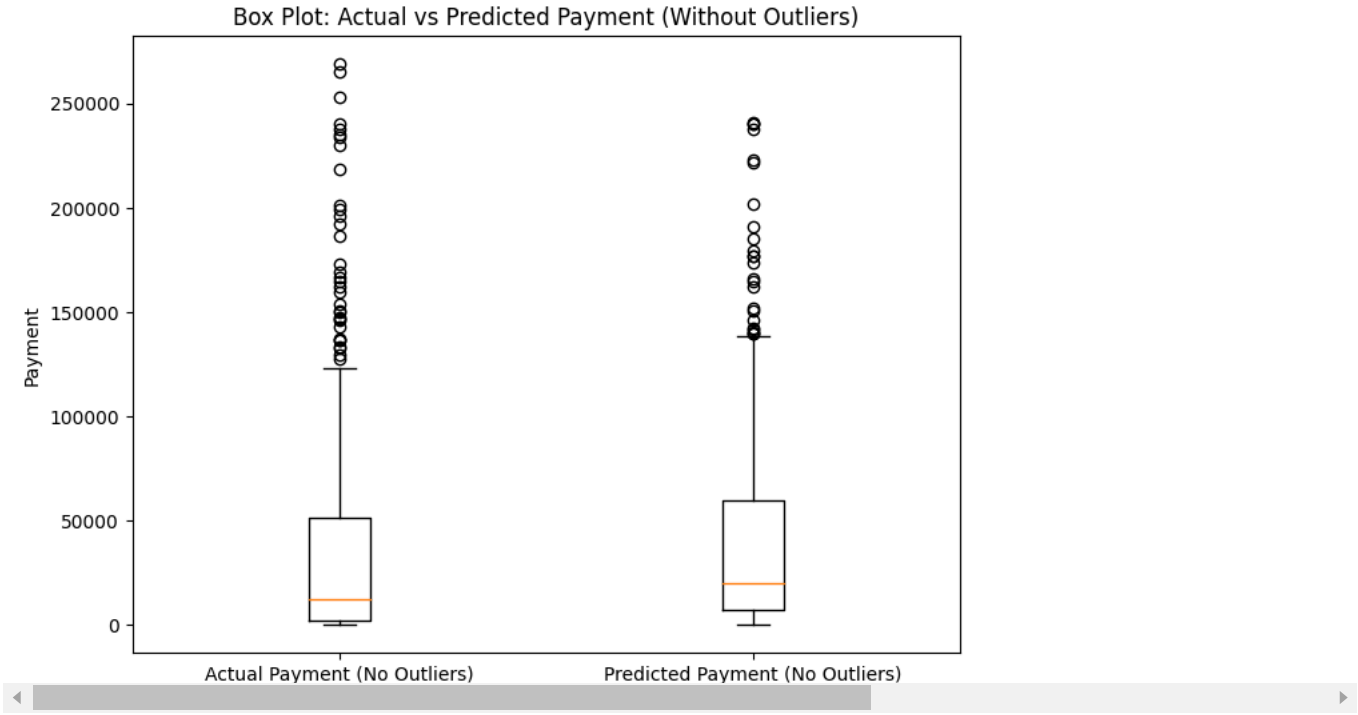


Ensemble - Car Insurance - Random Forest - Box Plot

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.ensemble import RandomForestRegressor
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import mean_squared_error, r2_score
7
8
9 # Load the dataset
10 data = pd.read_csv('/content/Car_Insurance_Dataset.csv')
11
12 # Extract independent and target variables
13 X = data.drop('Payment', axis=1)  # Features
14 y = data['Payment']  # Target
15
16 # Split data into training and testing sets
17 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
18
19 # Train a Random Forest Regressor
20 model = RandomForestRegressor(n_estimators=100, random_state=42)
21 model.fit(X_train, y_train)
22
23 # Make predictions on the test set
24 y_pred = model.predict(X_test)
25
26 # Evaluate the model
27 mse = mean_squared_error(y_test, y_pred)
28 r2 = r2_score(y_test, y_pred)
29
30 print(f"Mean Squared Error: {mse}")
31 print(f"R-squared: {r2}")
32
33
```

```
def remove_outliers_iqr(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data_no_outliers = data[(data >= lower_bound) & (data <= upper_bound)]
    return data_no_outliers


# Remove outliers from actual and predicted payments
y_test_no_outliers = remove_outliers_iqr(y_test)
y_pred_no_outliers = remove_outliers_iqr(y_pred)

