CS 441 - Final Project

You only need to complete and submit this once for the group. Be sure to add the other group members to the submission in Gradescope. List them here also.

Group Member Names	**NetIDs**
Noel Mrowiec	mrowiec3
Hanliang Jiang	hj33

Complete the sections below. You do not need to fill out the checklist.

Total Points Available	**[ ] / 150**
1. Problem description	[ ] / 20
2. Model comparison	
a. Which models	[ ] / 5
b. Hyperparameter experiments	[ ] / 35
c. Best model/parameters result	[ ] / 10
3. Analysis: Training Size or Features	[ ] / 30
4. Stretch Goal: Innovation	
a. Approach description, experiments, innovation	[ ] / 25
b. Publishable with justification	[ ] / 25
5. Attribution / Group Contributions	[ ] -5 if incomplete (or page not selected)

1. Problem Description

Give an overview of the problem you are trying to solve.

We decided to do one of the pre-selected challenges to do housing price regression. We selected this problem specifically because it was a regression instead of classification problem. Additionally, we wanted data that provided both continuous and discrete data. Finally, we selected the housing price prediction because we wanted to work on cleaning the data and practice feature selection. With the housing sale pricing challenge, we found that the criteria we were seeking proved to be our greatest challenges for this data set.

Explain what you are trying to predict, and what inputs (features) are available for prediction.

Given the data set found here, we attempted to correctly predict the sales price for a given house. The data set included train and test samples. But, the test samples did not have a corresponding sales price for the house; so we did not use the test samples. The test samples were what the competition used to grade the models, thus the correct outputs were not provided. The features included in the data set were obvious features like the number of bedrooms, bathrooms, and square footage of the house. We intuitively knew that a bigger house would tend to sell for more money. There were some very odd features, like the type of alley access. In that case, we did not know what kind of effect having alley access would have on the sales price of the home. We also noticed there were some very subjective data features, in addition to the objective features. For example, one feature was labeled as “kitchen quality.” The quality of the kitchen could be ranked as excellent, good, typical/average, fair, and poor. We personally supposed that we would not be able to distinguish between the quality levels of a kitchen, besides in very obvious cases. These kinds of rankings caused us to deliberate on what features we thought were actually important. It also caused us to question the validity of the data. Who recorded this data? Who performed these kinds of qualitative ranking of the houses? Was it an expert realtor or someone trying to sell their house? Since we did not know the source of the data, we assumed the former instead of the latter. Therefore, we acknowledge that this project may have placed too much confidence in the validity of the data. We acknowledge that with more expert knowledge about this data set, we could have instantly eliminated extraneous or erroneous data. The results of the project considered all the features to be correct and are all potentially valid in reducing the error between the predicted and expected values. We decided to use the techniques learned in this class to statistically find the important features.

Explain the experimental setup. How many examples for train, val, test? What are the metrics?

The data included 1460 samples (houses) with 79 features. Each of these samples had a corresponding sales price. We split the training data into train and test sets, 70% training and 30% test. This split was required since the test data did not include ground truth, and therefore was unusable for our needs. The splitting of the data gave us 1021 training samples and 439 test samples. When finding the hyperparameters, we used cross validation instead of splitting a specific evaluation set. We decided to use cross validation because we thought it would be best given our limited amount of training samples. Cross validation for every model was achieved by k-folding, with k=5. K-folding splits the training data into 5 folds and finds the RMSE for each train/val splits. Finally, we found the average RMSE of those 5 folds and used it as a metric to select the best hyperparameter.
Since there were many categorical features, we decided to do one-hot encoding to represent our data. Thus, we converted the categories to 0/1 for each state in a given category. This expanded the number of features from 79 to 317. This method of encoding created the curse of dimensionality problem since we drastically expanded the number of features and had many sparse features as well.

Since we had such a small data set (only 1460 samples), we decided not to use any neural networks or deep networks (additionally, we encountered some trouble trying to load the data into a neural network).
The metrics we used to compare our 3 models was RMSE since this was a regression problem.

What are some of the challenges in solving this problem?

As explained above, we drastically increased the number of features when doing our one-hot encoding. This created a sparse matrix since each option since categorical features were transformed into their own features (with true being present and false otherwise). For example, there was a paved driveway feature in the original data with 3 options - paved, partial pavement, and dirt/gravel; that single feature transformed into 3 new features. The vast majority of houses had paved driveways. Thus, the features relating to driveways were very sparse since it was pretty much all 1's (true) for pavement and 0's (false) for the other two features. We opted to keep all of the features, however sparse, and let each model filter out features they didn't find helpful. For example, in linear regression, we found that using 72 features, determined from univariate univariate statistical tests, provided the best results.
Another problem was we had a lot of trouble getting the data clean enough to work with. Firstly, the issue was trying to decide how to represent the categorical data. We decided to use one-hot encoding, as described above, and use the get_dummies function to create more binary features. Additionally, we had many categories in the data set that had an option of NA. In a dataframe, NA is represented as not a number which was a problem. Also, some of the items were blank. By exploring the data, we found that the blank items in the data were for numerical values. So, it seemed to us that these values should be zero. This was an assumption of course, and thus may have affected our results.
We realized we didn't fully understand standardizing and normalizing data and keeping the train and test sets separate. At first we were standardizing and normalizing all of the data at once, which is incorrect because that would leak information from the test set into the training/validation set. Additionally, we were unsure if it was better to normalize or standardize the data. We created a little experiment on the data to test if we should normalize or standardize the data. After we transformed the data, did linear regression using both techniques and we got the following results: root mean squared error for normalized data: 1.6E+18 and root mean squared error for standardized data: 3.0E+17. Therefore, we decided to standardize the data. Of course, for nonlinear model, normalization may have been better.

2. Model Comparison

Which three models will you compare? Which hyperparameter(s) will you test?

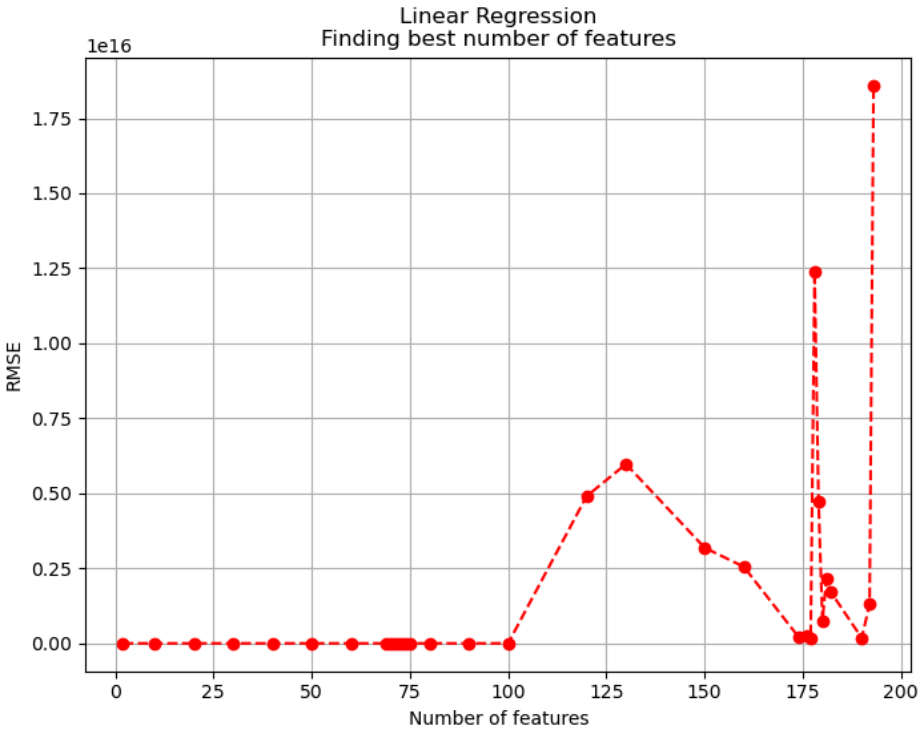
Model - Linear Regression: For linear regression, there are no hyperparameters to set. In order to find the best linear regression model, we modified other aspects as described below. During the fit, the linear regression model saw 317 features, which of course is the same as the number of features in the training/test data. Using all of these features gave us a very bad RMSE for linear regression. With all 317 features, we had a RMSE of 3.0E+17 which is worse than just finding a mean of all the training data (RMSE was about 78,000 using the mean). Thus, we decided to find the features that most accurately predicted the results. We selected the best features using the Scikit-Learn's SelectKBest method which uses univariate statistical tests to find the K best features. See the plot below for the linear regression's model using only the K best features on the training data. Using cross validation on the training data, we found that using 72 of the best features minimized the RMSE for the training data. The RMSE was 30,836.
Test Results:
Using the 72 best features of the test data, for our linear regression model, resulted in a RMSE=38,717.

Model - K nearest neighbor: For KNN, we used cross validation on the training data to find the ideal K nearest neighbors that minimized the RMSE. Thus K is the hyperparameter which the cross validation optimized. For training data, we found that K=9 gave us a RMSE=40,087. We used the KNN regressor from Scikit Learn. The distance is calculated as L2 distance.
Test Results:
Using K=9, we found the RMSE=43,776 for the test data.

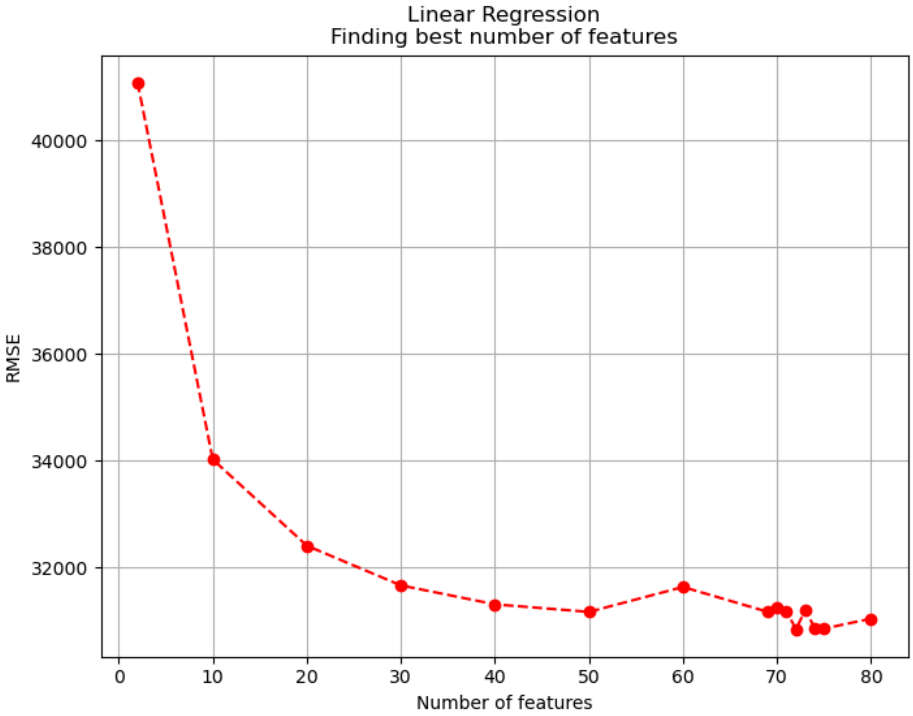
Model - Random Forest: For Random Forest, we used cross validation on the training data to find the ideal random forest that minimized the RMSE.The hyperparameters included the number of estimators and maximum depth of the trees. We used the Scikit-Learn's Random Forest Regressor. For training data, we found that the number of estimators=16 and maximum tree depth=64 gave us a RMSE=31,126 for the training data.
Test Results:
Using number of estimators=16 and maximum tree depth=64 on the test data gave us a RMSE=113,251

Use tables or plots to show the evaluated hyperparameter values for each model, and indicate which is best. These experiments should use only the training and validation sets. Feel free to delete the boxes, as long as it's clear.

