In [83]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
import warnings
```

This dataset is from UC Irvine and covers various information on imported cars taken from 1985. It was going to be used for a competition on risk assessment, but the competition was cancelled. However, we will be using it to predict the variable of interest which is mpg (miles per gallon). We would like to see which variables most contribute to the gas mileage of the imported cars. To do this we use regression machine learning techniques. First we performed linear regression and then we performed k nearest neighbors regression. This analysis will help determine which features can be added or removed from cars to affect their gas mileage.

Here is the link to the dataset we used in the analysis:
https://archive.ics.uci.edu/dataset/10/automobile

```
The dataset contains 26 features:

   - "symboling", #1
   - "normalized-losses" #2
   - "make" #3
   - "fuel-type" #4
   - "aspiration" #5
   - "num_of_doors" #6
   - "body-style" #7
   - "drive-wheels" #8
   - "engine-location" #9
   - "wheel-base" #10
   - "length" #11
   - "width  #12
   - "height" #13
   - "curbweight" #14
   - "enginetype" #15
   - "numcylinders" #16
   - "engine-size" #17
   - "fuel-system" #18
   - "bore" #19
   - "stroke" #20
   - "compression-ratio" #21
   - "horse-power" #22
   - "peak-rpm" #23
   - "city-mpg" #24
   - "highway-mpg" #25
   - "price" #26
```

There were 205 records per feature before we perfomed data cleaning.

In the following cell, we dropped the normalized losses column because it had an excessive amount of null values. Next, we replaced the "?"s with n/a's before dropping all records with n/a values.

Most importantly, we found the mean of city-mpg (which measured miles per gallon for cars in cities) and highway_mpg (which measured miles per gallon for cars on the highway) to create a field called "mpg".

In [89]:
```
# read in car data
df = pd.read_csv('imports-85.csv')

# named columns
df.columns = ["symboling", #1
              "normalized-losses", #2
              "make", #3
              "fuel-type", #4
              "aspiration", #5
              "num_of_doors", #6
              "body-style", #7
              "drive-wheels", #8
              "engine-location", #9
              "wheel-base", #10
              "length", #11
              "width", #12
              "height", #13
              "curbweight", #14
              "enginetype", #15
              "numcylinders", #16
              "engine-size", #17
              "fuel-system", #18
              "bore", #19
              "stroke", #20
              "compression-ratio", #21
              "horse-power", #22
              "peak-rpm", #23
              "city-mpg", #24
              "highway-mpg", #25
              "price"] #26

# dropped normalized losses column
df_2 = df.drop(columns=df.columns[1])

# dropped remaining rows with ? values
df_3 = df_2.replace("?", np.nan)

df_3 = df_3.dropna()

df_4 = df_3
df_4["mpg"] = df[["city-mpg", "highway-mpg"]].mean(axis=1) #averaging the two va
#df_4 = df_4.sort_values(by='peak-rpm') #sorting the dataset by rpm
```

In [91]:
```
df_4.head()
```

Out[91]:

	symboling	make	fuel-type	aspiration	num_of_doors	body-style	drive-wheels	engine-location	w
0	3	alfa-romero	gas	std	two	convertible	rwd	front	
1	1	alfa-romero	gas	std	two	hatchback	rwd	front	
2	2	audi	gas	std	four	sedan	fwd	front	
3	2	audi	gas	std	four	sedan	4wd	front	
4	2	audi	gas	std	two	sedan	fwd	front	

5 rows × 26 columns

(Above is the head of the cleaned dataframe)

For this section we used linear regression to determine which factors affect mpg the most, and we split the training and test data to allow us to test our predictions of which cars have the best mpg :

The comparisons we will make in this section are:

mpg vs Peak rpm

mpg vs Engine type

mpg vs RPM

mpg vs horse-power

mpg vs fuel-system

mpg vs fuel-type

mpg vs numcylinders

mpg vs car make

In [93]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.metrics import mean_squared_error, r2_score
```

Next, we dropped the make and price columns before performing our linear regression. In order to remove unecessary columns in the dataset, and help with the model's evaluation, we dropped the make and price column. We changed the values categorical columns into integers, and put it into a panda dummies dataframe to make it easier to work with.

In [95]:
```
data_frame = df_4.drop(columns=['make', 'price'])
convert_colums = ['wheel-base', 'length', 'width', 'height', 'curbweight', 'numc
```

```
for col in convert_colums:
    pd.to_numeric(data_frame[col], errors='coerce')
mtw_categorical_cols = data_frame.select_dtypes(include=['object']).columns
mtw_df = pd.get_dummies(data_frame, columns=mtw_categorical_cols, drop_first=Tru
```

In [97]:
```
mtw_df.head()
```

Out[97]:

	symboling	wheel-base	length	width	height	curbweight	engine-size	compression-ratio	city-mpg
0	3	88.6	168.8	64.1	48.8	2548	130	9.0	21
1	1	94.5	171.2	65.5	52.4	2823	152	9.0	19
2	2	99.8	176.6	66.2	54.3	2337	109	10.0	24
3	2	99.4	176.6	66.4	54.3	2824	136	8.0	18
4	2	99.8	177.3	66.3	53.1	2507	136	8.5	19

5 rows × 183 columns

Next, we split the data into training and testing sets.