# Plot box plots for actual vs predicted values without outliers
plt.figure(figsize=(8, 6))
plt.boxplot([y_test_no_outliers, y_pred_no_outliers],
            labels=['Actual Payment (No Outliers)', 'Predicted Payment (No Outliers)'])
plt.title('Box Plot: Actual vs Predicted Payment (Without Outliers)')
plt.ylabel('Payment')
plt.show()
```

```
Mean Squared Error: 23865698624.8345
R-squared: 0.9866576771943127
```

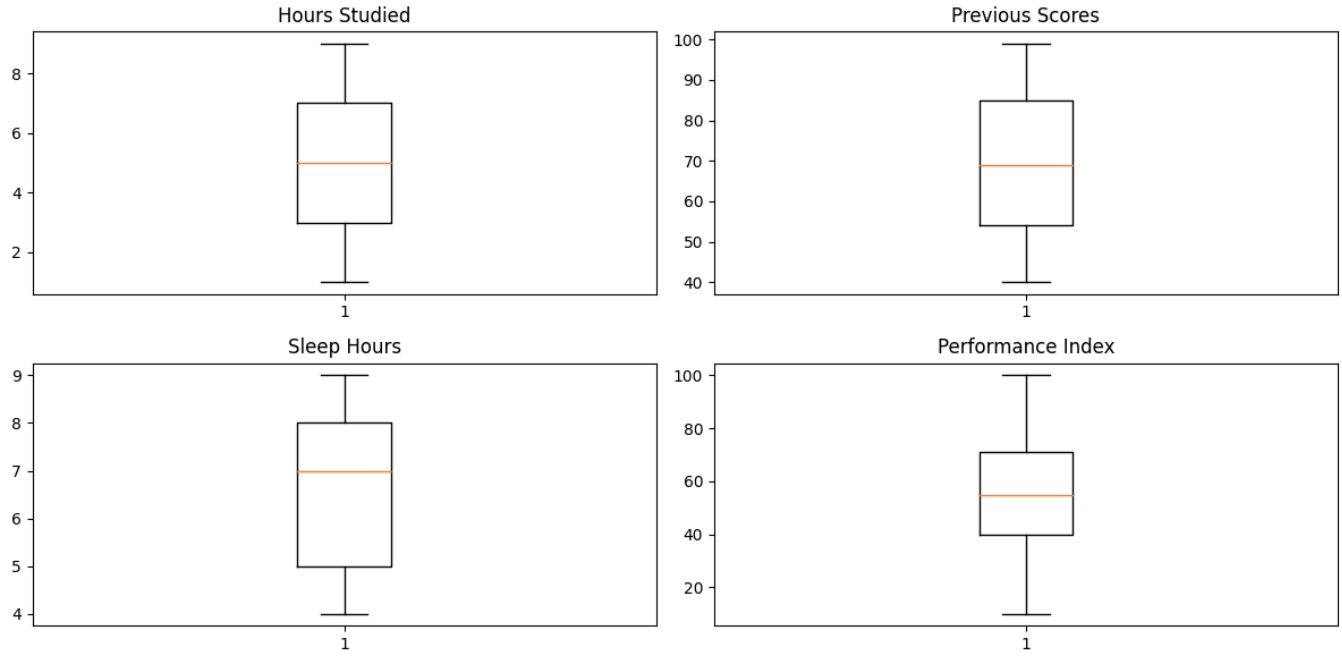


Multiple Linear Regression - Student Performance - Box Plot

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv("/content/Student_Performance_Dataset.csv")

# Define independent and dependent variables
X = df[['Previous Scores', 'Hours Studied', 'Sleep Hours']]
y = df['Performance Index']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

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

# Evaluate the model
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot box plots for the specified variables
plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.boxplot(df['Hours Studied'])
plt.title('Hours Studied')

plt.subplot(2, 2, 2)
plt.boxplot(df['Previous Scores'])
plt.title('Previous Scores')

plt.subplot(2, 2, 3)
plt.boxplot(df['Sleep Hours'])
plt.title('Sleep Hours')

plt.subplot(2, 2, 4)
plt.boxplot(df['Performance Index'])
plt.title('Performance Index')

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