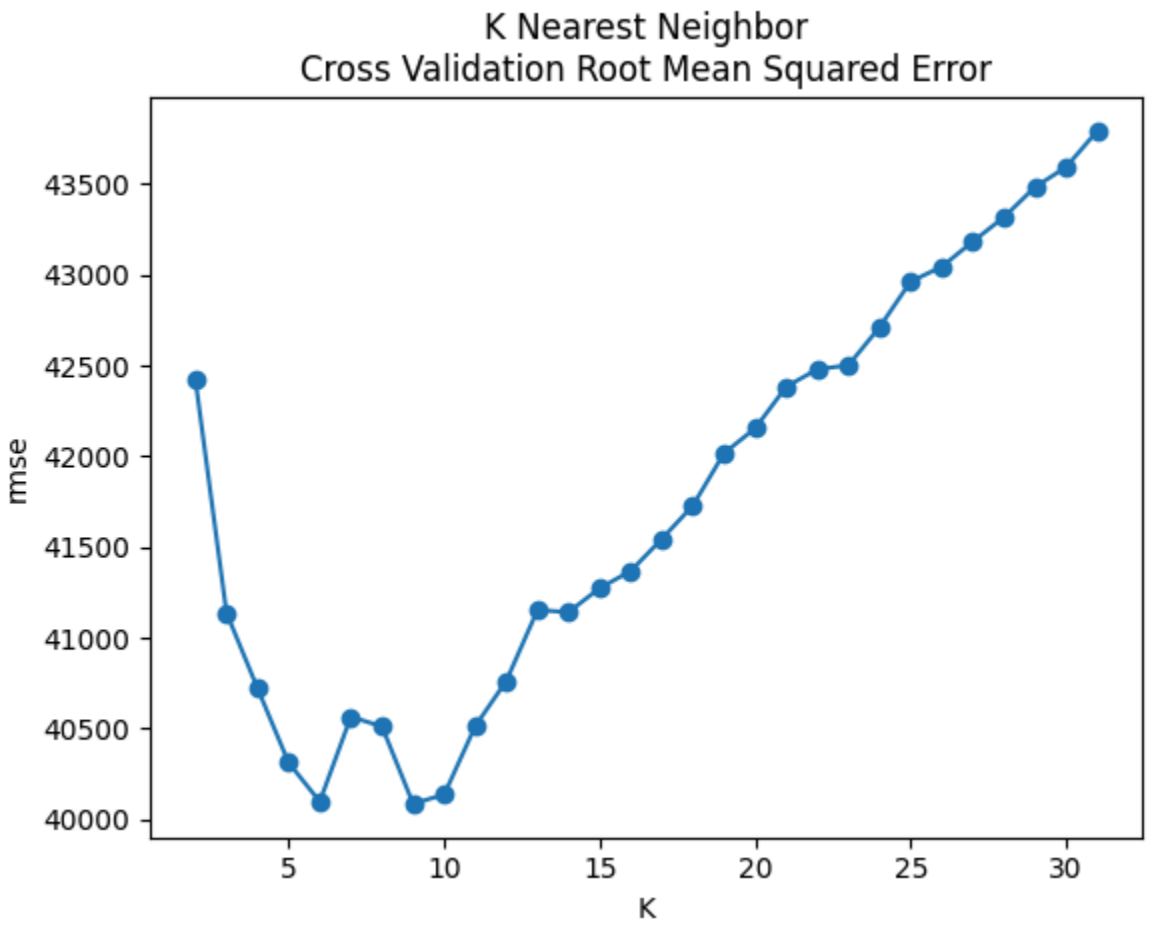
Below is the graph showing our cross validation of the training data for our linear regression in order to find the K best features. We did not show past 193 features because the RMSE continued to grow after 193 features. We can see that if the number of features is greater than 100 then the RMSE is greater than 1.0E+15. Therefore, in the next plot provide a closer look when the RMSE is minimized (less than 100 best features).



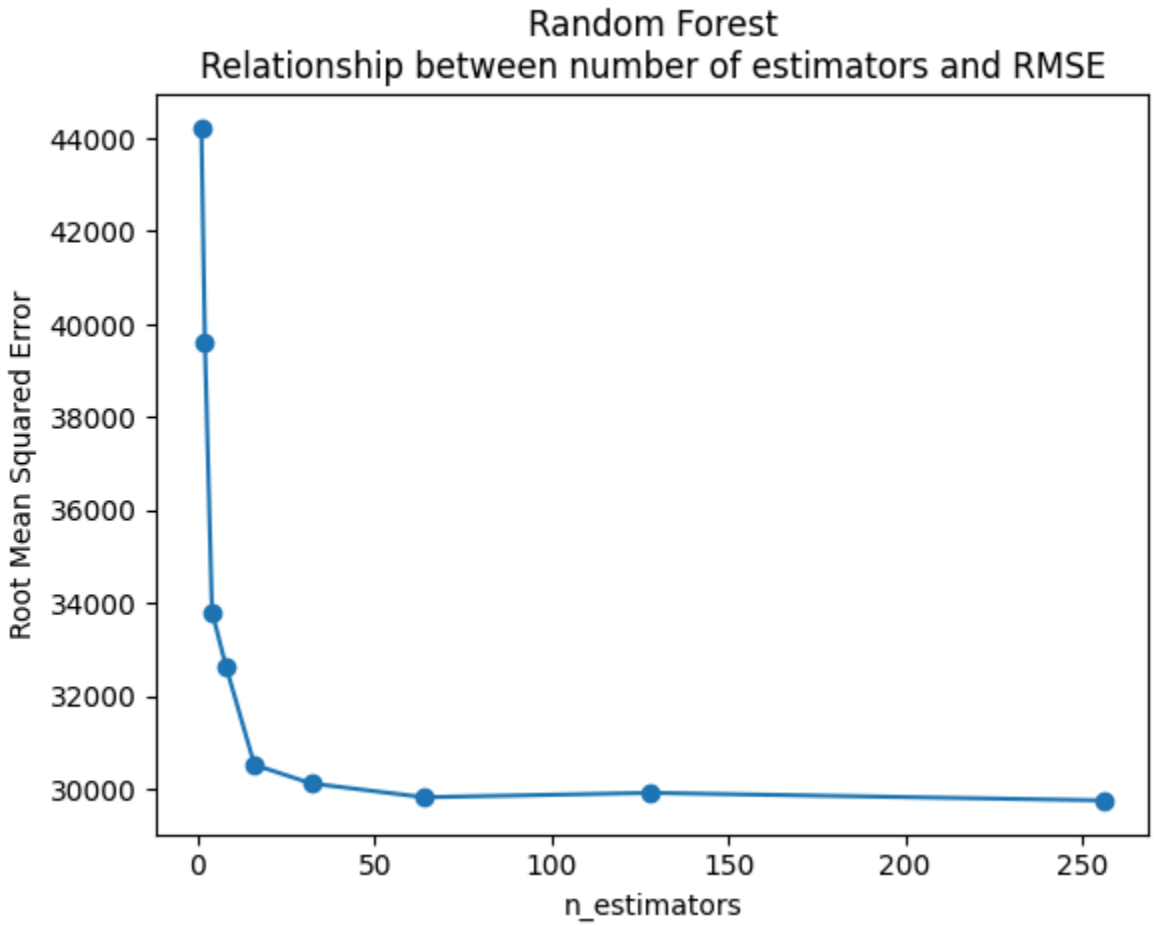
The plot below zooms in to when the number of features are below 100 features, and we can see that the RMSE is minimal with 72 best features of the training data.

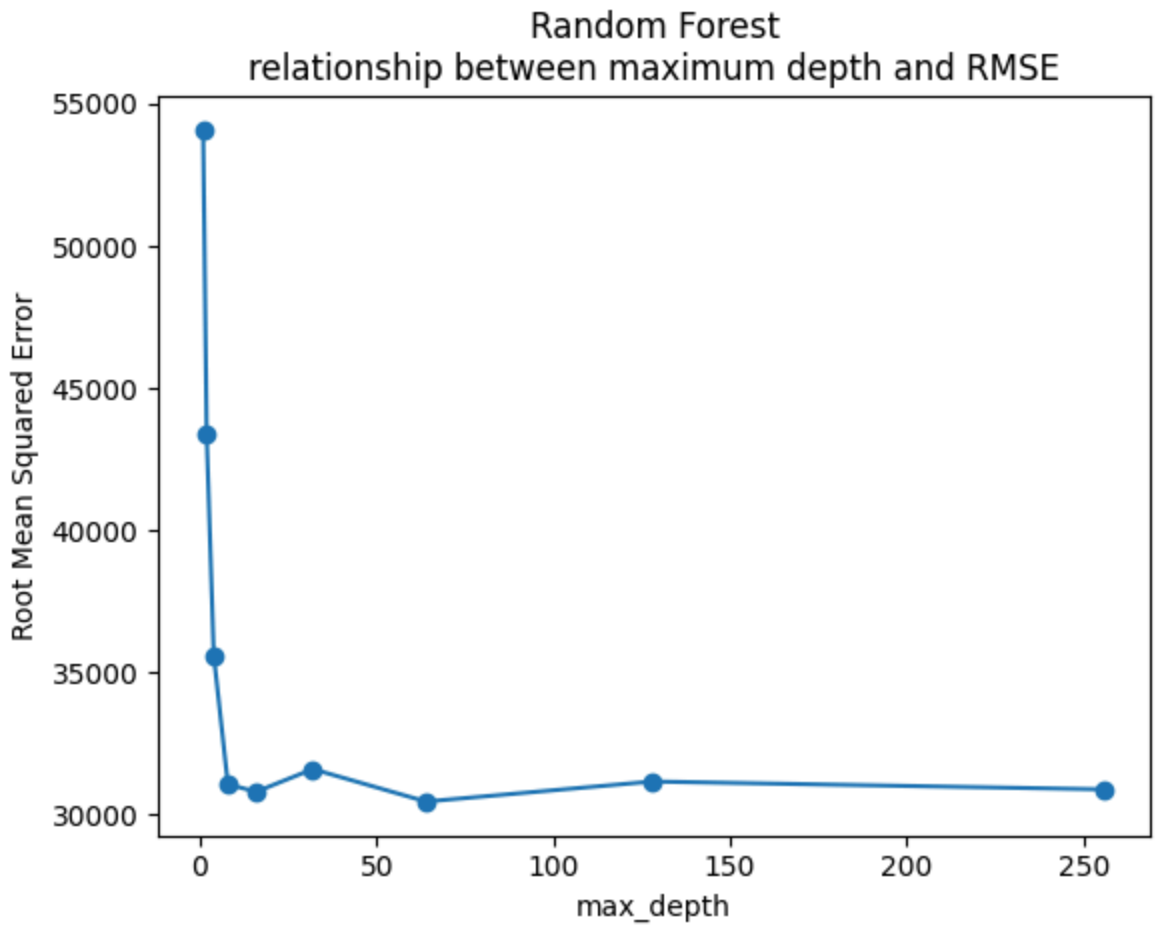


Below is the plot for the hyperparameters selection for KNN model on our training data. The only hyperparameter for the KNN model is the K number of neighbors. As above, we used cross validation to find the best K nearest neighbors. Below is the resulting plot from cross validation to find the K neighbors that minimize the RMSE. As displayed, the RMSE is minimized when K=9.