In [99]:
```
#Split the data into 30% testing 70% training
mtw_X = mtw_df.drop(columns=['mpg'])
mtw_y = mtw_df['mpg']
mtw_X_train, mtw_X_test, mtw_y_train, mtw_y_test = train_test_split(mtw_X, mtw_y
```

We then scaled and transformed the data using a StandardScaler from sklearn.

In [101…
```
mtw_scaler = StandardScaler()
mtw_X_train_scaled = pd.DataFrame(mtw_scaler.fit_transform(mtw_X_train), columns
mtw_X_test_scaled = pd.DataFrame(mtw_scaler.transform(mtw_X_test), columns=mtw_X
```

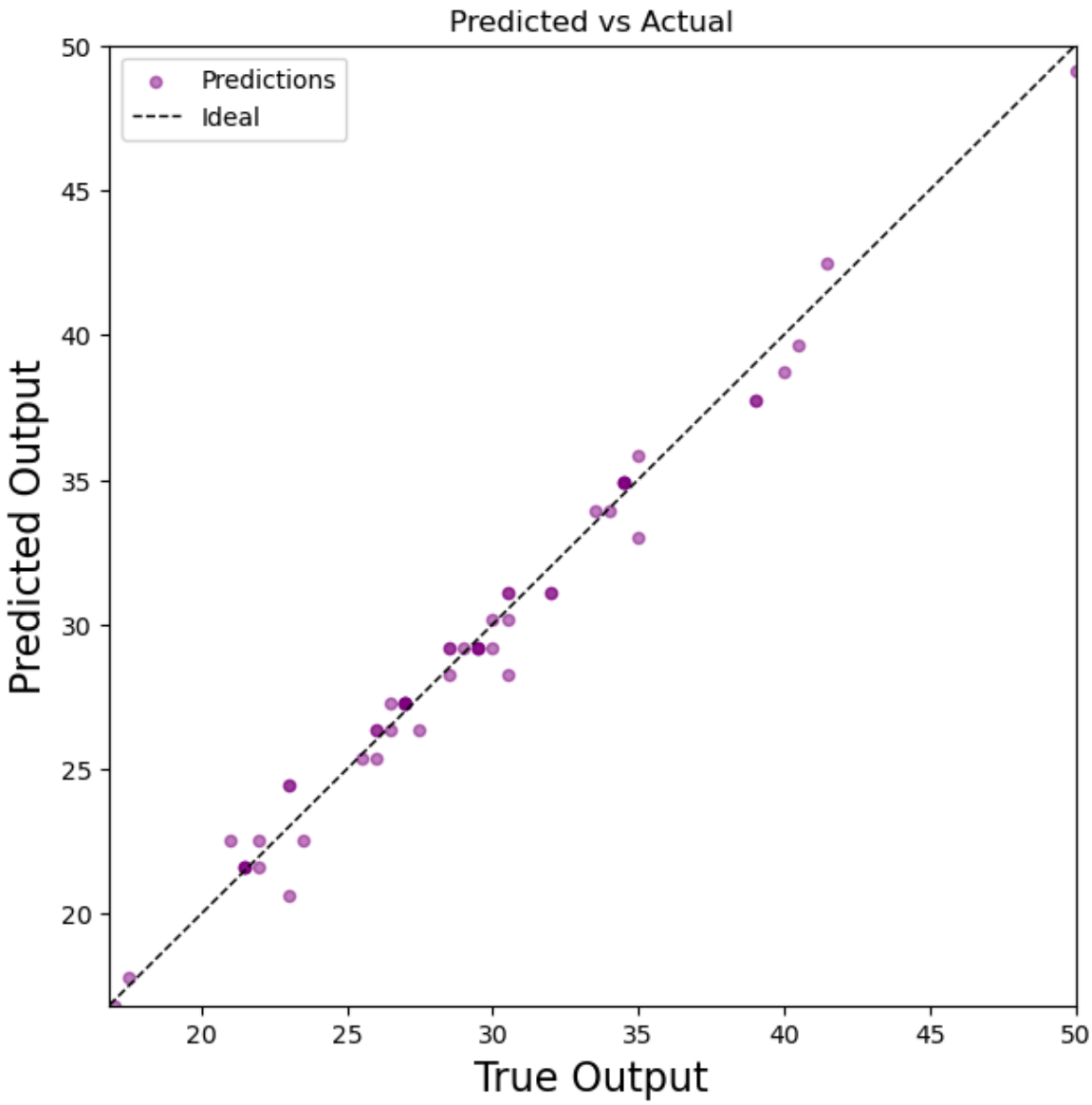
Finally, we set up the linear regression models. For this part we set up the linear regression model, we begin by defining the number of features, then find the RFE. Adapt the RFE to fit the scaled data then transform the testing data using the transform function. We then store the names of the columns into an array. Then set up the linear regression line according to the train data.