```
Mean Squared Error: 4.545107899420576
R-squared: 0.9877353198949831
```

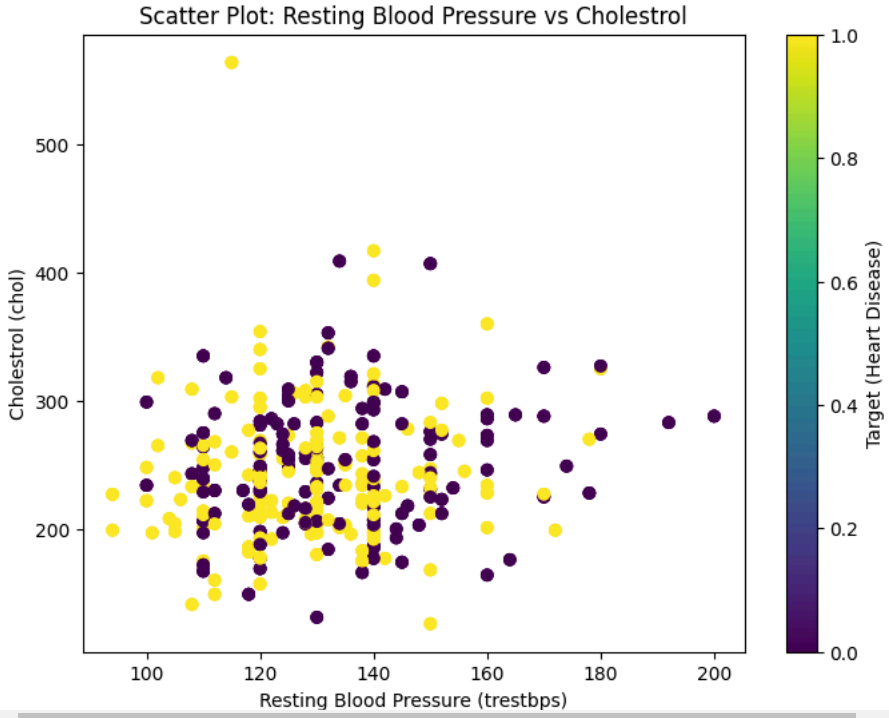


Multiple Logistic Regression - Heart Disease - Scatter plot Resting BP vs Cholestrol

```
# Load the dataset
df = pd.read_csv("/content/Heart_Disease_Dataset.csv")

# Define independent and dependent variables
X = df[['age', 'trestbps', 'chol']]
y = df['target']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

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

```
17
18 # Evaluate the model
19 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
20
21 accuracy = accuracy_score(y_test, y_pred)
22 precision = precision_score(y_test, y_pred)
23 recall = recall_score(y_test, y_pred)
24 f1 = f1_score(y_test, y_pred)
25
26 print(f"Accuracy: {accuracy}")
27 print(f"Precision: {precision}")
28 print(f"Recall: {recall}")
29 print(f"F1-score: {f1}")
30
31 # Scatter plot between 'trestbps' and 'chol'
32 plt.figure(figsize=(8, 6))
33 plt.scatter(df['trestbps'], df['chol'], c=df['target'], cmap='viridis')
34 plt.xlabel('Resting Blood Pressure (trestbps)')
35 plt.ylabel('Cholestrol (chol)')
36 plt.title('Scatter Plot: Resting Blood Pressure vs Cholestrol')
37 plt.colorbar(label='Target (Heart Disease)')
38 plt.show()
```

```
Accuracy: 0.6146341463414634
Precision: 0.6224489795918368
Recall: 0.5922330097087378
F1-score: 0.6069651741293532
```



Ensemble - XGBoost - Car Insurance - Boxplot

```
1  import pandas as pd
2  import xgboost as xgb
3  from sklearn.model_selection import train_test_split
4  from sklearn.metrics import mean_squared_error, r2_score
5  import matplotlib.pyplot as plt
6
7  # Load the dataset
8  data = pd.read_csv('/content/Car_Insurance_Dataset.csv')
9
10 # Extract independent and target variables
11 X = data.drop('Payment', axis=1)  # Features
12 y = data['Payment']  # Target
13
14 # Split data into training and testing sets
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
16
17 # Train an XGBoost Regressor
18 model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, random_state=42)
19 model.fit(X_train, y_train)
20
21 # Make predictions on the test set
```

```
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

def remove_outliers_iqr(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data_no_outliers = data[(data >= lower_bound) & (data <= upper_bound)]
    return data_no_outliers


# Remove outliers from actual and predicted payments
y_test_no_outliers = remove_outliers_iqr(y_test)
y_pred_no_outliers = remove_outliers_iqr(y_pred)

# Plot box plots for actual vs predicted values without outliers
plt.figure(figsize=(8, 6))
plt.boxplot([y_test_no_outliers, y_pred_no_outliers],
            labels=['Actual Payment (No Outliers)', 'Predicted Payment (No Outliers)'])
plt.title('Box Plot: Actual vs Predicted Payment (Without Outliers)')
plt.ylabel('Payment')
```