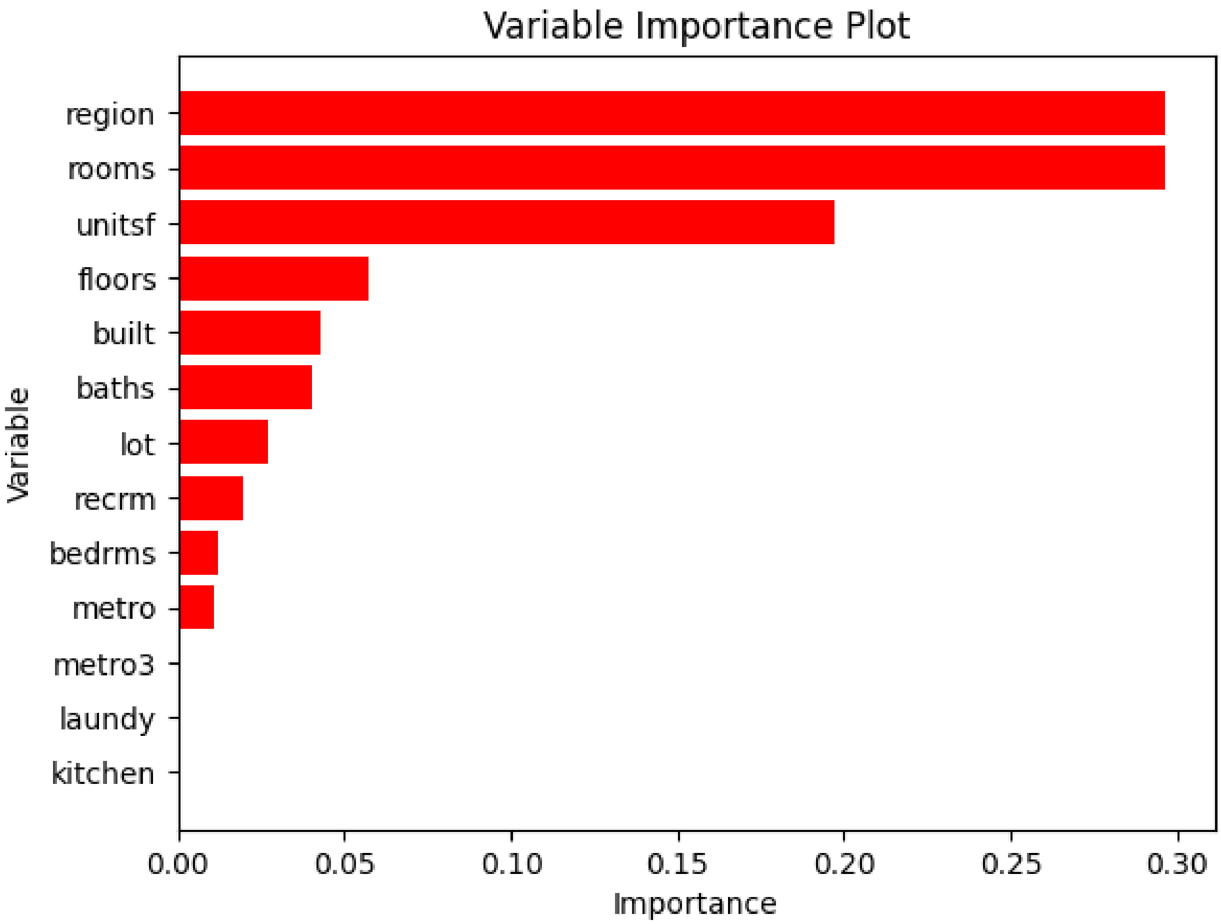
TheDebugger

R^2: 0.320673581416 26967

MSE: 0.8115699034007182

Question 2:



ChatGPT Interactions:

It suggested I traverse all the depths to get the best tree instead of just pruning it at one max depth.

how can i implement k-fold cross validation to get the best regression tree possible

To implement k-fold cross-validation to find the best regression tree model, you can follow these steps using Python and scikit-learn:

1. Import the necessary libraries:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold, cross_val_score
from sklearn.tree import DecisionTreeRegressor
```

1. Load your dataset (replace 'dataset.csv' with your data file) and prepare your features `X` and target variable `y`.