Below is the plot for the hyperparameters for a random forest model on our training data. The hyperparameters for the random forest model are the number of estimators and maximum depth of the trees in the forest. As above, we used cross validation to find the best number of estimators and ideal maximum depth to minimize the RMSE. Below are two plots showing the number of estimators vs the RMSE and the max depth vs. RMSE.





Which model and hyperparameters are best overall? Train it using the combination of the train and validation sets and report test performance.

Surprisingly, the linear regression worked the best on our trained data. We found that on the test data, the random forest performed the worst with a RMSE=113,251 and the linear regression performed the best with RMSE=38,717. We actually expected the random forest to perform the best instead of linear regression. We speculate that the poor performance of the random forest was due to our one-hot encoding and the increase in the number of dimensions from 79 to 317 features. We assume that if we passed the mixture of continuous and categorical data to the random forest, without the encoding, we would have seen a lower RMSE than the linear regression. For simplicity's sake, we did not implement a random forest with mixed data.
Note: since we did cross validation, there is not any held out validation data. Therefore, there is no need to “retrain [the] model on the union of training and validation sets” since the training process ultimately included all of the training data.

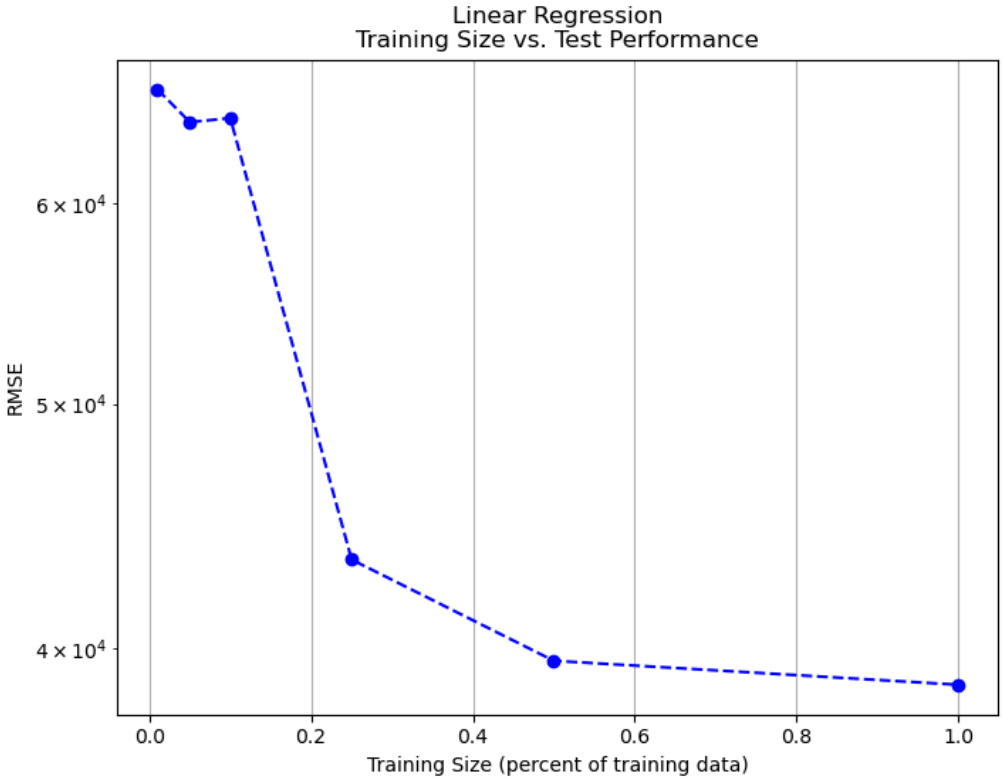
3. Additional Analysis: Either Effect of Training Data Size or Feature Analysis

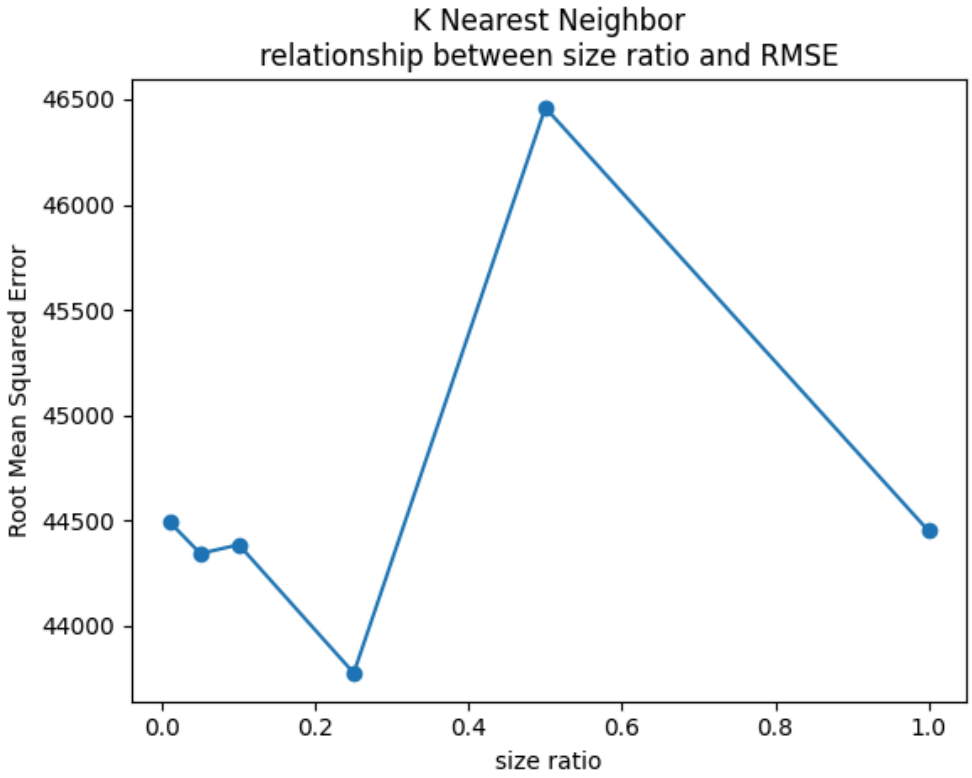
Which type of analysis are you doing?

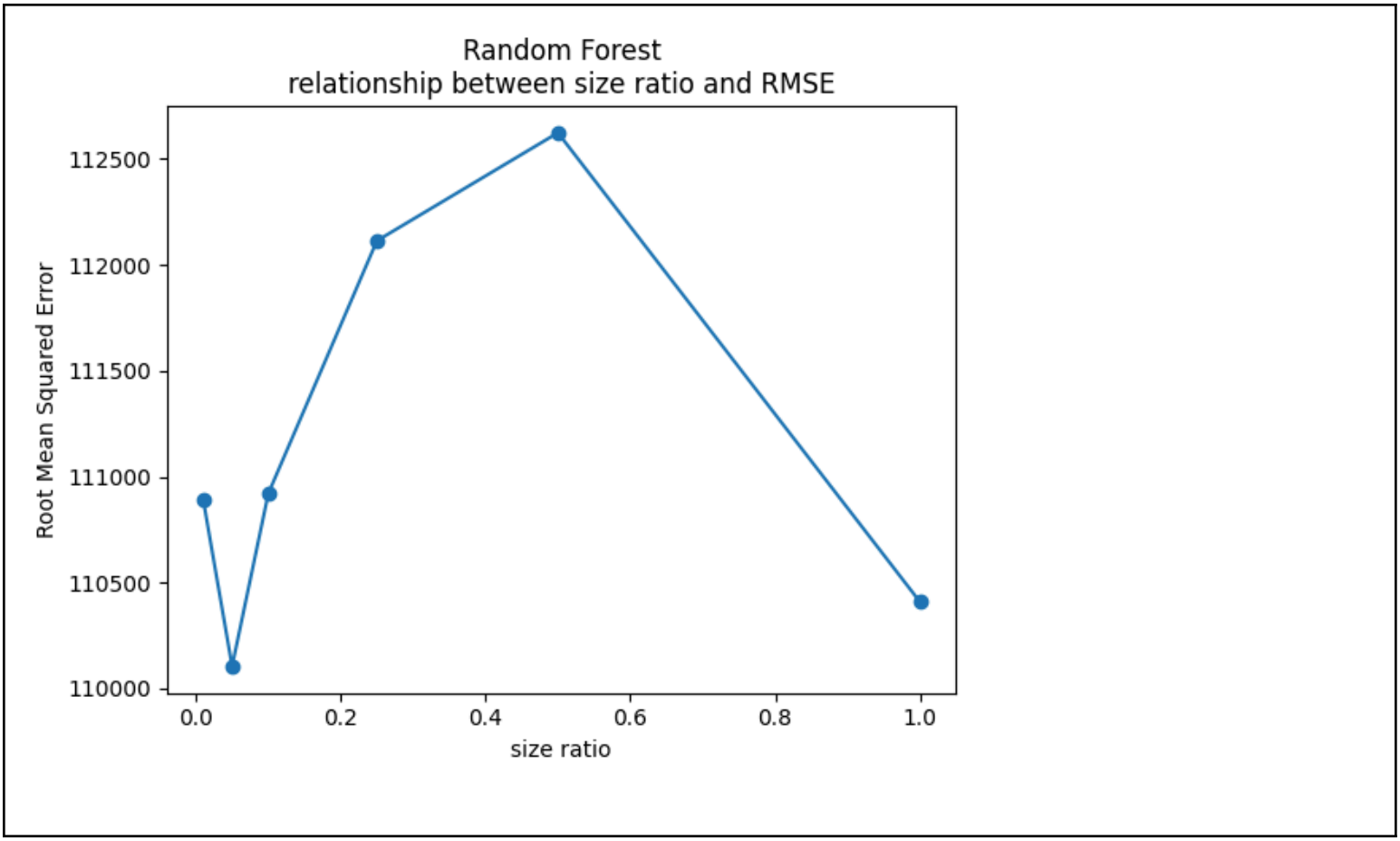
Training size

Present your experiments and conclusions. If evaluating features, include performance of your best model with different subsets of features and importance analysis using L1 regularization, mutual information, or similar. If evaluating training size, compare all three models using the selected hyperparameters.

In the plots below, we can see that linear regression performs best when using the entire training data set. We found that KNN and random forest perform better with a small amount of the training data. In fact, KNN performs best when 25% of the training data is used. For random forest, the model performs best when only 1% of the training data is used. As we discussed earlier, we can indeed see that the random forest model does not perform well with our one-hot encoded (and thus sparse) data. We can see more samples will confuse the trees more. As far as KNN, we believe that the sparse features reduce the model's accuracy similar to random forests . Thus, reducing the number of training samples allows for some features to be all the same. If all the features are the same for most of the samples, then that feature will not contribute to the distance calculation. We observe that fewer samples provide less opportunity for anomalies in features. Less sample presumably means more of the features are the same and do not negatively impact the performance of the model. On the other hand, we do see a dip in the RMSE when the full training data set is used, as compared to half of the data (for KNN and RF). Regardless, without removing features from the one-hot expanded data set, the models perform poorly.


