LAB 6 REPORT

Lab Task 1:

```
#-----------Task 1-----------------

model2 = tree. DecisionTreeRegressor()
model2.fit(X_train, y_train)
print("Decision Tree")
print("==========")
y_pred_train2 = model2.predict(X_train)
RMSE_train2 = mean_squared_error (y_train, y_pred_train2)
print("Decision Tree Train set: RMSE {}".format(RMSE_train2))
y_pred_test2 = model2.predict(X_test)
RMSE_test2 = mean_squared_error(y_test,y_pred_test2)
print("Decision Tree Test set: RMSE {}".format(RMSE_test2))
print("================")
```

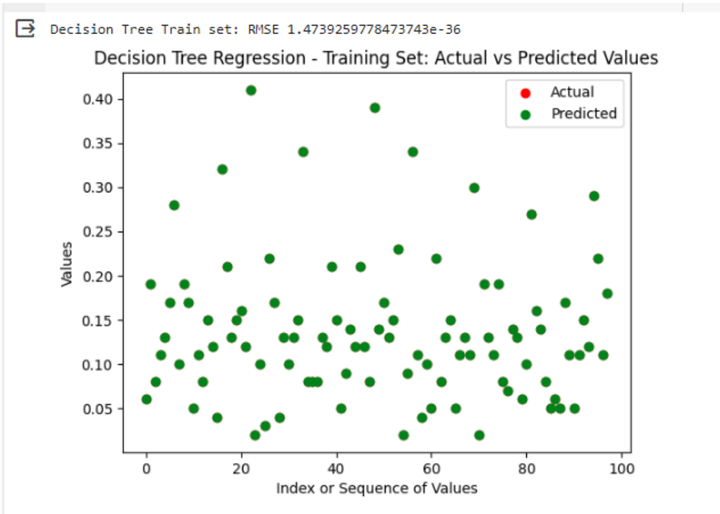
```
model2 = tree. DecisionTreeRegressor()

model2.fit(X_train, y_train)

print("Decision Tree")

y_pred_train2 = model2.predict(X_train)

RMSE_train2 = mean_squared_error(y_train,y_pred_train2)

print("Decision Tree Train set: RMSE {}".format(RMSE_train2))

y_pred_test2 = model2.predict(X_test)

RMSE_test2 = mean_squared_error(y_test,y_pred_test2)
print("Decision Tree Test set: RMSE {}".format (RMSE_test2))
print("-------------------------")
```

```
Decision Tree
Decision Tree Train set: RMSE 1.4739259778473743e-36
Decision Tree Test set: RMSE 0.009276000000000001
-------------------------
```

Lab Task 2:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
```

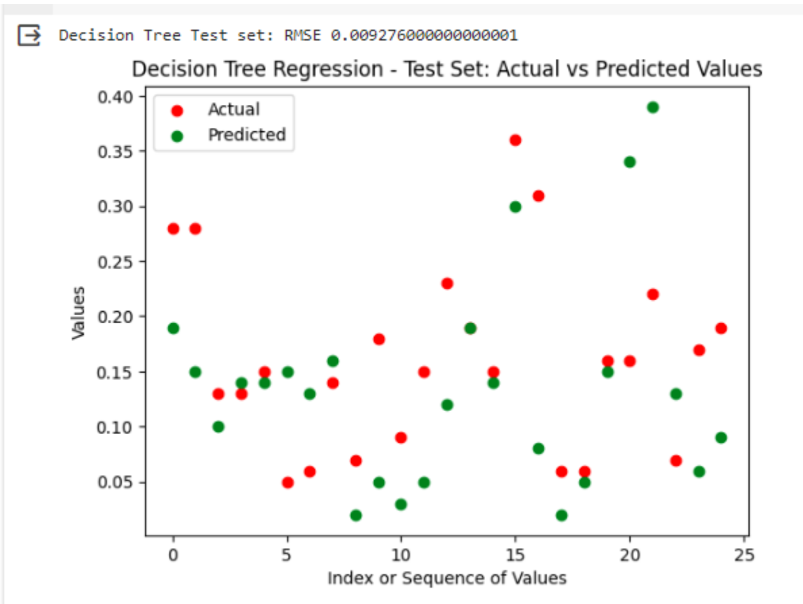
```
# Assuming X_train, y_train, X_test, y_test are defined and imported

# Create and fit the Decision Tree Regressor model
model2 = DecisionTreeRegressor()
model2.fit(X_train, y_train)

# Predictions on the training set
y_pred_train2 = model2.predict(X_train)

# Calculate RMSE for the training set
RMSE_train2 = mean_squared_error(y_train, y_pred_train2)
print("Decision Tree Train set: RMSE {}".format(RMSE_train2))