```python
data = pd.read_csv('dataset.csv')
X = data.drop('Target_Column', axis=1)  # Features
y = data['Target_Column']  # Target variable
```

1. Define the number of folds for cross-validation (`k`) and create a Decision Tree Regressor.

```python
k = 5  # You can adjust the number of folds
model = DecisionTreeRegressor(max_depth=10)  # You can adjust tree de
```

Regenerate

1. Perform k-fold cross-validation and calculate the mean squared error (MSE) as the evaluation metric.

```python
kf = KFold(n_splits=k, shuffle=True, random_state=42)  # Create a k-fold cro

# Perform cross-validation
mse_scores = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squared_

# Calculate the mean MSE across folds
mean_mse = np.mean(mse_scores)
```

1. Repeat steps 3 and 4 with different hyperparameters of the Decision Tree Regressor (e.g., different `max_depth` values) to find the best model.

Here's a complete example using a loop to find the best tree depth:

```python
best_depth = None
best_mean_mse = float('inf')

for depth in range(1, 11):  # Trying different tree depths from 1 to 10
    model = DecisionTreeRegressor(max_depth=depth)
    mse_scores = -cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squa
    mean_mse = np.mean(mse_scores)

    if mean_mse < best_mean_mse:
        best_mean_mse = mean_mse
        best_depth = depth

print(f"Best Tree Depth: {best_depth}")
print(f"Best Mean MSE: {best_mean_mse}")
```

This code performs k-fold cross-validation with different tree depths and selects the one that results in the lowest mean squared error, which is a common criterion for model selection in regression tasks. You can adjust the range of tree depths and other parameters to fine-tune the model further.

<u>My Code:</u>

My approach was to loop through all depths, find best pruned tree at each depth, and then choose the one with the minimal MSE. Please see the code across the following pages.

```python
# Import Statements for everything we use
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import matplotlib.pyplot as plt
import sklearn.model_selection as skm
from ISLP.models import ModelSpec as MS
from sklearn.tree import (DecisionTreeRegressor as DTR, plot_tree)
from sklearn.metrics import r2_score


# Load data and check that it looks okay
houseData = pd.read_csv("./Econ424_F2023_PC2_training_set_v1.csv")
print(houseData.describe())


# Format data as needed for the model
```

```python
model = MS(houseData.columns.drop('LOGVALUE'), intercept=False)
D = model.fit_transform(houseData)
feature_names = list(D.columns)
print(len(feature_names))
X = np.asarray(D)

# Split data into train and val, used 80-20 split since we already have 10,000 in given test set
(X_train,
X_val ,
y_train ,
y_val) = skm.train_test_split(X,
houseData['LOGVALUE'], test_size=0.2, random_state=20)

# Create base regression tree of depth to compare our findings with
reg = DTR(max_depth=8)
reg.fit(X_train, y_train)
ax = subplots(figsize=(12,12))[1]
plot_tree(reg,feature_names=feature_names, ax=ax)
np.mean((y_val - reg.predict(X_val))**2)

# Apply k-fold cross validation on it to prune it
ccp_path = reg.cost_complexity_pruning_path(X_train, y_train)
kfold = skm.KFold(6, shuffle=True, random_state=20)
grid = skm.GridSearchCV(reg, {'ccp_alpha': ccp_path.ccp_alphas}, refit=True, cv=kfold,
scoring='neg_mean_squared_error')
G = grid.fit(X_train, y_train)

# Plot the pruned tree found at max_depth=8 for this dataset
best_ = grid.best_estimator_
ax = subplots(figsize=(12,12))[1]
plot_tree(G.best_estimator_ ,feature_names=feature_names, ax=ax)

# Print the mean squared error for original regression tree vs best one from k-fold
print(np.mean((y_val - reg.predict(X_val))**2))
print(np.mean((y_val - best_.predict(X_val))**2))
```

```python
# General Idea:
# Loop through possible depths, create regression trees and perform cross-validation
# Take the best possible tree from all depth levels
# Plot errors across each tree depth best/pruned trees

# Track all MSE for each depth as we run the loop
depths = np.array([])
trainingMSE = np.array([])
valMSE = np.array([])
kFoldMSE = np.array([])