Linear Regression
Training Size vs. Test Performance
RMSE
6×10^4
5×10^4
4×10^4
0.0
0.2
0.4
0.6
0.8
1.0
Training Size (percent of training data)


K Nearest Neighbor
relationship between size ratio and RMSE
Root Mean Squared Error
46500
46000
45500
45000
44500
44000
0.0
0.2
0.4
0.6
0.8
1.0
size ratio


Random Forest
relationship between size ratio and RMSE
Root Mean Squared Error
112500
112000
111500
111000
110500
110000
0.0
0.2
0.4
0.6
0.8
1.0
size ratio

4. Stretch Goal: Innovation

Describe your proposed approach

N/A

Provide and explain the experimental analysis of your proposed approach

N/A

What is innovative about your approach?

N/A

Is your approach potentially publishable? If not (the usual case), skip this part. If so, explain the research contributions and how it fits in with related work. What venue would you consider for publication?

N/A

5. Acknowledgments / Attribution

Link to your code (required!!)

https://github.com/noelmrowiec/CS441/tree/main/final_project

Link to your data (if not pre-selected)

Using the given House Prices dataset

External citations or resources

https://www.kaggle.com/code/prashant111/comprehensive-guide-on-feature-selection
https://towardsdatascience.com/predicting-house-prices-with-linear-regression-machine-learning-from-scratch-part-ii-47a0238aeac1
https://www.c-sharpcorner.com/article/a-beginners-guide-to-one-hot-encoding-using-pandas-getdummies-method/
https://www.kaggle.com/code/gusthema/house-prices-prediction-using-tfdf
https://machinelearningmastery.com/feature-selection-for-regression-data/
https://www.kaggle.com/code/hobeomlee/prediction-stacked-regressions
https://scikit-learn.org/
https://towardsdatascience.com/random-forest-regression-5f605132d19d
https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

Group member contributions

Noel Mrowiec - did data cleaning, writing the report, and linear regression, and was overall coordinator.
Hanliang Jiang - did random forest, KNN, github, and version source control.

Did group members contribute roughly equally or unequally? If unequally, explain and specify whether one member went above and beyond, or someone contributed less than agreed or expected.

Equally

N/A

In [ ]:

```
#read data
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("house-prices-advanced-regression-techniques/train.csv")
data_X, data_Y  = data.iloc[:,1:-1], data.iloc[:,-1:]      #not including ID. Keep

print(f"Shape of raw data {data_X.shape}")

# test_data = pd.read_csv("house-prices-advanced-regression-techniques/test.csv").i
# test_X, test_Y  = test_data, pd.read_csv("house-prices-advanced-regression-techni

col_list = ['MSZoning','Street','Alley','LotShape','LandContour','Utilities','LotCo


print(f"train_X.shape[1] {data.shape[1]}, test_X.shape[1] {data.shape[1]}")

print(f"all_data.shape[1] {data_X.shape[1]}")

#currently, data has NaN
total_nan_count = pd.isna(data_X).sum().sum()
print(f"Total number of NaN values in the DataFrame: {total_nan_count}")

#fix the NaN
data_X[col_list] = data_X[col_list].fillna('No')
data_X = data_X.fillna(0)

#now, no NaN or blanks
total_nan_count = pd.isna(data_X).sum().sum()
print(f"Total number of NaN values in the DataFrame: {total_nan_count}")

#data_X.to_csv("data_X.txt", sep="\t", index=False)

#treat the month and year sold as a category
#source https://www.kaggle.com/code/hobeomlee/prediction-stacked-regressions
data_X['MoSold'] = data_X['MoSold'].replace({1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr'
data_X['YrSold'] = data_X['YrSold'].astype('str')

#do one-hot encoding
all_data_dummy = pd.get_dummies(data_X).astype(float)
#print(all_data_dummy.describe(include='all'))


train_X_raw, test_X_raw, train_Y, test_Y = train_test_split(all_data_dummy, data_Y,

#standarize
scaler = StandardScaler()
train_X_stand = scaler.fit_transform(train_X_raw)
test_X_stand  = scaler.transform(test_X_raw)

#normalized data
from sklearn.preprocessing import normalize
```

```
train_X_norm = normalize(train_X_raw, axis=0)
test_X_norm = normalize(test_X_raw, axis=0)

#don't use val unless needed
#train_X_norm, val_X_norm, train_Y_norm, val_Y_norm = train_test_split(train_X_norm

#todo only use the validation set when tuning do not set at top level because linea
#train_X, val_X, train_Y, val_Y = train_test_split(train_X, train_Y, train_size=.8,
#print(clear_data.columns)


assert train_X_norm.shape[1] == train_X_norm.shape[1], print(f"train_X.shape[1] {tr
print(f"train_X_stand.shape {train_X_stand.shape}, test_X_stand.shape {test_X_stand

# train_X.to_csv("train_X.txt", sep="\t", index=False)
# test_X.to_csv("test_X.txt", sep="\t", index=False)

total_nan_count = pd.isna(train_X_norm).sum().sum()
print(f"Total number of NaN values in the DataFrame: {total_nan_count}")
total_nan_count = pd.isna(test_X_norm).sum().sum()
print(f"Total number of NaN values in the DataFrame: {total_nan_count}")
# total_nan_count = pd.isna(val_X).sum().sum()
# print(f"Total number of NaN values in the DataFrame: {total_nan_count}")

# train_X.to_csv("train_X.txt", sep="\t", index=False)
# test_X.to_csv("test_X.txt", sep="\t", index=False)
# val_X.to_csv("val_X.txt", sep="\t", index=False)
#test_X = test_X.fillna('No')


assert train_X_stand.shape[1] == test_X_stand.shape[1], print(f"train_X.shape[1] {t
assert test_X_stand.shape[0] == test_Y.shape[0]
```

```
Shape of raw data (1460, 79)
train_X.shape[1] 81, test_X.shape[1] 81
all_data.shape[1] 79
Total number of NaN values in the DataFrame: 7829
Total number of NaN values in the DataFrame: 0
train_X_stand.shape (1021, 317), test_X_stand.shape (439, 317)
Total number of NaN values in the DataFrame: 0
Total number of NaN values in the DataFrame: 0
```

In [ ]:
```
#convert to numpy
print(type(train_Y))
train_Y = train_Y.to_numpy()
#val_Y   = val_Y.to_numpy()
test_Y  = test_Y.to_numpy()
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [ ]:
```
print(type(train_Y))

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#normalized
lr = LinearRegression().fit(train_X_norm, train_Y)
```

```
y_pred = lr.predict(test_X_norm)
rmse = np.sqrt(mean_squared_error(test_Y, y_pred))
print(f"Root Mean Squared Error for normalized data (RMSE): {rmse:.2E}")

#standardized
lr = LinearRegression().fit(train_X_stand, train_Y)
# Make predictions
y_pred = lr.predict(test_X_stand)
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, y_pred))
print(f"Root Mean Squared Error for standardized data (RMSE): {rmse:.2E}")

print(type(train_Y))
```

```
<class 'numpy.ndarray'>
Root Mean Squared Error for normalized data (RMSE): 1.58E+18
Root Mean Squared Error for standardized data (RMSE): 3.08E+17
<class 'numpy.ndarray'>
```

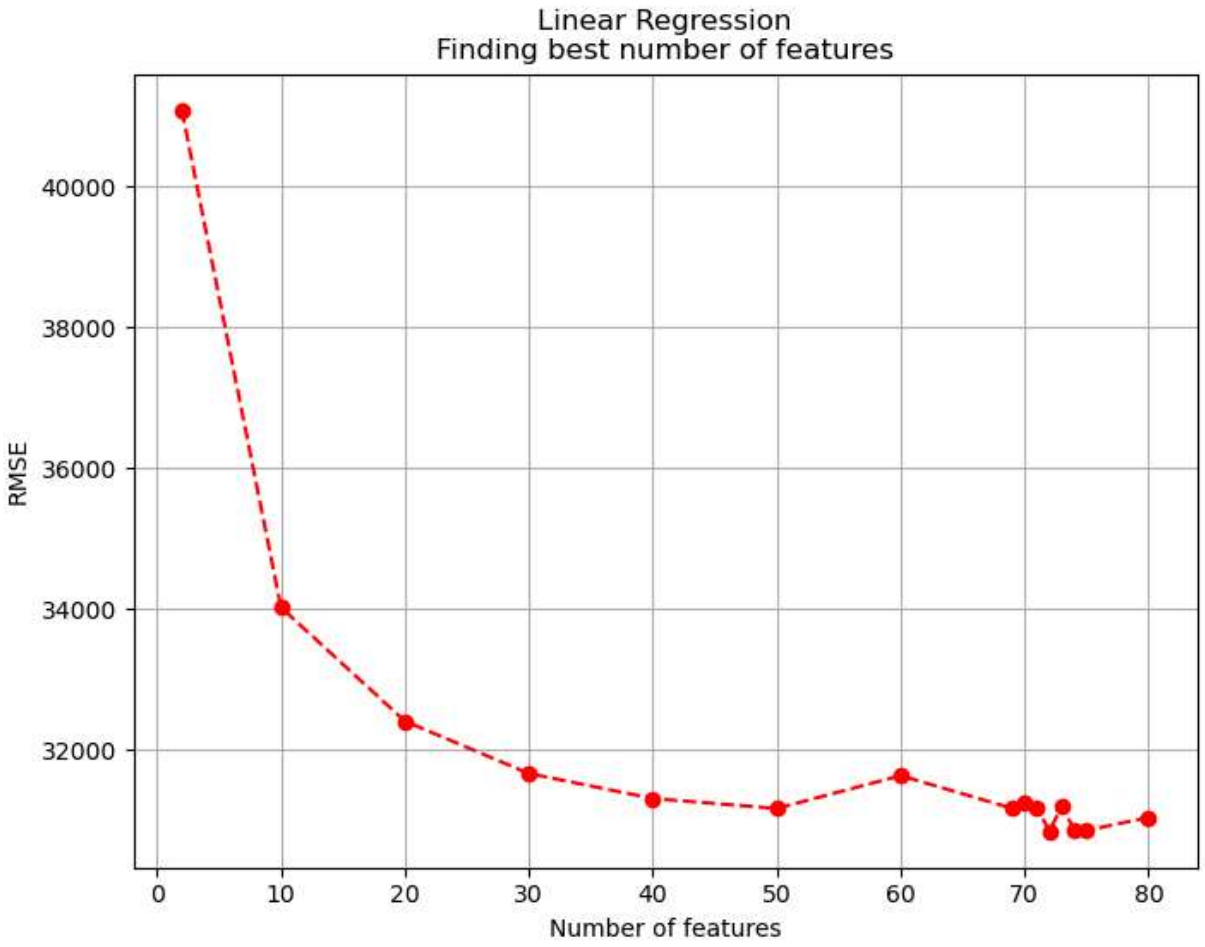
In [ ]:
```
#compare to just a average
y_pred = [train_Y.mean()] * len(train_Y)
rmse = np.sqrt(mean_squared_error(train_Y.flatten(), y_pred))
print(f"Baseline RMSE: {rmse:.2f}")
```

```
Baseline RMSE: 78129.67
```

In [ ]:
```
#picking features to do linear regression
#source https://machinelearningmastery.com/feature-selection-for-regression-data/

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score



def linear_regression_select(x_train, y_train, features):
    # feature selection
    #k is the number of top features

    cv_scores = []

    for k in features:
        feature_selection = SelectKBest(score_func=f_regression, k=k)
        feature_selection.fit(x_train, y_train.flatten())

        train_X_best_features = feature_selection.transform(x_train)
        #test_X_tranformed  = feature_selection.transform(x_val)
        lr = LinearRegression().fit(train_X_best_features, y_train.flatten())

        score = cross_val_score(lr, train_X_best_features, y_train, scoring='neg_ro

        cv_scores.append(np.mean(score)*-1)    #multiply by negitive 1 because gett
        #print(f"For train size 100%, number of features: {k}")
        # np.savetxt(f"linear_regression_output/coef_{1}.txt", np.round(model.coef_
        # print(f"model y-intercet: {model.intercept_}")
```