In [103…
```
mtw_model = LinearRegression()
mtw_n_features = 1
mtw_rfe = RFE(mtw_model, n_features_to_select=mtw_n_features)
mtw_X_train_selected = mtw_rfe.fit_transform(mtw_X_train_scaled, mtw_y_train) #F
mtw_X_test_selected = mtw_rfe.transform(mtw_X_test_scaled) #Transforms the testi

mtw_selected_features = mtw_X_train.columns[mtw_rfe.support_].tolist() #stores t

mtw_model.fit(mtw_X_train_selected, mtw_y_train) #Sets up the linear regression

mtw_y_pred = mtw_model.predict(mtw_X_test_selected) #predict y values based off
mtw_rmse = np.sqrt(mean_squared_error(mtw_y_test, mtw_y_pred)) #calculates the R
mtw_r2 = r2_score(mtw_y_test, mtw_y_pred) #caculates the R-squared score
```

Plot for the graphs. We look at the actual vs predicted values on a scatter plot with the linear line.

In [105…

```
fig, ax1 = plt.subplots(1, 1, figsize=(7, 7))
ax1.scatter(mtw_y_test, mtw_y_pred, alpha=0.5, s=20, color='purple', label='Pred
mtw_min_val = min(min(mtw_y_test), min(mtw_y_pred))
mtw_max_val = max(max(mtw_y_test), max(mtw_y_pred))
ax1.plot([mtw_min_val, mtw_max_val], [mtw_min_val, mtw_max_val], linewidth=1, co
ax1.set_xlabel('True Output', fontsize=16)
ax1.set_ylabel('Predicted Output', fontsize=16)
ax1.set_xlim(mtw_min_val, mtw_max_val)
ax1.set_ylim(mtw_min_val, mtw_max_val)
ax1.set_title("Predicted vs Actual")
ax1.legend()
plt.show()
```



When we graphed it the data was all along the linear regression line. The model works amazingly. The data lines up perfectly with the linear regression line. The difference between the actual and predicted values was very small. We think this works very well as a model for predicting the mpg. It appears to be very accurate.

I then predicted the y values based off the test data, and then calculated the RMSE as well as the R-squared score.

In [107…
```
print(f"RMSE: {mtw_rmse}")
print(f"R-Squared Score: {mtw_r2}")
```

```
RMSE: 0.8257367003161341
R-Squared Score: 0.9822841307345045
```

The RMSE is .83 and the R-squared is .98. This further indicates that our model is very strong.

For our next section, we decided to use k nearest neighbors regression model. We thought that model would lend itself well to the mixture of categorical and numerical data. It required us to do some binning of the categorical data, but we think the results worked well.

In [ ]:
```
First we did imported more sklearn modules to utilize their useful functions.
```

In [109…
```
from sklearn.feature_selection import chi2, SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

We manipulated the dataframes some more here. We also calculated a feature called "volume" which is the width times the length times the height of the cars. We thought this metric would be useful because a larger volume usually indicates a heavier car which would presumably impact mpg.

In [113…
```
df_CM_test = pd.read_csv('imports-85.csv')

df_CM_test.columns = ["symboling", "normalized-losses", "make", "fuel-type", "as
              "body-style", "drive-wheels", "engine-location", "wheel-base", "le
              "height", "curbweight", "enginetype", "numcylinders", "engine-size
              "horse-power", "peak-rpm", "city-mpg", "highway-mpg", "price"]

df_1_CM = df_CM_test.replace("?", np.NaN)
df_1_CM = df_1_CM.dropna()

# dropped normalized losses column
df_1_CM = df_1_CM.drop(columns=df_CM_test.columns[2])


# average city-mpg and highway-mpg to put in new column called average_mpg

df_1_CM['average_mpg'] = (df_CM_test['city-mpg'] + df_CM_test['highway-mpg']) /
df_1_CM['volume'] = df_CM_test['length'] * df_CM_test['width']* df_CM_test['heig
```

Next we binned the data. We created bins, labels and split the mpg into different parts. We turned the fuel-types, aspiration, drive-wheels and numcylinders data into discrete numerical values. Then we chose the columns with the most relevant xdata: curb weiht,

engine size, number of cylinders, and horsepower. There were some warnings, but nothing that prevented the model from running or affected the code.

In [141…

