Predicting Car Prices

In this PROJECT, we explored the fundamentals of machine learning using the k-nearest neighbors algorithm to predict a car's market price using its attributes. The data set we will be working with contains information on various cars. For each car we have information about the technical aspects of the vehicle such as the motor's displacement, the weight of the car, the miles per gallon, how fast the car accelerates, and more. You can read more about the data set here and can download it directly from here. Here's a preview of the data set:

In [11]:

```
import pandas as pd
import numpy as np

cars = pd.read_csv('imports-85.data')
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 26 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   3            204 non-null    int64  
 1   ?            204 non-null    object 
 2   alfa-romero  204 non-null    object 
 3   gas          204 non-null    object 
 4   std          204 non-null    object 
 5   two          204 non-null    object 
 6   convertible  204 non-null    object 
 7   rwd          204 non-null    object 
 8   front        204 non-null    object 
 9   88.60        204 non-null    float64
 10  168.80       204 non-null    float64
 11  64.10        204 non-null    float64
 12  48.80        204 non-null    float64
 13  2548         204 non-null    int64  
 14  dohc         204 non-null    object 
 15  four         204 non-null    object 
 16  130          204 non-null    int64  
 17  mpfi         204 non-null    object 
 18  3.47         204 non-null    object 
 19  2.68         204 non-null    object 
 20  9.00         204 non-null    float64
 21  111          204 non-null    object 
```

```
 22  5000          204 non-null    object
 23  21            204 non-null    int64
 24  27            204 non-null    int64
 25  13495         204 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.6+ KB
```

In [12]:

```python
# Renaming the coloums.
cars = cars.rename(columns={'3':'symboling', '?':'normalized-losses','alfa-romero':'make','gas':'fuel-type',
 'std':'aspiration','two':'num-of-doors','convertible':'body-style','rwd':'drive-wheels','front':'engine-location',
'88.60':'wheel-base','168.80':'length','64.10':'width','48.80':'height','2548':'curb-weight','dohc':'engine-type','fo
'130':'engine-size','mpfi':'fuel-system', '3.47':'bore','2.68':'stroke', '9.00':'compression-rate','111':'horsepower
 ,'5000':'peak-rpm' ,'21':'city-mpg', '27':'highway-mpg' ,'13495':'price'})

cars
```

Out[12]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compressio ra
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
199	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9
200	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8
201	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8
202	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23
203	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9

204 rows × 26 columns

In [13]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          204 non-null    int64  
 1   normalized-losses  204 non-null    object 
 2   make               204 non-null    object 
 3   fuel-type          204 non-null    object 
 4   aspiration         204 non-null    object 
 5   num-of-doors       204 non-null    object 
 6   body-style         204 non-null    object 
 7   drive-wheels       204 non-null    object 
 8   engine-location    204 non-null    object 
 9   wheel-base         204 non-null    float64
 10  length             204 non-null    float64
 11  width              204 non-null    float64
 12  height             204 non-null    float64
 13  curb-weight        204 non-null    int64  
 14  engine-type        204 non-null    object 
 15  num-of-cylinders   204 non-null    object 
 16  engine-size        204 non-null    int64  
 17  fuel-system        204 non-null    object 
 18  bore               204 non-null    object 
 19  stroke             204 non-null    object 
 20  compression-rate   204 non-null    float64
 21  horsepower         204 non-null    object 
 22  peak-rpm           204 non-null    object 
 23  city-mpg           204 non-null    int64  
 24  highway-mpg        204 non-null    int64  
 25  price              204 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.6+ KB
```

In [14]:
```
# We select only columns with numeric value.
numric_cols = ['normalized-losses', 'wheel-base', 'length', 'width', 'height', 'curb-weight','engine-size', 'bore'
               , 'stroke', 'compression-rate', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']

numeric_cars = cars[numric_cols]
numeric_cars
```

Out[14]:

	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-rate	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	?	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	5000	21	27	16500
1	?	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154	5000	19	26	16500
2	164	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102	5500	24	30	13950
3	164	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115	5500	18	22	17450
4	?	99.8	177.3	66.3	53.1	2507	136	3.19	3.40	8.5	110	5500	19	25	15250
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
199	95	109.1	188.8	68.9	55.5	2952	141	3.78	3.15	9.5	114	5400	23	28	16845
200	95	109.1	188.8	68.8	55.5	3049	141	3.78	3.15	8.7	160	5300	19	25	19045
201	95	109.1	188.8	68.9	55.5	3012	173	3.58	2.87	8.8	134	5500	18	23	21485
202	95	109.1	188.8	68.9	55.5	3217	145	3.01	3.40	23.0	106	4800	26	27	22470
203	95	109.1	188.8	68.9	55.5	3062	141	3.78	3.15	9.5	114	5400	19	25	22625

204 rows × 15 columns

Data Cleaning

In [15]:

```
numeric_cars = numeric_cars.replace("?", np.nan)
numeric_cars = numeric_cars.astype('float')
numeric_cars.isnull().sum()
```

Out[15]:

```
normalized-losses    40
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-size           0
bore                  4
stroke                4
compression-rate      0
```

```
horsepower           2
peak-rpm             2
city-mpg             0
highway-mpg          0
price                4
dtype: int64
```

In [16]:
```
# The 'price' column is the target and we going to remove the missing value for it.

numeric_cars = numeric_cars.dropna(subset=['price'])

# Filling or Replace the missing values using the average values from that column.
numeric_cars = numeric_cars.fillna(numeric_cars.mean())

# Confirm that there's no more missing values!
numeric_cars.isnull().sum()
```

Out[16]:
```
normalized-losses    0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-size          0
bore                 0
stroke               0
compression-rate     0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

In [17]:
```
# Normalize all columnns to range from 0 to 1 except the target column.
price_col= numeric_cars['price']

numeric_cars = ((numeric_cars - numeric_cars.min())/ (numeric_cars.max() - numeric_cars.min()) )
numeric_cars
```

Out[17]:

	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-rate	horsepower	peak-rpm	city-mpg

	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-rate	horsepower	peak-rpm	city-mpg
0	0.298429	0.058309	0.413433	0.324786	0.083333	0.411171	0.260377	0.664286	0.290476	0.12500	0.294393	0.346939	0.222222
1	0.298429	0.230321	0.449254	0.444444	0.383333	0.517843	0.343396	0.100000	0.666667	0.12500	0.495327	0.346939	0.166667
2	0.518325	0.384840	0.529851	0.504274	0.541667	0.329325	0.181132	0.464286	0.633333	0.18750	0.252336	0.551020	0.305556
3	0.518325	0.373178	0.529851	0.521368	0.541667	0.518231	0.283019	0.464286	0.633333	0.06250	0.313084	0.551020	0.138889
4	0.298429	0.384840	0.540299	0.512821	0.441667	0.395268	0.283019	0.464286	0.633333	0.09375	0.289720	0.551020	0.166667
...	...	...	...	...	...	...	...	...	...	...	...	...	...
199	0.157068	0.655977	0.711940	0.735043	0.641667	0.567882	0.301887	0.885714	0.514286	0.15625	0.308411	0.510204	0.277778
200	0.157068	0.655977	0.711940	0.726496	0.641667	0.605508	0.301887	0.885714	0.514286	0.10625	0.523364	0.469388	0.166667
201	0.157068	0.655977	0.711940	0.735043	0.641667	0.591156	0.422642	0.742857	0.380952	0.11250	0.401869	0.551020	0.138889
202	0.157068	0.655977	0.711940	0.735043	0.641667	0.670675	0.316981	0.335714	0.633333	1.00000	0.271028	0.265306	0.361111
203	0.157068	0.655977	0.711940	0.735043	0.641667	0.610551	0.301887	0.885714	0.514286	0.15625	0.308411	0.510204	0.166667

200 rows × 15 columns

Univariate Model

In [18]:

```
half = int(len(cars)/2)
half
```

Out[18]: 102

In [25]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
def knn_train_test(train_col, target_col, df):
    knn = KNeighborsRegressor()
    np.random.seed(1)

    # Randomize order of rows in data frame.
```

```python
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Divide number of rows in half and round.
    last_train_row = int(len(rand_df) / 2)

    # Select the first half and set as training set.
    # Select the second half and set as test set.
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    # Fit a KNN model using default k value.
    knn.fit(train_df[[train_col]], train_df[target_col])