```
        #y_pred = lr.predict(test_X_tranformed)

        #Calculate RMSE
        # rmse = np.sqrt(mean_squared_error(y_val.flatten(), y_pred))
        # if rmse < min_rmse:
        #     min_features = k
        #     min_rmse = rmse

        # rmse_list.append(rmse)


    min_CV_RSME = min(cv_scores)
    index = cv_scores.index(min_CV_RSME)
    print(f"The min RSME is {min_CV_RSME:.0f} for {features[index]} number of featu



    # Create a plot
    plt.figure(figsize=(8, 6))
    plt.plot(features, cv_scores, marker="o", linestyle="--", color="r")
    #plt.yscale('log')
    plt.xlabel("Number of features")
    plt.ylabel("RMSE")
    plt.title("Linear Regression\nFinding best number of features")
    plt.grid(True)
    plt.show()

    return index

features = [2,10,20,30,40, 50, 60, 69, 70, 71, 72, 73,74,75, 80, ]#90, 100, 120, 13
best_features_index = linear_regression_select(train_X_stand, train_Y, features)
print(f"best features {features[best_features_index]}")
```

```
The min RSME is 30836 for 72 number of features
```




```
best features 72
```

In [ ]:
```
#using the ideal number of features to perform testing
best_num_features = features[best_features_index]

feature_selection = SelectKBest(score_func=f_regression, k=best_num_features)
feature_selection.fit(train_X_stand, train_Y.flatten())

train_X_best_features = feature_selection.transform(train_X_stand)
test_X_best_features  = feature_selection.transform(test_X_stand)

lr = LinearRegression().fit(train_X_best_features, train_Y.flatten())

y_pred = lr.predict(test_X_best_features)

rmse = np.sqrt(mean_squared_error(test_Y.flatten(), y_pred))

print(f"For testing our linear regression model with {best_num_features} features,
```

```
For testing our linear regression model with 72 features, the RSME is 38717.05
```

below didn't work

In [ ]:
```
#remove constant features
# using sklearn variancethreshold to find constant features

# from sklearn.feature_selection import VarianceThreshold
# sel = VarianceThreshold(threshold=0.1)
```

```
# sel.fit(train_X_stand)  # fit finds the features with zero variance

# # get_support is a boolean vector that indicates which features are retained
# # if we sum over get_support, we get the number of features that are not constant
# print(f"number of features that are not constant {sum(sel.get_support())}")
# print(f"num features {train_X_stand.shape}")
# # # print the constant features
# # print(
# #     len([
# #         x for x in train_X_stand.columns
# #         if x not in train_X_stand.columns[sel.get_support()]
# #     ]))
# mask = sel.get_support()
# false_indices = np.where(~mask)[0]
# print("Indices where the mask is False:", false_indices)

# # we can then drop these columns from the train and test sets
# train_X_trans = sel.transform(train_X_stand)
# test_X_trans = sel.transform(test_X_stand)

# #see how it did
# lr = LinearRegression().fit(train_X_trans, train_Y)
# y_pred = lr.predict(test_X_trans)
# rmse = np.sqrt(mean_squared_error(test_Y, y_pred))
# print(f"RMSE: {rmse:.2E}")
```

So we see that didn't work since we are already dropping those features value

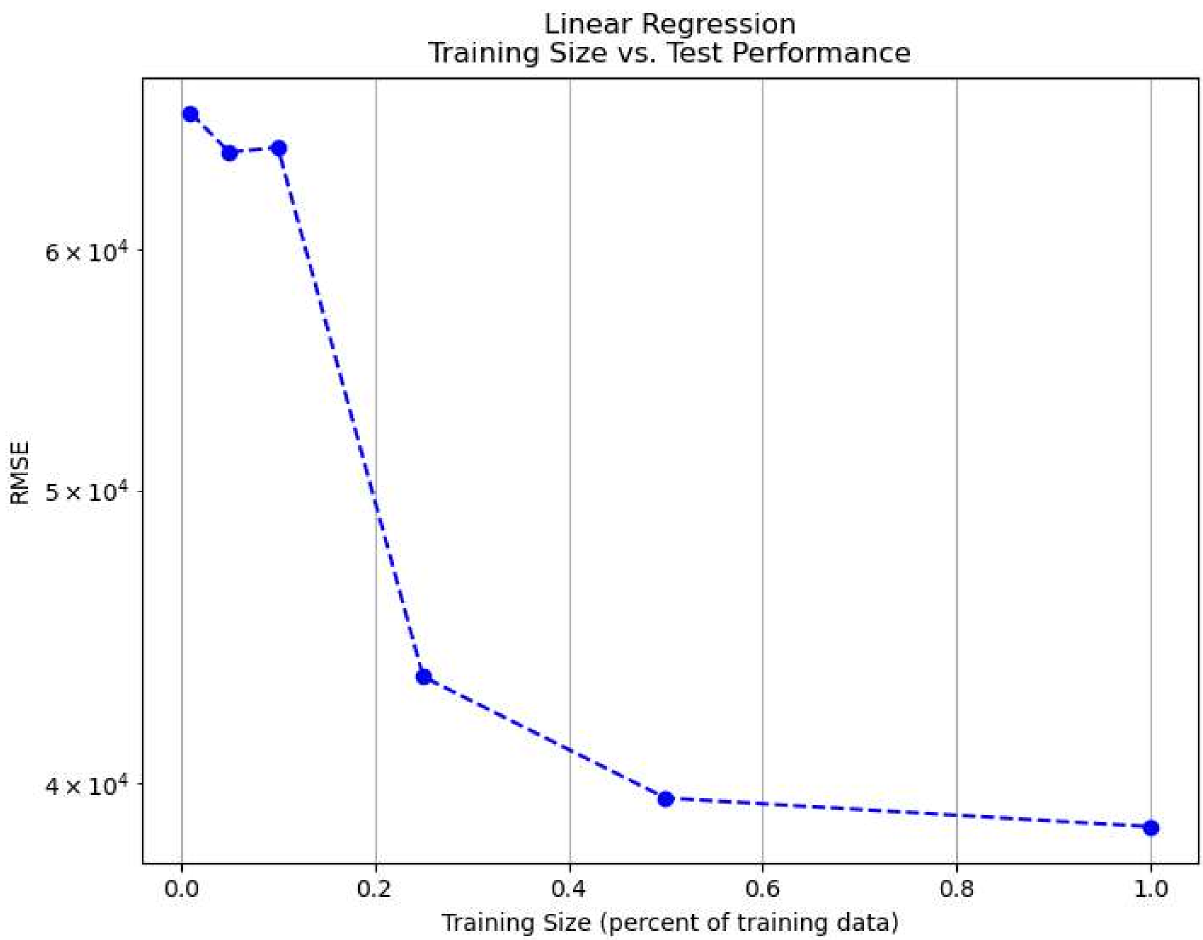
In [ ]:

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

rmse_list = []

# Evaluate models with different training sizes
training_sizes = [0.01, 0.05, 0.1, 0.25, 0.5]
for size in training_sizes:
    best_num_features = features[best_features_index]

    X_train_small, _, y_train_small, _  = train_test_split(train_X_stand, train_Y,
    feature_selection = SelectKBest(score_func=f_regression, k=best_num_features)
    feature_selection.fit(X_train_small, y_train_small.flatten())

    train_X_best_features = feature_selection.transform(X_train_small)
    test_X_best_features  = feature_selection.transform(test_X_stand)

    print(f"train shape {train_X_best_features.shape}, train y {y_train_small.shape
    lr = LinearRegression().fit(train_X_best_features, y_train_small.flatten())

    y_pred = lr.predict(test_X_best_features)

    print(f"For train size {size}, number of features: {lr.n_features_in_}")
    # np.savetxt(f"linear_regression_output/coef_{size}.txt", np.round(model.coef_,
    # print(f"model y-intercet: {model.intercept_}")
    y_pred = lr.predict(test_X_best_features)
    rmse = np.sqrt(mean_squared_error(test_Y, y_pred))
```

```
    rmse_list.append(rmse)

#100% of data. Do it apart from above since cannot split 100% of data
best_num_features = features[best_features_index]

feature_selection = SelectKBest(score_func=f_regression, k=best_num_features)
feature_selection.fit(train_X_stand, train_Y.flatten())

train_X_best_features = feature_selection.transform(train_X_stand)
test_X_best_features  = feature_selection.transform(test_X_stand)

lr = LinearRegression().fit(train_X_best_features, train_Y.flatten())

y_pred = lr.predict(test_X_best_features)

print(f"For train size {size}, number of features: {lr.n_features_in_}")
# np.savetxt(f"linear_regression_output/coef_{size}.txt", np.round(model.coef_, 1),
# print(f"model y-intercet: {model.intercept_}")
y_pred = lr.predict(test_X_best_features)
rmse = np.sqrt(mean_squared_error(test_Y, y_pred))
rmse_list.append(rmse)
training_sizes.append(1)        #add the 100%

# Create a plot
plt.figure(figsize=(8, 6))
plt.plot(training_sizes, rmse_list, marker="o", linestyle="--", color="b")
plt.yscale('log')
plt.xlabel("Training Size (percent of training data)")
plt.ylabel("RMSE")
plt.title("Linear Regression\nTraining Size vs. Test Performance")
plt.grid(True)
plt.show()
```

```
train shape (10, 72), train y (10, 1)
For train size 0.01, number of features: 72
train shape (51, 72), train y (51, 1)
For train size 0.05, number of features: 72
train shape (102, 72), train y (102, 1)
For train size 0.1, number of features: 72
train shape (255, 72), train y (255, 1)
For train size 0.25, number of features: 72
train shape (510, 72), train y (510, 1)
For train size 0.5, number of features: 72
For train size 0.5, number of features: 72
```

```
/home/noel/anaconda3/envs/cs441_project/lib/python3.12/site-packages/sklearn/feature
_selection/_univariate_selection.py:379: RuntimeWarning: invalid value encountered i
n sqrt
  X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
/home/noel/anaconda3/envs/cs441_project/lib/python3.12/site-packages/sklearn/feature
_selection/_univariate_selection.py:379: RuntimeWarning: invalid value encountered i
n sqrt
  X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
/home/noel/anaconda3/envs/cs441_project/lib/python3.12/site-packages/sklearn/feature
_selection/_univariate_selection.py:379: RuntimeWarning: invalid value encountered i
n sqrt
  X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
/home/noel/anaconda3/envs/cs441_project/lib/python3.12/site-packages/sklearn/feature
_selection/_univariate_selection.py:379: RuntimeWarning: invalid value encountered i
n sqrt
  X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
/home/noel/anaconda3/envs/cs441_project/lib/python3.12/site-packages/sklearn/feature
_selection/_univariate_selection.py:379: RuntimeWarning: invalid value encountered i
n sqrt
  X_norms = np.sqrt(row_norms(X.T, squared=True) - n_samples * X_means**2)
```