```
warnings.filterwarnings("ignore", category=DeprecationWarning)

bins = [0, 25, 32, 52]
labels = ['low', 'middle', 'high']

df_1_CM['mpg_labels'] = pd.cut(df_1_CM['average_mpg'], bins, labels=labels)


df_1_CM.infer_objects(copy=False)
df_1_CM['fuel-type'].replace({'gas': 1, 'deisel': 2, 'diesel':2 }, inplace=True)

df_1_CM['aspiration'].replace({'std': 1, 'turbo': 2 }, inplace=True)
df_1_CM['drive-wheels'].replace({'fwd': 1, 'rwd': 2 , '4wd':4}, inplace=True)
df_1_CM['numcylinders'].replace({'two': 2, 'twelve': 12, 'three': 3, 'four':4, '

df_1_CM['mpg_labels'].value_counts(normalize=True)

#selected the columns with the most relevant xdata: curb weight, engine size, #
X_CM= df_1_CM.iloc[:,[2,3,6,8,12,14,15,17,18,19,20,21,24,26]]
X_CM = X_CM.astype(float)
col_names = X_CM.columns

y_CM= df_1_CM.iloc[:,27]
```

Next we again split the data into training and test data.

In [117…

```
X_train_CM, X_test_CM, y_train_CM, y_test_CM = train_test_split(X_CM, y_CM, test
```

We then standardized the data because k nearest neighbors performs better with scaled data. We subtract by the mean and then divide by the range. Lastly, we reshaped the data set using a melt function.

In [119…

```
data_CM = X_CM

data_std_CM = (data_CM - data_CM.mean())/(data_CM.max() - data_CM.min())
data_CM = pd.concat([data_std_CM,y_CM], axis=1)

# reshape the dataframe using melt()
data_CM = pd.melt(data_CM, id_vars = 'mpg_labels', var_name = 'features',value_n
```

K nearest neighbors suffers from the curse of dimensionality, and with only 192 records, we did not want to risk overfitting the data. So, we decided to use a smaller number of features. To accomplish this we used a chi-squared test from SelectKbest function to choose the 5 most predictive variables.

In [121…

```
ft_CM= SelectKBest(chi2, k = 5).fit(X_train_CM, y_train_CM)
cols_idxs = ft_CM.get_support(indices=True)
xall_CM = X_CM.iloc[:,cols_idxs]
```

The variables chosen are curbweight, engine-size, horse-power, price, and volume. By inspection, it seems that the price has no actual effect on the mpg, so the select function probably chose a less than ideal variable, so we removed it from the dataframe before

generating the model. Preliminary testing we did running the model with price showed that it actually lowered the R-squared and increased RMSE, so our choice to drop it was justified.

In [123…
```
xall_CM =  xall_CM.drop('price', axis=1)
```

Next we created a dataframe of input and output variables, imported more modules, and pre-allocated memory for the arrays we will use to find optimal k values for the model.

In [125…
```
#our dependent variable
yall_CM = df_1_CM.iloc[:,25]

# create data frame df that consists only of our input and output variables
df_best_CM = pd.concat([xall_CM, yall_CM], axis=1)#simple data frame for conveni

#importing necessary functions
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error # Import mean_squared_error from
import math