# Create a scatter plot for Actual vs Predicted values on the training
set
x_values_train = np.arange(len(y_train))

plt.scatter(x_values_train, y_train, color='red', label='Actual')
plt.scatter(x_values_train, y_pred_train2, color='green',
label='Predicted')

plt.xlabel('Index or Sequence of Values')
plt.ylabel('Values')
plt.title("Decision Tree Regression - Training Set: Actual vs Predicted
Values")
plt.legend()
plt.show()
```



Lab task 3:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Assuming X_train, y_train, X_test, y_test are defined and imported

# Create and fit the Decision Tree Regressor model
model2 = DecisionTreeRegressor()
model2.fit(X_train, y_train)

# Predictions on the test set
y_pred_test2 = model2.predict(X_test)

# Calculate RMSE for the test set
RMSE_test2 = mean_squared_error(y_test, y_pred_test2)
print("Decision Tree Test set: RMSE {}".format(RMSE_test2))

# Create a scatter plot for Actual vs Predicted values on the test set
x_values_test = np.arange(len(y_test))

plt.scatter(x_values_test, y_test, color='red', label='Actual')
plt.scatter(x_values_test, y_pred_test2, color='green',
label='Predicted')

plt.xlabel('Index or Sequence of Values')
```

```
plt.ylabel('Values')
plt.title("Decision Tree Regression - Test Set: Actual vs Predicted
Values")
plt.legend()
plt.show()
```



Lab task 4:

```
model2 = tree. DecisionTreeRegressor()
model2.fit(X_train, y_train)
print("Decision Tree")

y_pred_train2 = model2.predict(X_train)

RMSE_train2 = mean_squared_error(y_train,y_pred_train2)
print("Decision Tree Train set: RMSE {}".format(RMSE_train2))

y_pred_test2 = model2.predict(X_test)
RMSE_test2 = mean_squared_error(y_test,y_pred_test2)
print("Decision Tree Test set: RMSE {}".format (RMSE_test2))
```

```
print("------------------")
```

```
model2 = tree. DecisionTreeRegressor()
model2.fit(X_train, y_train)
print("Decision Tree")

y_pred_train2 = model2.predict(X_train)

RMSE_train2 = mean_squared_error(y_train,y_pred_train2)
print("Decision Tree Train set: RMSE {}".format(RMSE_train2))

y_pred_test2 = model2.predict(X_test)
RMSE_test2 = mean_squared_error(y_test,y_pred_test2)
print("Decision Tree Test set: RMSE {}".format (RMSE_test2))
print("-----------------")
```

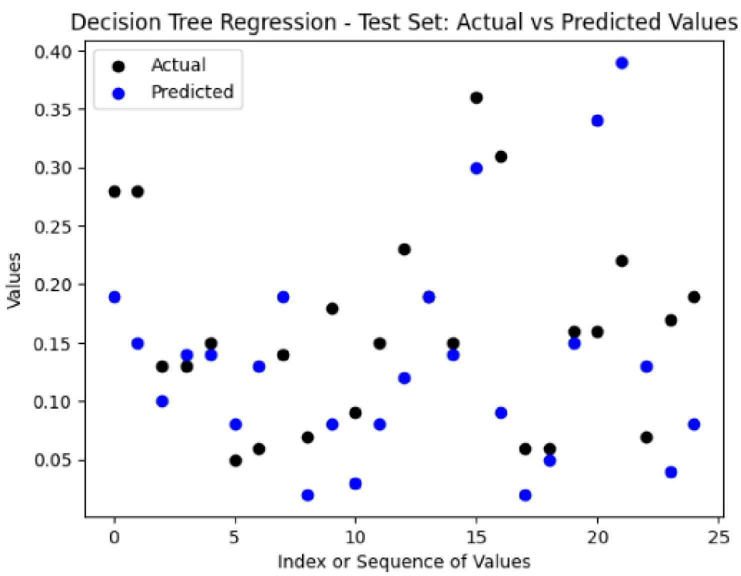
```
Decision Tree
Decision Tree Train set: RMSE 1.4739259778473743e-36
Decision Tree Test set: RMSE 0.009052
------------------
```

Lab Task 5:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error



# Predictions on the test set
y_pred_test2 = model2.predict(X_test)

# Calculate RMSE for the test set
RMSE_test2 = mean_squared_error(y_test, y_pred_test2)
print("Decision Tree Test set: RMSE {}".format(RMSE_test2))

# Create a scatter plot for Actual vs Predicted values on the test set
x_values_test = np.arange(len(y_test))

plt.scatter(x_values_test, y_test, color='black', label='Actual')
plt.scatter(x_values_test, y_pred_test2, color='blue',
label='Predicted')

plt.xlabel('Index or Sequence of Values')
plt.ylabel('Values')
```

```
plt.title("Decision Tree Regression - Test Set: Actual vs Predicted
Values")
plt.legend()
plt.show()

# Print additional information
print("Decision Tree Test set: RMSE {}".format(RMSE_test2))
print("-----------------")
```

```
Decision Tree Test set: RMSE 0.008612000000000002
```




```
Decision Tree Test set: RMSE 0.008612000000000002
-----------------
```

Post Lab:

Model 1 outperformed Model 2 and 3 on the test set, displaying the lowest RMSE and superior predictive performance. The success of Model 1 may be attributed to its capacity to capture complex data relationships. However, it's crucial to acknowledge the simplicity of Models 2 and 3, which might make them more interpretable but at the cost of predictive accuracy. Visualizations, particularly scatter plots, offered insights into the models' behavior, revealing patterns and potential outliers. These insights can inform strategies for improvement. Feature engineering and hyperparameter tuning are recommended for all models to enhance predictive capabilities.

Additionally, exploring ensemble methods such as Random Forest or Gradient Boosting could further boost overall performance. The decision on which strategy to prioritize should consider the balance between interpretability and predictive accuracy based on the specific goals and constraints of the problem.