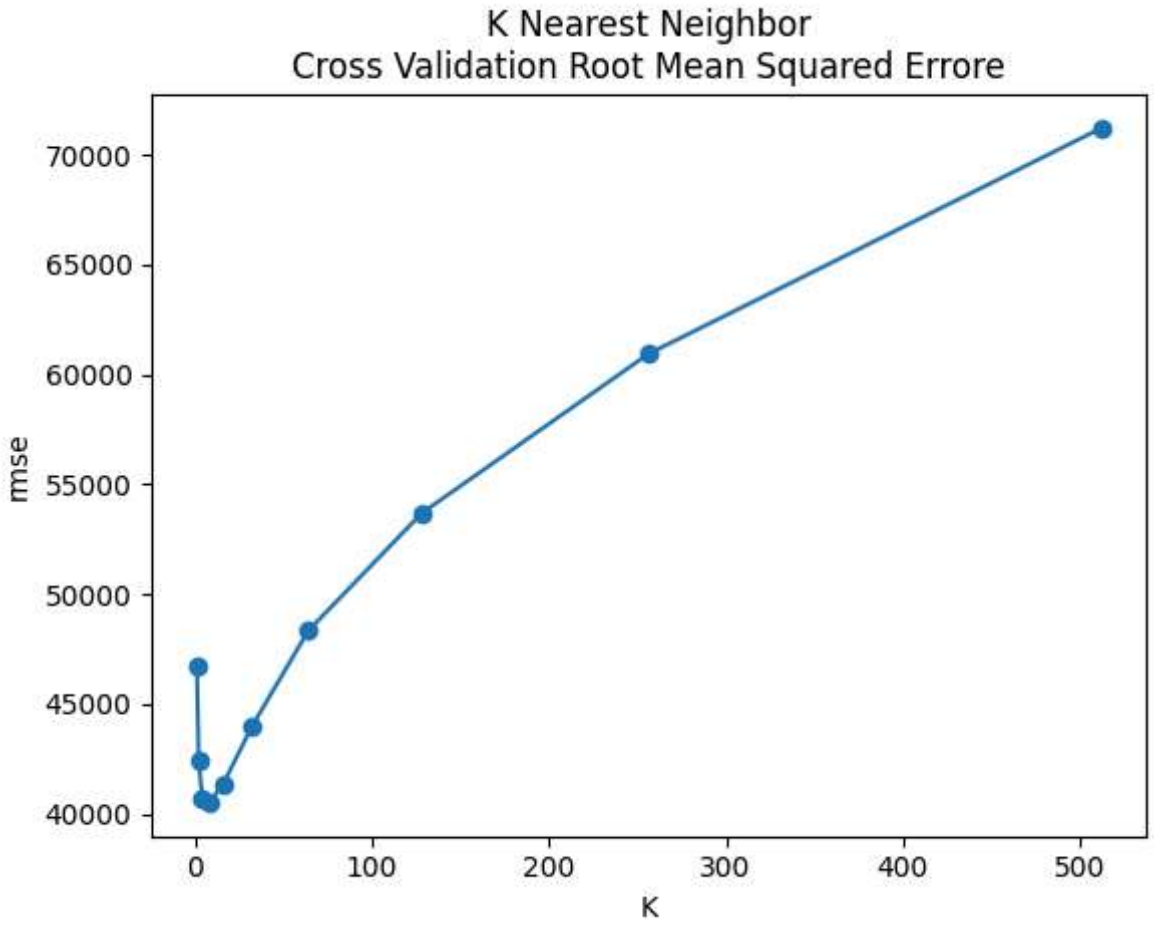
In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import warnings
import matplotlib.pyplot as plt
from sklearn.model_selection import  cross_val_score
```

```
def Forest(X_train, Y_train, X_val, Y_val, num):
  forest = RandomForestRegressor(n_estimators=num)
  forest.fit(X_train, Y_train)
  Y_pred = forest.predict(X_val)
  rmse = mean_squared_error(Y_val, Y_pred, squared=False)
  print("number of trees:", forest.n_estimators)
  print("max features:", forest.max_features)
  print("depth:", forest.max_depth)
  return rmse, Y_pred, forest
```

In [ ]:

```
K_set = []
rmse_set = []
# for K in [1]:
scores = []
for K in [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]:
  preds = []
  knn = KNeighborsRegressor(n_neighbors=K)
  knn.fit(train_X_stand, train_Y)
  score = cross_val_score(knn, train_X_stand, train_Y, cv=5, scoring='neg_root_mean
  scores.append(np.mean(score)*-1)
  # Y_pred = knn.predict(train_X_stand)
  print("KNN Model Parameters:", knn.get_params())
  K_set.append(K)

print(scores)
plt.plot(K_set, scores, marker='o')
plt.xlabel('K')
plt.ylabel('rmse')
plt.title('K Nearest Neighbor\nCross Validation Root Mean Squared Errore')
plt.show()
```

```
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 1, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 2, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 4, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 8, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 16, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 32, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 64, 'p': 2, 'weights': 'unifor
m'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 128, 'p': 2, 'weights': 'unifo
rm'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 256, 'p': 2, 'weights': 'unifo
rm'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 512, 'p': 2, 'weights': 'unifo
rm'}
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 1024, 'p': 2, 'weights': 'unif
orm'}
[46724.34703320634, 42423.71162625196, 40721.406389332275, 40512.175677312334, 4136
8.380337374554, 43963.4898235289, 48335.61287144683, 53680.99746164781, 60932.058140
68015, 71217.31102763559, nan]
```

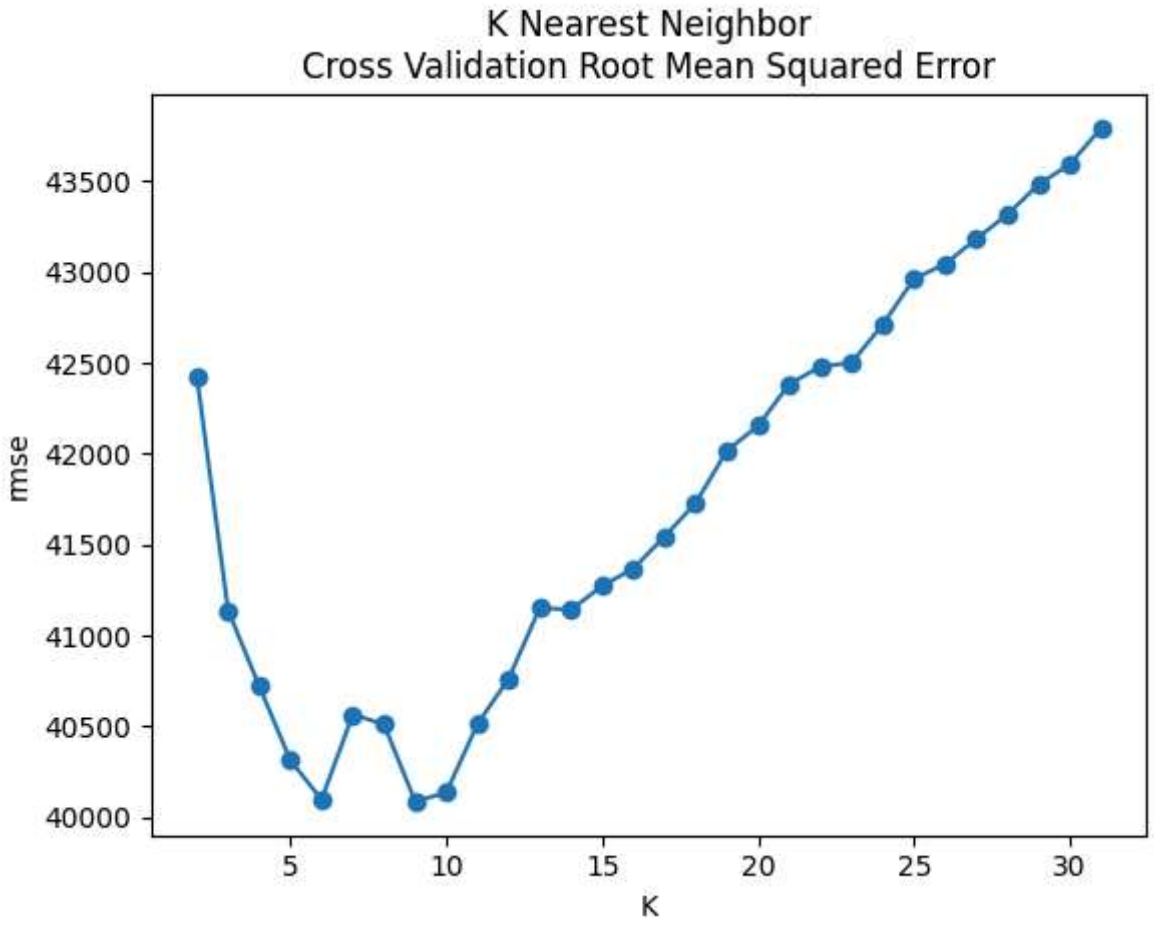


In [ ]:
```
K_set = []
rmse_set = []
# for K in [1]:
scores = []
for K in range(2, 32):
  preds = []
  knn = KNeighborsRegressor(n_neighbors=K)
  knn.fit(train_X_stand, train_Y)
  score = cross_val_score(knn, train_X_stand, train_Y, cv=5, scoring='neg_root_mean
  scores.append(np.mean(score)*-1)
  # Y_pred = knn.predict(train_X_stand)
  print("KNN Model Parameters:", knn.get_params())
  print(f"K={K},accuracy={np.mean(score)}")
  K_set.append(K)

print(scores)
print(np.argmin(scores))
plt.plot(K_set, scores, marker='o')
plt.xlabel('K')
plt.ylabel('rmse')
plt.title('K Nearest Neighbor\nCross Validation Root Mean Squared Error')
plt.show()
```

```
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 2, 'p': 2, 'weights': 'unifor
m'}
K=2,accuracy=-42423.71162625196
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 3, 'p': 2, 'weights': 'unifor
m'}
K=3,accuracy=-41138.14708977206
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 4, 'p': 2, 'weights': 'unifor
m'}
K=4,accuracy=-40721.406389332275
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'unifor
m'}
K=5,accuracy=-40312.01373913968
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 6, 'p': 2, 'weights': 'unifor
m'}
K=6,accuracy=-40098.1753698371
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 7, 'p': 2, 'weights': 'unifor
m'}
K=7,accuracy=-40565.180371926595
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 8, 'p': 2, 'weights': 'unifor
m'}
K=8,accuracy=-40512.175677312334
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 9, 'p': 2, 'weights': 'unifor
m'}
K=9,accuracy=-40086.699901112515
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 10, 'p': 2, 'weights': 'unifor
m'}
K=10,accuracy=-40136.73763055567
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 11, 'p': 2, 'weights': 'unifor
m'}
K=11,accuracy=-40513.82916374094
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 12, 'p': 2, 'weights': 'unifor
m'}
K=12,accuracy=-40763.32423747037
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 13, 'p': 2, 'weights': 'unifor
m'}
K=13,accuracy=-41152.89532067507
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 14, 'p': 2, 'weights': 'unifor
m'}
K=14,accuracy=-41141.7628993081
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 15, 'p': 2, 'weights': 'unifor
m'}
K=15,accuracy=-41273.47472427724
```