#pre-allocating memory for arrays
inrmse_values_CM = np.arange(0.0,47.0)
outrmse_values_CM = np.arange(0.0,47.0)
k_values_CM = np.arange(0,47)
```

We created a function that tests various k values for the prospective models and graphs the training data rmse and test data rmse values vs the k values used in the models.

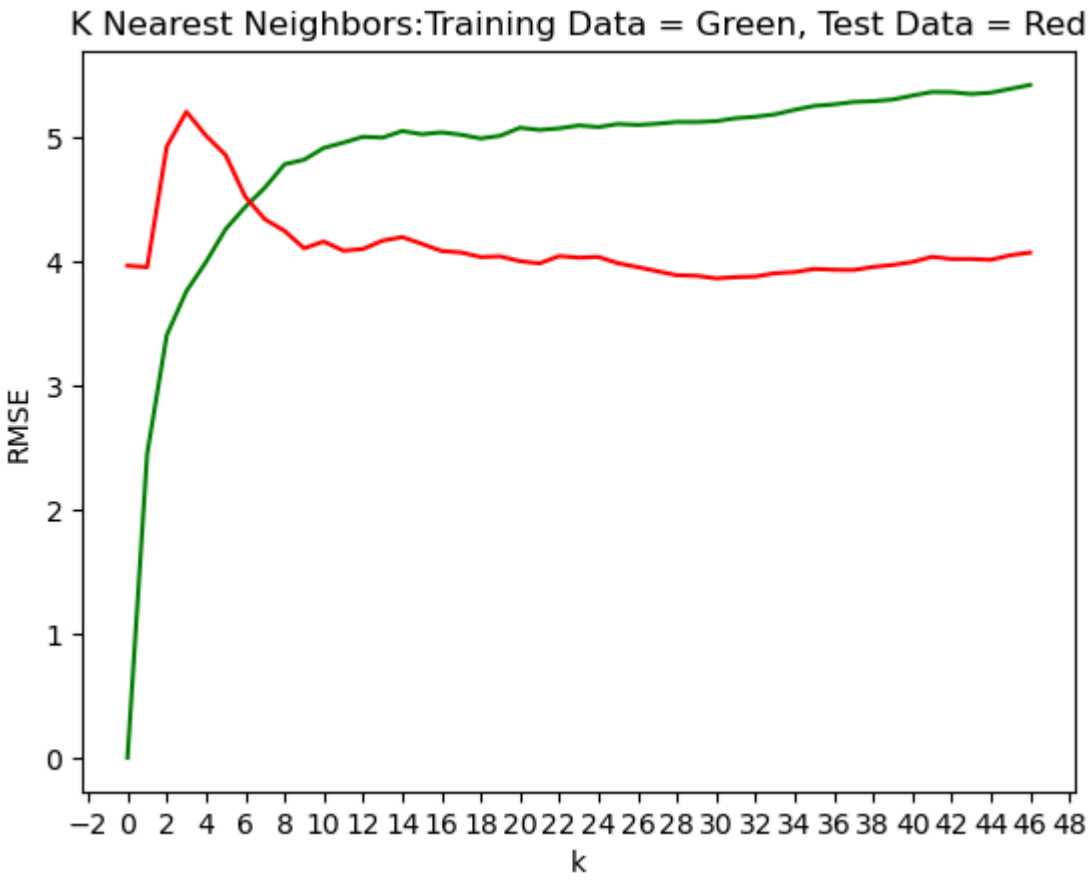
In [127…
```
#function for testing for optimal k's
def knn_optimal_k_test(xall_attempt_CM, scaled, plot):
    xall_train_CM, xall_test_CM, yall_train_CM, yall_test_CM = train_test_split(

    knn_CM = KNeighborsRegressor()


    #Fits and models k nearest neighbors based off various k values and calculat
    for i in k_values_CM:
        knn_CM = KNeighborsRegressor(n_neighbors=i+1)
        knn_CM.fit(xall_train_CM, yall_train_CM)

            # In-sample RMSE
        y_train_pred_CM = knn_CM.predict(xall_train_CM)
        inrmse_CM = np.sqrt(mean_squared_error(yall_train_CM, y_train_pred_CM))
        inrmse_values_CM[i] = inrmse_CM
            # Out-of-sample RMSE
        y_test_pred_CM = knn_CM.predict(xall_test_CM)
        outrmse_CM = np.sqrt(mean_squared_error(yall_test_CM, y_test_pred_CM))
        outrmse_values_CM[i] = outrmse_CM
    #plots RMSE values for train and test data vs various k's
    if plot == True:
        plt.plot(k_values_CM, inrmse_values_CM, color='green')

        plt.plot(k_values_CM, outrmse_values_CM, color='red')