    # Make predictions using model.
    predicted_labels = knn.predict(test_df[[train_col]])

    # Calculate and return RMSE.
    mse = mean_squared_error(test_df[target_col], predicted_labels)
    rmse = np.sqrt(mse)
    return rmse

rmse_results = {}
train_cols = numeric_cars.columns.drop('price')

# For each column (minus `price`), train a model, return RMSE value
# and add to the dictionary `rmse_results`.
for col in train_cols:
    rmse_val = knn_train_test(col, 'price', numeric_cars)
    rmse_results[col] = rmse_val

# Create a Series object from the dictionary so
# we can easily view the results, sort, etc

rmse_results_series = pd.Series(rmse_results)
rmse_results_series.sort_values()
```

Out[25]:
```
engine-size          0.080611
curb-weight          0.085385
highway-mpg          0.092775
width                0.093668
city-mpg             0.094662
```

```
horsepower           0.110624
length               0.127860
wheel-base           0.135144
bore                 0.154087
peak-rpm             0.160329
compression-rate     0.178581
height               0.183224
stroke               0.203172
normalized-losses    0.205837
dtype: float64
```

In [26]:

```python
def knn_train_test(train_col, target_col, df):
    np.random.seed(1)

    # Randomize order of rows in data frame.
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Divide number of rows in half and round.
    last_train_row = int(len(rand_df) / 2)

    # Select the first half and set as training set.
    # Select the second half and set as test set.
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    k_values = [1,3,5,7,9]
    k_rmses = {}

    for k in k_values:
        # Fit model using k nearest neighbors.
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[[train_col]], train_df[target_col])

        # Make predictions using model.
        predicted_labels = knn.predict(test_df[[train_col]])

        # Calculate and return RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses
```

```python
k_rmse_results = {}

# For each column (minus `price`), train a model, return RMSE value
# and add to the dictionary `rmse_results`.
train_cols = numeric_cars.columns.drop('price')
for col in train_cols:
    rmse_val = knn_train_test(col, 'price', numeric_cars)
    k_rmse_results[col] = rmse_val

k_rmse_results
```

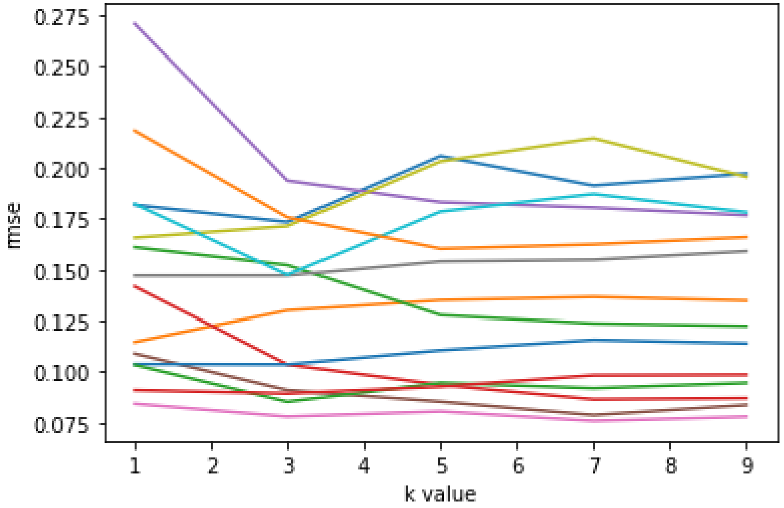
Out[26]:
```
{'normalized-losses': {1: 0.1818763045968068,
  3: 0.17344105965183793,
  5: 0.20583693425897426,
  7: 0.1913746146529398,
  9: 0.19717945344969953},
 'wheel-base': {1: 0.11461336816072751,
  3: 0.13014284920353655,
  5: 0.1351436707966052,
  7: 0.13678560541170387,
  9: 0.13493694151726923},
 'length': {1: 0.16106247820811412,
  3: 0.1522785828170357,
  5: 0.12785981770710697,
  7: 0.12370207624703503,
  9: 0.1224340389552924},
 'width': {1: 0.14184043272575958,
  3: 0.10365348902972947,
  5: 0.09366802786854177,
  7: 0.08654077208025306,
  9: 0.08710781493630108},
 'height': {1: 0.27084948978743506,
  3: 0.19376767770067874,
  5: 0.1832239923366418,
  7: 0.1805405938911076,
  9: 0.17678381557993628},
 'curb-weight': {1: 0.10900347357296251,
  3: 0.09106695158443022,
  5: 0.08538532444655243,
  7: 0.07879622438965092,
  9: 0.0837518613233152},
 'engine-size': {1: 0.08435915617662802,
  3: 0.07804085616859407,
```

