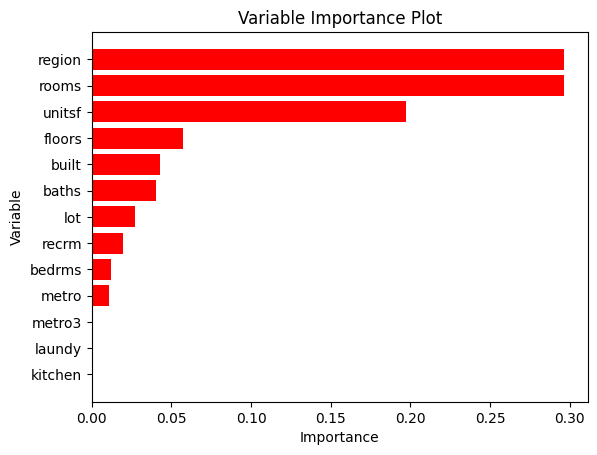
TheDebugger

R^2: 0.32067358141626967

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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

My Code:

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.

# 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

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))

# 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)

# 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")