        plt.locator_params(axis='x', nbins=45)
        plt.locator_params(axis='y', nbins=10)
```

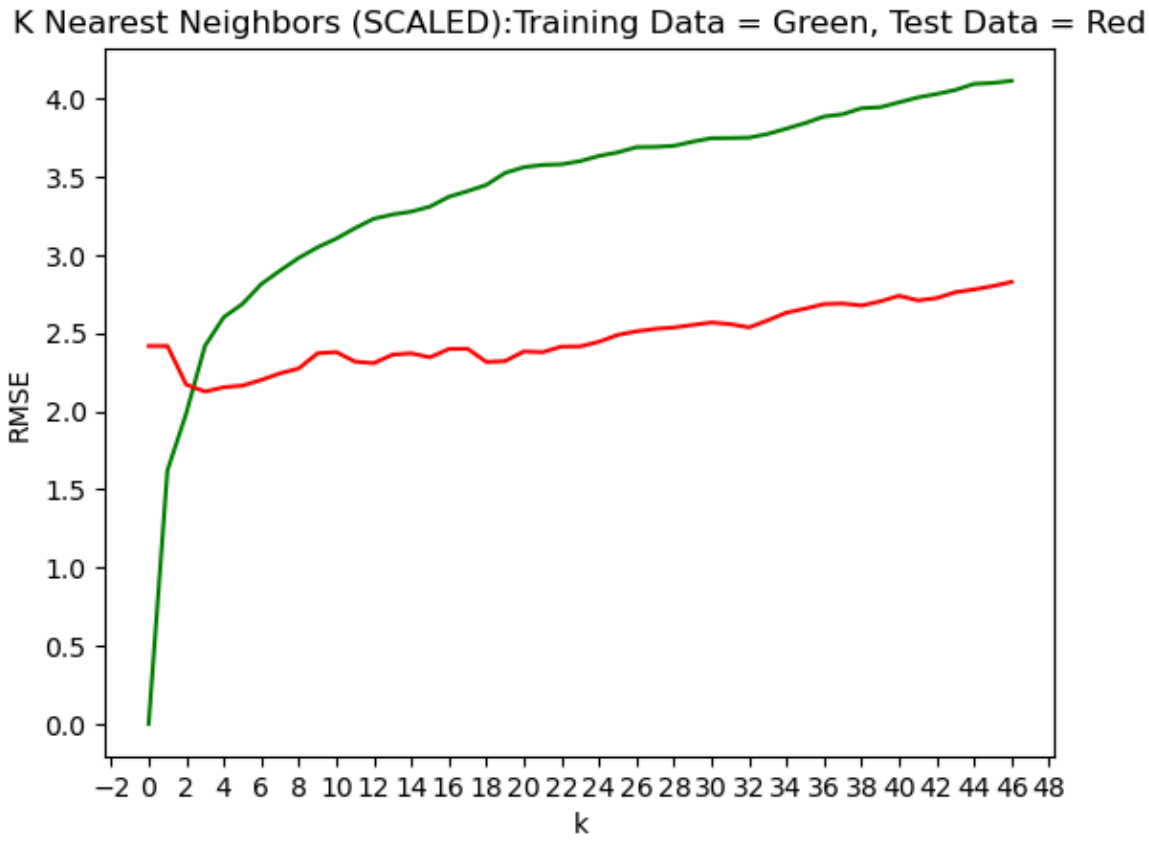
```
        if scaled == True:
            plt.title("K Nearest Neighbors (SCALED):Training Data = Green, Test
        else:
            plt.title("K Nearest Neighbors:Training Data = Green, Test Data = Re
        plt.xlabel("k")
        plt.ylabel("RMSE")

        plt.show()
    return {"xall_train_CM": xall_train_CM, "xall_test_CM": xall_test_CM, "yall_
```

We then tested for optimal k values and graphed their performance. This was suboptimal because the x values were not scaled which is often necessary or k nearest neighbor models to produce useful results. So we scaled the data and ran the function again.

In [129…

```
#xall_CM is unscaled so it does not give good results
result_attempt_1_CM = knn_optimal_k_test(xall_CM, False, True)
unscaled_k_CM = 4


#scaling x values
xall_std_CM = (xall_CM - xall_CM.mean())/(xall_CM.max() - xall_CM.min())


#xall_std_CM is the k based off the scaled values of x
result_attempt_2_CM = knn_optimal_k_test(xall_std_CM, True, True)
scaled_k_CM = 3
```

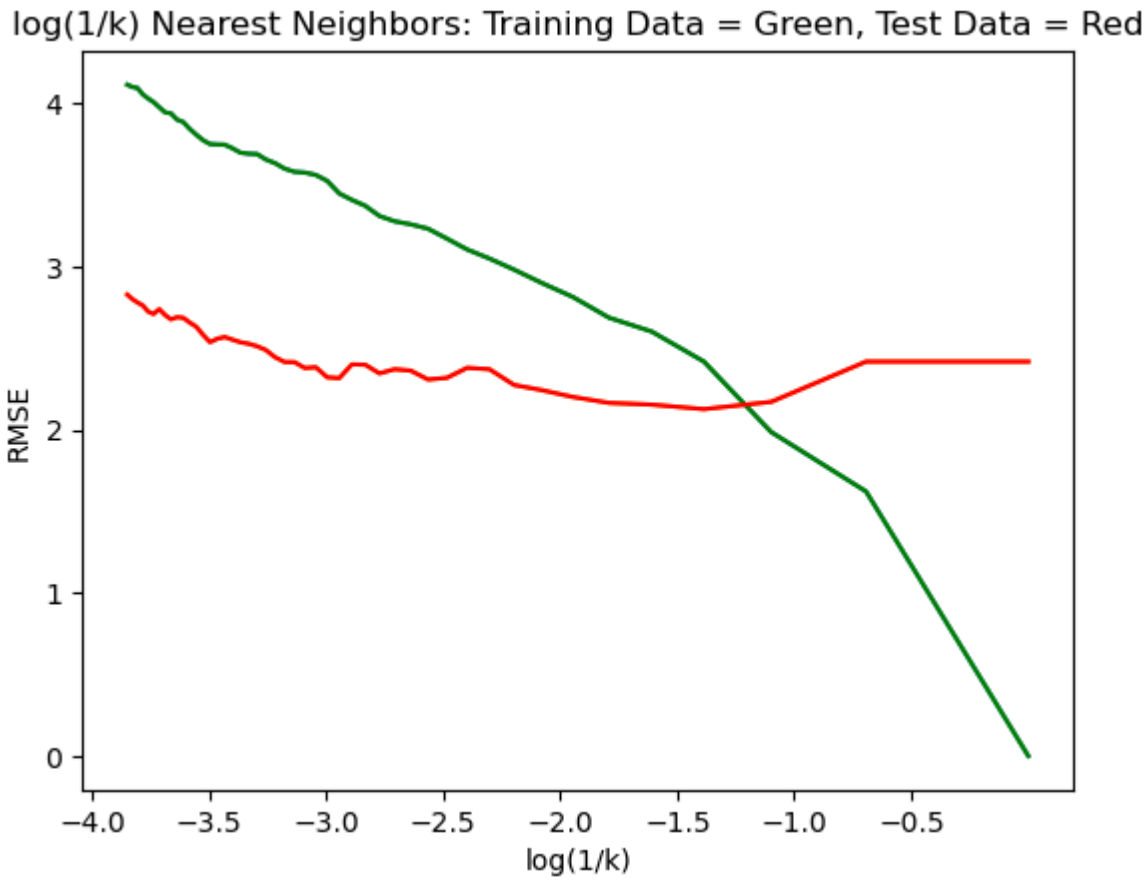




We next look at the log(1/k) version.

In [143…