```
 5: 0.08061121569086101,
 7: 0.07592112695960995,
 9: 0.07799135029764635},
'bore': {1: 0.1471256146860233 6,
 3: 0.14716030365349458,
 5: 0.15408680210251005,
 7: 0.15488613250237387,
 9: 0.15907522673699317},
'stroke': {1: 0.1657062602465034,
 3: 0.17147135492631432,
 5: 0.20317228751708463,
 7: 0.21453565331975855,
 9: 0.19562461198365874},
'compression-rate': {1: 0.18231971967389596,
 3: 0.14754525362636353,
 5: 0.17858072089604907,
 7: 0.18719478744191398,
 9: 0.17825358296102287},
'horsepower': {1: 0.1038453063524153,
 3: 0.10366968829584426,
 5: 0.11062448786759559,
 7: 0.1156360098368225,
 9: 0.113982883125065},
'peak-rpm': {1: 0.21826663136572744,
 3: 0.1757794057000707,
 5: 0.1603291413865285,
 7: 0.16246465227267864,
 9: 0.1659906621228989},
'city-mpg': {1: 0.10352103387194748,
 3: 0.08534852284784691,
 5: 0.09466245837776571,
 7: 0.09201971060380984,
 9: 0.09460898192640313},
'highway-mpg': {1: 0.09105075799560801,
 3: 0.08943818679198769,
 5: 0.0927749875105845,
 7: 0.09838198244104572,
 9: 0.09861658024688481}}
```

In [27]:

```
import matplotlib.pyplot as plt
for k , v in k_rmse_results.items():
    x = list(v.keys())
    y = list(v.values())
```

```
    plt.plot(x,y)
    plt.xlabel('k value')
    plt.ylabel('rmse')
```



In [28]:

```
# Compute average RMSE across different `k` values for each feature.
feature_avg_rmse = {}
for k,v in k_rmse_results.items():
    avg_rmse = np.mean(list(v.values()))
    feature_avg_rmse[k] = avg_rmse
series_avg_rmse = pd.Series(feature_avg_rmse)
sorted_series_avg_rmse = series_avg_rmse.sort_values()
print(sorted_series_avg_rmse)

sorted_features = sorted_series_avg_rmse.index
```

```
engine-size         0.079385
curb-weight         0.089601
city-mpg            0.094032
highway-mpg         0.094052
width               0.102562
horsepower          0.109552
```

```
wheel-base           0.130324
length               0.137467
bore                 0.152467
compression-rate     0.174779
peak-rpm             0.176566
normalized-losses    0.189942
stroke               0.190102
height               0.201033
dtype: float64
```

In [30]:
```
sorted_features
```

Out[30]:
```
Index(['engine-size', 'curb-weight', 'city-mpg', 'highway-mpg', 'width',
       'horsepower', 'wheel-base', 'length', 'bore', 'compression-rate',
       'peak-rpm', 'normalized-losses', 'stroke', 'height'],
      dtype='object')
```

In [31]:
```
def knn_train_test(train_cols, target_col, df):
    np.random.seed(1)

    # Randomize order of rows in data frame.
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Divide number of rows in half and round.
    last_train_row = int(len(rand_df) / 2)

    # Select the first half and set as training set.
    # Select the second half and set as test set.
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    k_values = [5]
    k_rmses = {}

    for k in k_values:
        # Fit model using k nearest neighbors.
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[train_cols], train_df[target_col])

        # Make predictions using model.
        predicted_labels = knn.predict(test_df[train_cols])
```

```
        # Calculate and return RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses

k_rmse_results = {}

for nr_best_feats in range(2,7):
    k_rmse_results['{} best features'.format(nr_best_feats)] = knn_train_test(
        sorted_features[:nr_best_feats],
        'price',
        numeric_cars
    )

k_rmse_results
```

Out[31]:
```
{'2 best features': {5: 0.06893373527829134},
 '3 best features': {5: 0.07465738053571394},
 '4 best features': {5: 0.07962147864727218},
 '5 best features': {5: 0.06603803515304489},
 '6 best features': {5: 0.06766700291128268}}
```

Hyperparameter Tuning

In [38]:
```
def knn_train_test(train_cols, target_col, df):
    np.random.seed(1)