```
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 16, 'p': 2, 'weights': 'unifor
m'}
K=16,accuracy=-41368.380337374554
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 17, 'p': 2, 'weights': 'unifor
m'}
K=17,accuracy=-41544.758196524665
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 18, 'p': 2, 'weights': 'unifor
m'}
K=18,accuracy=-41728.96015226352
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 19, 'p': 2, 'weights': 'unifor
m'}
K=19,accuracy=-42017.17497100057
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 20, 'p': 2, 'weights': 'unifor
m'}
K=20,accuracy=-42153.557114611904
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 21, 'p': 2, 'weights': 'unifor
m'}
K=21,accuracy=-42383.18831270566
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 22, 'p': 2, 'weights': 'unifor
m'}
K=22,accuracy=-42479.15143453049
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 23, 'p': 2, 'weights': 'unifor
m'}
K=23,accuracy=-42501.1166565336
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 24, 'p': 2, 'weights': 'unifor
m'}
K=24,accuracy=-42710.782310692695
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 25, 'p': 2, 'weights': 'unifor
m'}
K=25,accuracy=-42962.16171203875
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 26, 'p': 2, 'weights': 'unifor
m'}
K=26,accuracy=-43043.68479259903
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 27, 'p': 2, 'weights': 'unifor
m'}
K=27,accuracy=-43182.61432156182
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 28, 'p': 2, 'weights': 'unifor
m'}
K=28,accuracy=-43316.45856976545
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 29, 'p': 2, 'weights': 'unifor
m'}
K=29,accuracy=-43482.42986935028
```

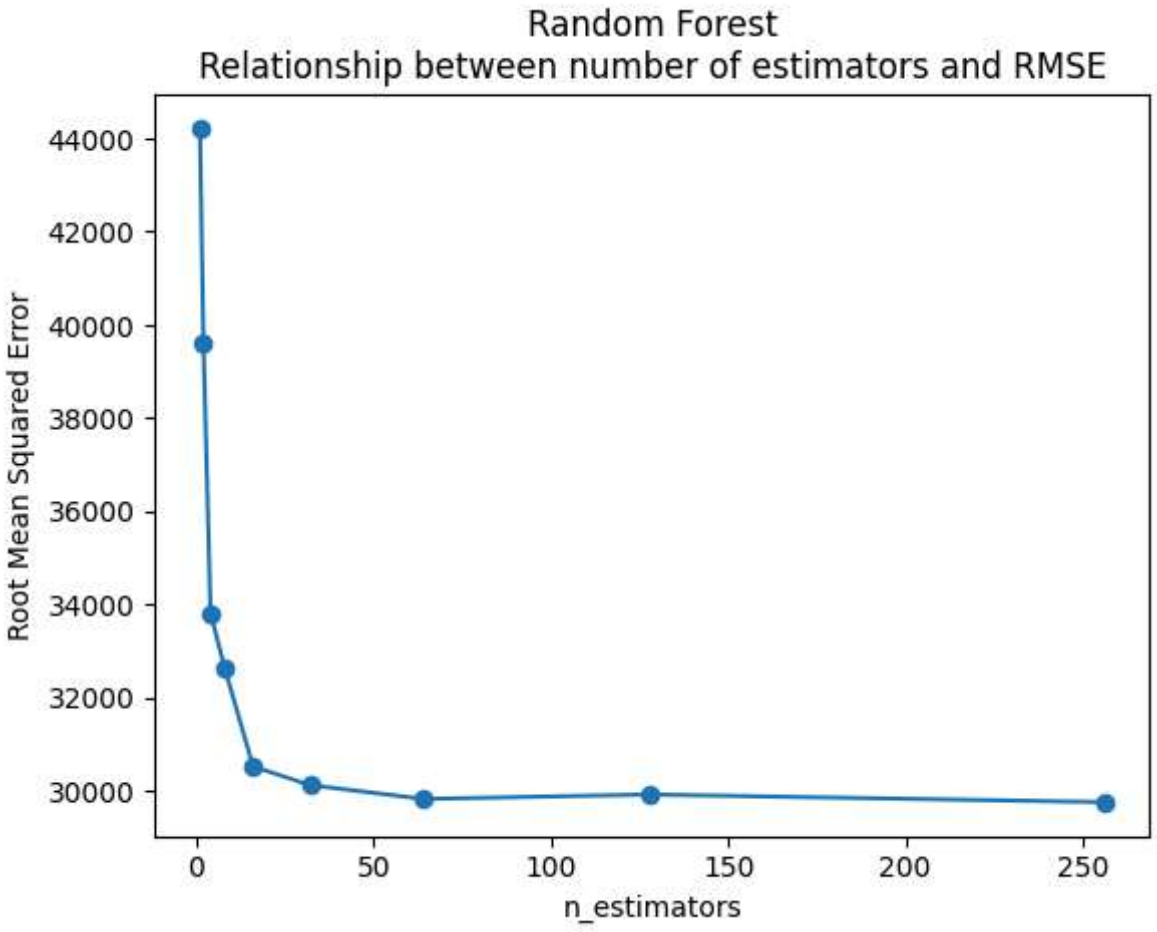
```
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 30, 'p': 2, 'weights': 'unifor
m'}
K=30,accuracy=-43592.539863578684
KNN Model Parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',
'metric_params': None, 'n_jobs': None, 'n_neighbors': 31, 'p': 2, 'weights': 'unifor
m'}
K=31,accuracy=-43791.23238804847
[42423.71162625196, 41138.14708977206, 40721.406389332275, 40312.01373913968, 40098.
1753698371, 40565.180371926595, 40512.175677312334, 40086.699901112515, 40136.737630
55567, 40513.82916374094, 40763.32423747037, 41152.89532067507, 41141.7628993081, 41
273.47472427724, 41368.380337374554, 41544.758196524665, 41728.96015226352, 42017.17
497100057, 42153.557114611904, 42383.18831270566, 42479.15143453049, 42501.116656533
6, 42710.782310692695, 42962.16171203875, 43043.68479259903, 43182.61432156182, 4331
6.45856976545, 43482.42986935028, 43592.539863578684, 43791.23238804847]
7
```



In [ ]:
```python
K_set = []
rmse_set = []
scores = []
for n_estimators in [1, 2, 4, 8, 16, 32, 64, 128, 256]:
  preds = []
  knn = RandomForestRegressor(n_estimators=n_estimators)
  knn.fit(train_X_stand, train_Y)
  score = cross_val_score(knn, train_X_stand, train_Y, cv=5, scoring='neg_root_mean
  scores.append(np.mean(score)*-1)
  # Y_pred = knn.predict(train_X_stand)
  print(f"n_estimators={n_estimators},rmse={np.mean(score)*-1}")
```

```
  K_set.append(n_estimators)

print(scores)
print(np.argmin(scores))
plt.plot(K_set, scores, marker='o')
plt.xlabel('n_estimators')
plt.ylabel('Root Mean Squared Error')
plt.title('Random Forest\nRelationship between number of estimators and RMSE')
plt.show()
```

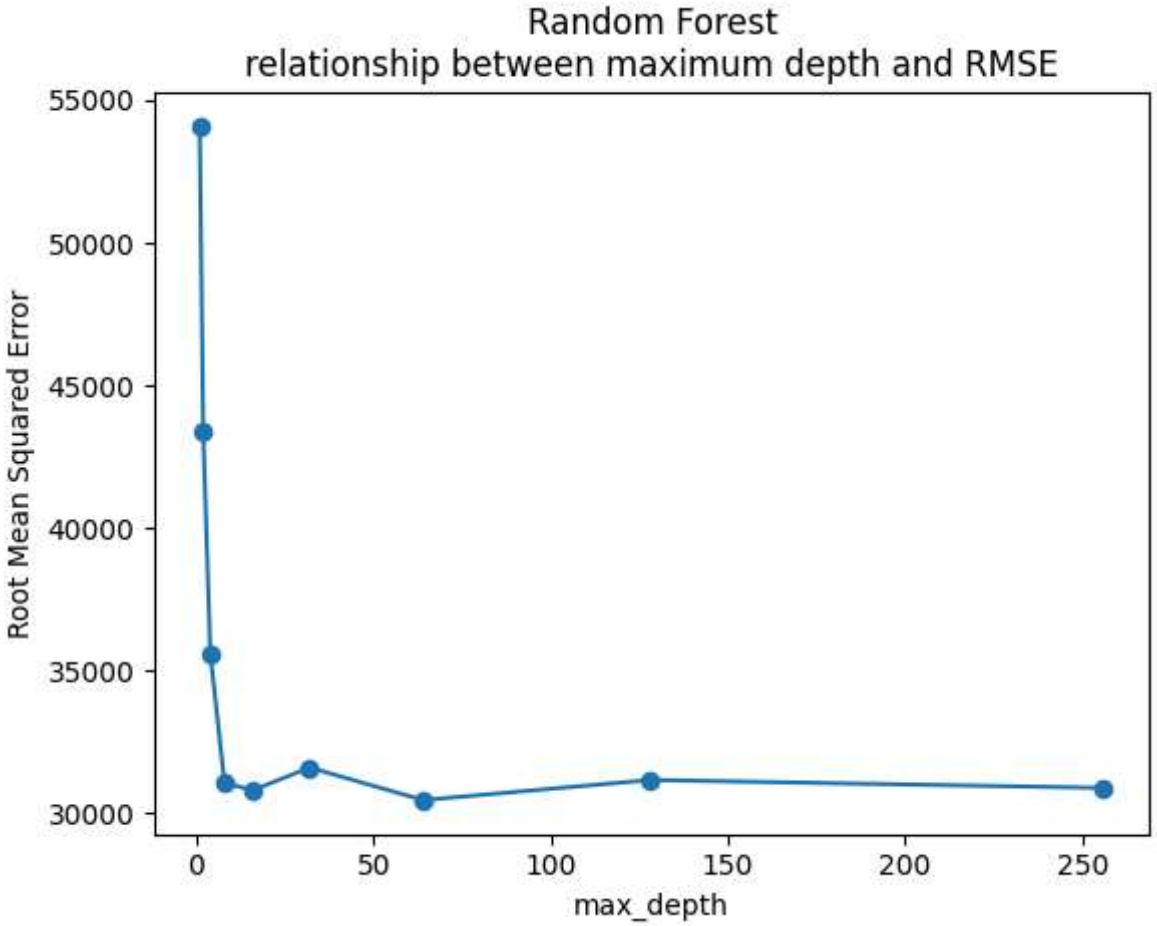
```
n_estimators=1,rmse=44221.49814288576
n_estimators=2,rmse=39587.43376716284
n_estimators=4,rmse=33816.66691302977
n_estimators=8,rmse=32627.86800838616
n_estimators=16,rmse=30523.52526483578
n_estimators=32,rmse=30123.859222654584
n_estimators=64,rmse=29826.215622731997
n_estimators=128,rmse=29921.496869006747
n_estimators=256,rmse=29758.63845371455
[44221.49814288576, 39587.43376716284, 33816.66691302977, 32627.86800838616, 30523.5
2526483578, 30123.859222654584, 29826.215622731997, 29921.496869006747, 29758.638453
71455]
8
```



In [ ]:
```
K_set = []
rmse_set = []
scores = []
for max_depth in [1, 2, 4, 8, 16, 32, 64, 128, 256]:
```

```
  preds = []
  knn = RandomForestRegressor(n_estimators=16, max_depth=max_depth)
  knn.fit(train_X_stand, train_Y)
  score = cross_val_score(knn, train_X_stand, train_Y, cv=5, scoring='neg_root_mean
  scores.append(np.mean(score)*-1)
  # Y_pred = knn.predict(train_X_stand)
  print(f"max_depth={max_depth},rmse={np.mean(score)*-1}")
  K_set.append(max_depth)

print(scores)
print(np.argmin(scores))
plt.plot(K_set, scores, marker='o')
plt.xlabel('max_depth')
plt.ylabel('Root Mean Squared Error')
plt.title('Random Forest\nrelationship between maximum depth and RMSE')
plt.show()
```