```
#finding the log(1/k) graph for optimal k's
complexity_values_CM = np.arange(0.0,47.0)
for i in k_values_CM:
    complexity_CM = math.log(1/(k_values_CM[i]+1))
    complexity_values_CM[i] = complexity_CM
plt.plot(complexity_values_CM, inrmse_values_CM)
plt.plot(complexity_values_CM, outrmse_values_CM)
plt.plot(complexity_values_CM, inrmse_values_CM, color='green')
plt.plot(complexity_values_CM, outrmse_values_CM, color='red')
plt.title("log(1/k) Nearest Neighbors: Training Data = Green, Test Data = Red")
plt.xlabel("log(1/k)")
plt.ylabel("RMSE")

plt.xticks(np.arange(-4, 0, step=.5))
plt.yticks(np.arange(0, 5, step=1))
plt.show()
```



From the graphs we can determine that the optimal value of k for our model is likely k =3. We use this to fit another model.

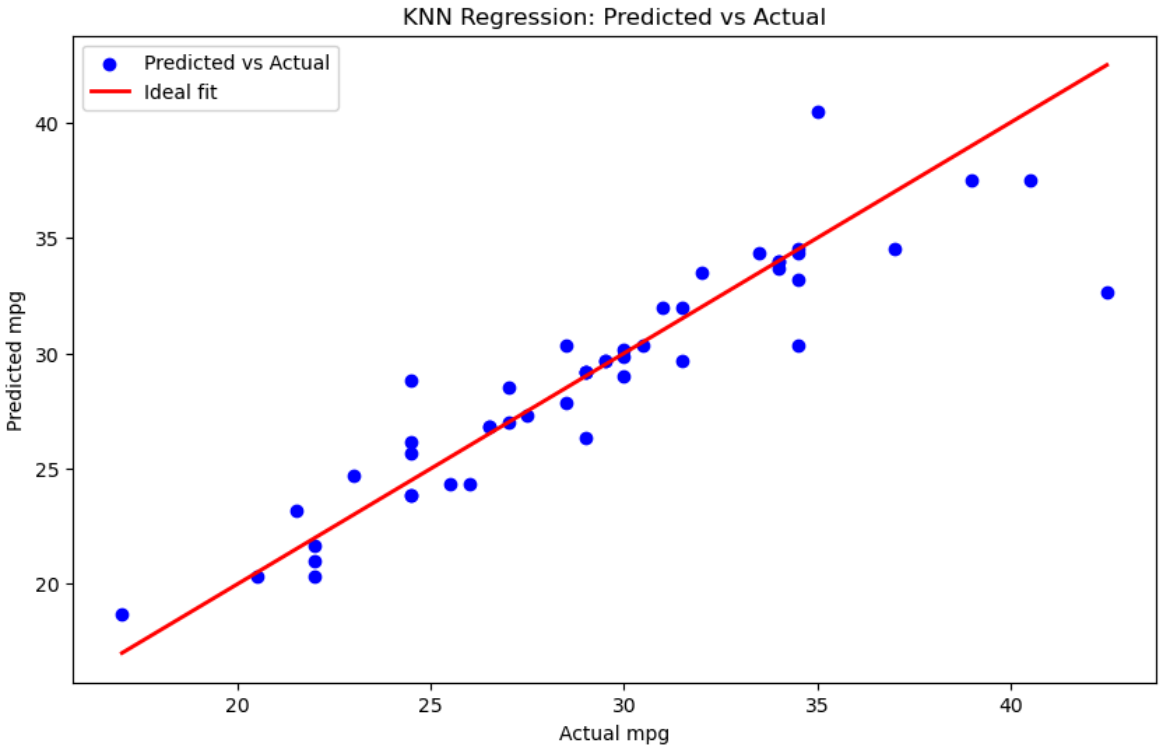
In [133…
```
#fitting k nearest neighbors using data and the optimal k value
kbest_CM = scaled_k_CM