# Initialize a best estimator found so far
bestKFold_MSE = np.mean((y_val - best_.predict(X_val))**2)
bestEstimator = best_
bestDepth = bestEstimator.get_depth()

# Loop through all possible depths
for depth in range(1,13):
    # Make a regression tree for current depth
    reg = DTR(max_depth=depth)
    reg.fit(X_train, y_train)

    # Use unpruned tree to make predictions on train and val set
    mse_reg_val = np.mean((y_val - reg.predict(X_val))**2)
    mse_reg_train = np.mean((y_train - reg.predict(X_train))**2)

    # Apply k-fold cross-validation to find best pruned tree
    ccp_path = reg.cost_complexity_pruning_path(X_train, y_train)
    kfold = skm.KFold(5, shuffle=True, random_state=20)
    grid = skm.GridSearchCV(reg, {'ccp_alpha': ccp_path.ccp_alphas}, refit=True, cv=kfold,
scoring='neg_mean_squared_error')
    G = grid.fit(X_train, y_train)

    # Best pruned tree for current depth after cross validation
    currBest = grid.best_estimator_
    currMSE = np.mean((y_val-currBest.predict(X_val))**2)
    print(currMSE)
```

```python
    # Append prediction accuracies to the relative arrays
    depths = np.append(depths, depth)
    kFoldMSE = np.append(kFoldMSE, currMSE)
    valMSE = np.append(valMSE, mse_reg_val)
    trainingMSE = np.append(trainingMSE, mse_reg_train)

    # Plot the best pruned tree found at each depth level
    ax = subplots(figsize=(12,12))[1]
    plot_tree(currBest,feature_names=feature_names, ax=ax)



    # compare with best mse seen so far, if better, update it
    if currMSE < bestKFold_MSE:
        # save new max MSE
        bestKFold_MSE = currMSE
        # save best tree
        bestEstimator = currBest



# Make our predictions on the entire training dataset
y_pred = bestEstimator.predict(X)
# Calculate MSE
outputMSE = np.mean((houseData["LOGVALUE"] - y_pred)**2)
# Calculate R Squared
outputRSquared = r2_score(houseData["LOGVALUE"], y_pred)

# Print them
print("Output MSE is: " + str(outputMSE))
print("Output R squared is: " + str(outputRSquared))






# Plot differences in Mean Squared Error across the different depths applying k-fold
plt.plot(depths, trainingMSE, label='training')
```

```
plt.plot(depths, valMSE, label='validation')
plt.plot(depths, kFoldMSE, label='cross-validation')
plt.xlabel("Depth")
plt.ylabel("Mean Squared Error")
plt.legend()
plt.title("K-Fold Cross Validation Across Depths")
plt.show()

# Print out the best depth that we found from the loop
bestDepth = bestEstimator.get_depth()
print("Best depth is at: " + str(bestDepth))
# Plot the best tree
ax=subplots(figsize=(12,12))[1]
plot_tree(bestEstimator,feature_names=feature_names,ax=ax)


# Create bar graph for question 2 feature importance
# Initalize the feature importance from the best tree
feature_imp = pd.DataFrame( {'importance':bestEstimator.feature_importances_}, index=feature_names)
feature_imp.sort_values(by='importance', ascending=True)

# Sort the names and importances
sorted_names, sorted_imp = zip(*sorted(zip(feature_names, feature_imp['importance']), key=lambda x: x[1]))

# Plot the bar graph
plt.barh(sorted_names, sorted_imp, label='Difference', color='red')
plt.xlabel("Difference")
plt.ylabel("Variable")
plt.title("Variable Importance Plot")
plt.show()




# Read in test dataset
testData = pd.read_csv('./Econ424_F2023_PC2_test_set_without_response_variable_v1.csv')
```

```
modelTest = MS(testData.columns, intercept=False)

# Format as needed
DTest = modelTest.fit_transform(testData)
feature_names = list(DTest.columns)

# Make Predictions on the Test Set
X_Test = np.asarray(DTest)
Y_Test = bestEstimator.predict(X_Test)

# Specify the file path where you want to save the CSV
csv_file = "./output.csv"

# Use numpy.savetxt to save the array as a CSV file
np.savetxt(csv_file, Y_Test, delimiter="\n", fmt="%f")
```