```
max_depth=1,rmse=54084.00310793494
max_depth=2,rmse=43364.5485430894
max_depth=4,rmse=35606.80552494674
max_depth=8,rmse=31064.008298681107
max_depth=16,rmse=30799.627475199337
max_depth=32,rmse=31596.528608938883
max_depth=64,rmse=30455.357281425542
max_depth=128,rmse=31160.441614643845
max_depth=256,rmse=30883.212971857727
[54084.00310793494, 43364.5485430894, 35606.80552494674, 31064.008298681107, 30799.6
27475199337, 31596.528608938883, 30455.357281425542, 31160.441614643845, 30883.21297
1857727]
6
```

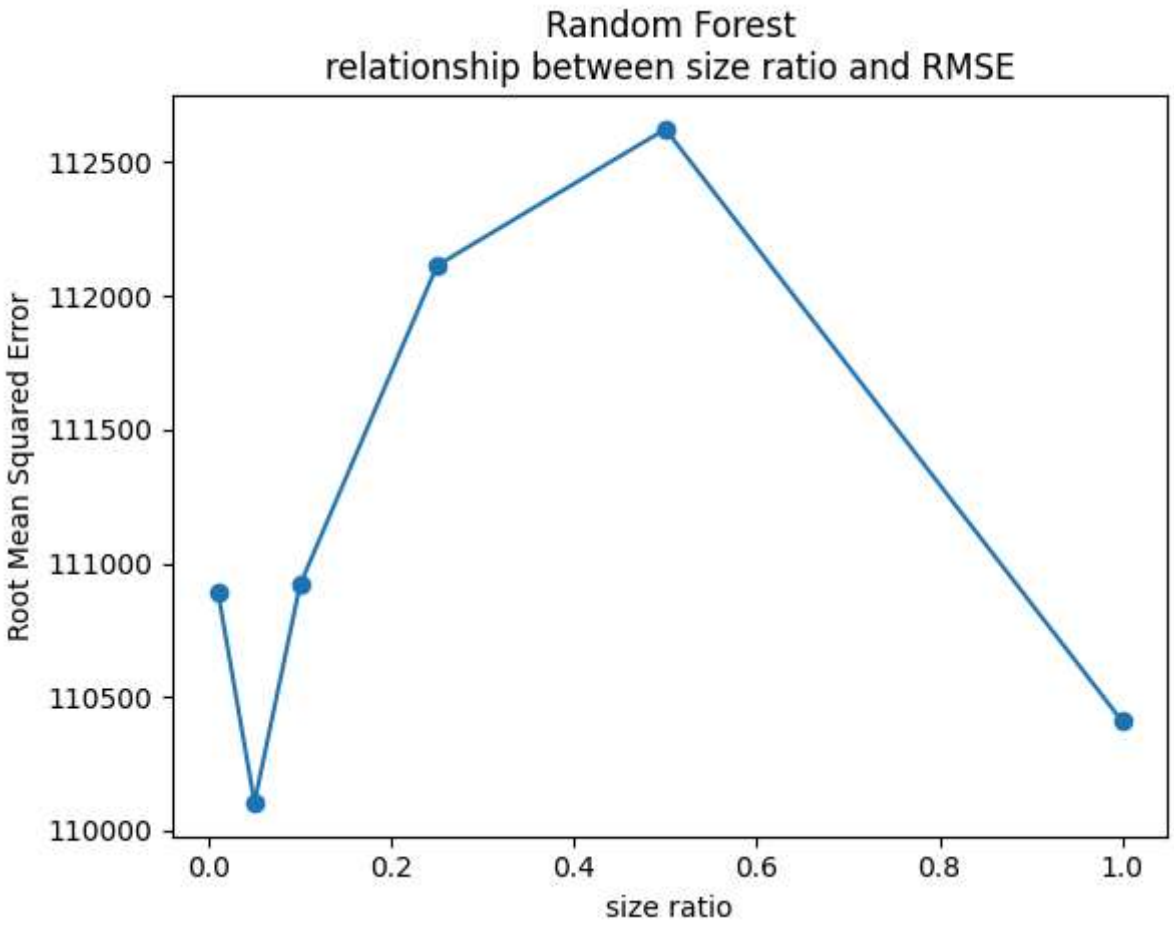


In [ ]:

```
rmses = []
sizeset = [0.01, 0.05, 0.1, 0.25, 0.5]
for size in sizeset:
  print(f"size ratio: {size}")
  size = size*train_X_stand.shape[0]
  size = int(size)
  rf = RandomForestRegressor(n_estimators=16, max_depth=64)
  # rf = RandomForestRegressor(n_estimators=2, max_depth=max_depth)
  rf.fit(train_X_stand[size:], train_Y[size:])
  pred_test = rf.predict(test_X_stand)
  rmse = np.sqrt(np.mean((pred_test-test_Y)**2))
  print(f"rmse: {rmse}")
  rmses.append(rmse)


print(f"size ratio: {1}")
rf = RandomForestRegressor(n_estimators=16, max_depth=64)
rf.fit(train_X_stand, train_Y)
pred_test = rf.predict(test_X_stand)
rmse = np.sqrt(np.mean((pred_test-test_Y)**2))
rmses.append(rmse)
print(f"rmse: {rmse}")
sizeset.append(1)
print(rmses)
plt.plot(sizeset, rmses, marker='o')
plt.xlabel('size ratio')
plt.ylabel('Root Mean Squared Error')
```

```
plt.title('Random Forest\nrelationship between size ratio and RMSE')
plt.show()
```

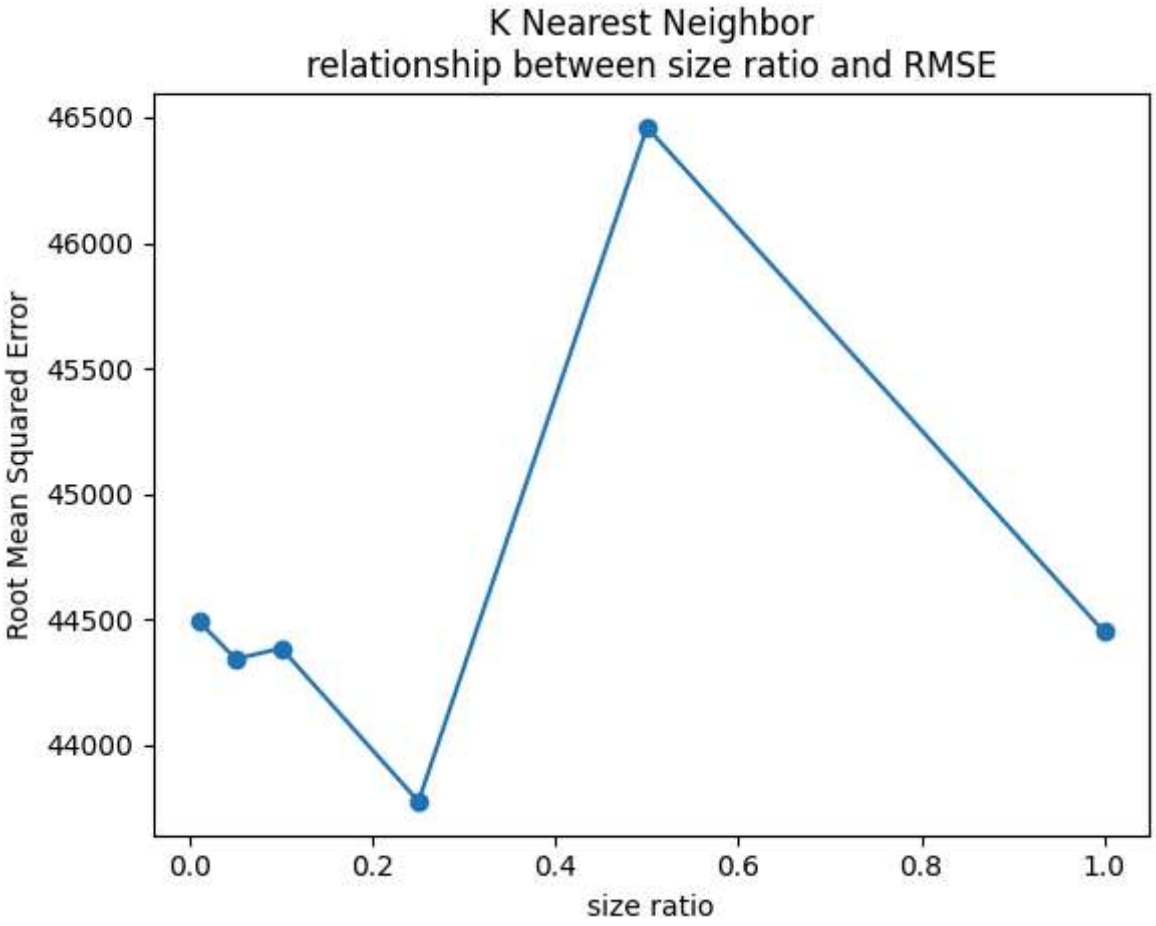
```
size ratio: 0.01
rmse: 110893.50873223983
size ratio: 0.05
rmse: 110105.36514674137
size ratio: 0.1
rmse: 110920.531620701
size ratio: 0.25
rmse: 112113.31285714186
size ratio: 0.5
rmse: 112622.94089708012
size ratio: 1
rmse: 110409.67381222571
[110893.50873223983, 110105.36514674137, 110920.531620701, 112113.31285714186, 11262
2.94089708012, 110409.67381222571]
```



In [ ]:
```
rmses = []
sizeset = [0.01, 0.05, 0.1, 0.25, 0.5]
for size in sizeset:
  print(f"size ratio: {size}")
  size = size*train_X_stand.shape[0]
  size = int(size)
  rf = KNeighborsRegressor(n_neighbors=9)
  # rf = RandomForestRegressor(n_estimators=2, max_depth=max_depth)
  rf.fit(train_X_stand[size:], train_Y[size:])
  pred_test = rf.predict(test_X_stand)
```

```
  rmse = np.sqrt(np.mean((pred_test-test_Y)**2))
  print(f"rmse: {rmse}")
  rmses.append(rmse)


print(f"size ratio: {1}")
rf = KNeighborsRegressor(n_neighbors=9)
rf.fit(train_X_stand, train_Y)
pred_test = rf.predict(test_X_stand)
rmse = np.sqrt(np.mean((pred_test-test_Y)**2))
rmses.append(rmse)
print(f"rmse: {rmse}")
sizeset.append(1)
print(rmses)
plt.plot(sizeset, rmses, marker='o')
plt.xlabel('size ratio')
plt.ylabel('Root Mean Squared Error')
plt.title('K Nearest Neighbor\nrelationship between size ratio and RMSE')
plt.show()
```

```
size ratio: 0.01
rmse: 44491.88167906422
size ratio: 0.05
rmse: 44343.43385283737
size ratio: 0.1
rmse: 44385.74119841336
size ratio: 0.25
rmse: 43776.10358243226
size ratio: 0.5
rmse: 46460.39419034753
size ratio: 1
rmse: 44451.44428498494
[44491.88167906422, 44343.43385283737, 44385.74119841336, 43776.10358243226, 46460.3
9419034753, 44451.44428498494]
```


K Nearest Neighbor
relationship between size ratio and RMSE
Root Mean Squared Error
46500
46000
45500
45000
44500
44000
0.0
0.2
0.4
0.6
0.8
1.0
size ratio