knn_regressor_CM = KNeighborsRegressor(n_neighbors=kbest_CM)
knn_regressor_CM.fit(result_attempt_2_CM["xall_train_CM"], result_attempt_2_CM["
```

Out[133…
```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=3)
```

In [ ]:
```
Next, we make predictions using the model and graph the scatterplot and calculat
```

In [135…
```
# Make predictions on the test data using K nearest neighbors model and prints c
yall_pred_CM = knn_regressor_CM.predict(result_attempt_2_CM["xall_test_CM"])
rmse_CM = np.sqrt(mean_squared_error(result_attempt_2_CM["yall_test_CM"], yall_p
r2_CM = r2_score(result_attempt_2_CM["yall_test_CM"], yall_pred_CM)
print("RMSE:", rmse_CM)
print( "r2:", r2_CM)

plt.figure(figsize=(10, 6))
plt.scatter(result_attempt_2_CM["yall_test_CM"], yall_pred_CM, color='blue', lab
plt.plot([min(result_attempt_2_CM["yall_test_CM"]), max(result_attempt_2_CM["yal
plt.title('KNN Regression: Predicted vs Actual')
plt.xlabel('Actual mpg')
plt.ylabel('Predicted mpg')
plt.legend()
plt.show()
```

```
RMSE: 2.1704027617352843
r2: 0.8338582667120801
```



An R-squared value of 0.83 and a root mean squared error of 2.17 indicate that our model is fairly effective because it could explain 83% ofvariance. 2.17 mpg is a relatively small amount of error as well.

Finally, we end the report by performing cross-validation. For this we chose to use k-fold validation scoring on R-squared.

In [137…

```
#Cross Validation. I used k-fold cross validation scoring on R2.
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kf_CM = KFold(n_splits=10, shuffle=True, random_state=123)
model_CM = KNeighborsRegressor(n_neighbors=2)
scores_CM = cross_val_score(model_CM, xall_std_CM, yall_CM, cv=kf_CM, scoring='r
average_r2_CM = np.mean(scores_CM)

print(f"R² Score for each fold (k=2): {[round(score, 4) for score in scores_CM]}
print(f"Average R² (k=2): {average_r2_CM:.2f}")

kf_2_CM = KFold(n_splits=10, shuffle=True, random_state=123)
model_2_CM = KNeighborsRegressor(n_neighbors=3)
scores_2_CM = cross_val_score(model_2_CM, xall_std_CM, yall_CM, cv=kf_2_CM, scor
average_r2_2_CM= np.mean(scores_2_CM)

print(f"R² Score for each fold (k=3): {[round(score, 4) for score in scores_2_CM
print(f"Average R² (k=3): {average_r2_2_CM:.2f}")

kf_3_CM = KFold(n_splits=10, shuffle=True, random_state=123)
model_3_CM = KNeighborsRegressor(n_neighbors=4)
scores_3_CM = cross_val_score(model_3_CM, xall_std_CM, yall_CM, cv=kf_3_CM, scor
average_r2_3_CM = np.mean(scores_3_CM)
```

```
print(f"R² Score for each fold (k=4): {[round(score, 4) for score in scores_3_CM
print(f"Average R² (k=4): {average_r2_3_CM:.2f}")
```

```
R² Score for each fold (k=2): [0.7118, 0.7232, 0.5666, 0.4714, 0.7595, 0.6974, 0.
8195, 0.9797, 0.8891, 0.694]
Average R² (k=2): 0.73
R² Score for each fold (k=3): [0.7555, 0.7014, 0.5092, 0.7743, 0.859, 0.7193, 0.7
671, 0.9411, 0.8987, 0.652]
Average R² (k=3): 0.76
R² Score for each fold (k=4): [0.7857, 0.7051, 0.5113, 0.8553, 0.9174, 0.7413, 0.
7314, 0.8775, 0.9048, 0.6144]
Average R² (k=4): 0.76
```

The crossvalidation validates our choice for k.

In conclusion: We believe that this data-set can easily be used to estimate the mpg of imported cars. Our linear regression had an R-squared of .98 and a lower RMSE than the k-nearest neighbors model, so it likely is superior in some ways. The k nearest neighbor model on the other hand got an R-squared of .83 with only 4 predictor variables, making it the simpler model. For the linear regression we used every paramater and for the k nearest neighbor we used curbweight, engine-size, horse-power, and volume . The linear regression problem has a low level of variance and a high level of bias, and the k nearest neighbor has a higher level of variance and a lower level of bias. The linear regression is probably worse at predictions for out of sample data on average.