    # Randomize order of rows in data frame.
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)


    # Divide number of rows in half and round.
    last_train_row = int(len(rand_df) / 2)

    # Select the first half and set as training set.
    # Select the second half and set as test set.
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]
```

```python
    k_values = [i for i in range(1, 25)]
    k_rmses = {}

    for k in k_values:
        # Fit model using k nearest neighbors.
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[train_cols], train_df[target_col])

        # Make predictions using model.
        predicted_labels = knn.predict(test_df[train_cols])

        # Calculate and return RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses

k_rmse_results = {}

for nr_best_feats in range(2,6):
    k_rmse_results['{} best features'.format(nr_best_feats)] = knn_train_test(
        sorted_features[:nr_best_feats],
        'price',
        numeric_cars
    )

k_rmse_results
```

Out[38]:
```
{'2 best features': {1: 0.08452909750575176,
  2: 0.06967916007172928,
  3: 0.07030551262142683,
  4: 0.06841481781796252,
  5: 0.06893373527829134,
  6: 0.07063530849867736,
  7: 0.07298455834534161,
  8: 0.07671069533450645,
  9: 0.08160955838337274,
  10: 0.08503025678185276,
```

```
 11: 0.08898339859199846,
 12: 0.09025187398347476,
 13: 0.09094890375973841,
 14: 0.09097198068437448,
 15: 0.09139547174356448,
 16: 0.09204173648539048,
 17: 0.09286188909600467,
 18: 0.09213555521205848,
 19: 0.09304262187875068,
 20: 0.09263133338475311,
 21: 0.09238504129063176,
 22: 0.09320644806554826,
 23: 0.09356446048058417,
 24: 0.09447572330721757},
'3 best features': {1: 0.06895437769280939,
 2: 0.06469461208969603,
 3: 0.06963242771366582,
 4: 0.07382559998358658,
 5: 0.07465738053571394,
 6: 0.0785615918795779,
 7: 0.08244172717588394,
 8: 0.0801193115298529,
 9: 0.07999904524553479,
 10: 0.08395215619541548,
 11: 0.0872514947788235,
 12: 0.08634598033460358,
 13: 0.08607312867484797,
 14: 0.0850444823642875,
 15: 0.08642892194197596,
 16: 0.08755515528566764,
 17: 0.08841108578049602,
 18: 0.08825221214091544,
 19: 0.08857001743940204,
 20: 0.08971138135908387,
 21: 0.09138140129608689,
 22: 0.09310720225330019,
 23: 0.09431223703049021,
 24: 0.09532983417225747},
'4 best features': {1: 0.06279084831216285,
 2: 0.06047828574808828,
 3: 0.0667132230670264,
 4: 0.07421391898440018,
 5: 0.07962147864727218,
 6: 0.07913357368399024,
 7: 0.08297552533569798,
```

```
 8: 0.08570241466546578,
 9: 0.08627424274136392,
 10: 0.08471590400104777,
 11: 0.08643483793124518,
 12: 0.08538416813460914,
 13: 0.08767391370501407,
 14: 0.08801388514383543,
 15: 0.08724659649182212,
 16: 0.08722247262388896,
 17: 0.08799211796899863,
 18: 0.08892697369419231,
 19: 0.08909958844669832,
 20: 0.09001713049001828,
 21: 0.09167054578928759,
 22: 0.09384149205066769,
 23: 0.09582742354306427,
 24: 0.09709564794201345},
'5 best features': {1: 0.06257463252817799,
 2: 0.061700896570656305,
 3: 0.06861415569220723,
 4: 0.06669624856801665,
 5: 0.06603803515304489,
 6: 0.0645594884545092,
 7: 0.06675137780776288,
 8: 0.065525453691271,
 9: 0.06952628358772024,
 10: 0.07346616821569749,
 11: 0.07751080301669909,
 12: 0.08148031886617897,
 13: 0.08511927510442399,
 14: 0.08663290694444761,
 15: 0.08961254155838318,
 16: 0.09140995014896292,
 17: 0.09208114317236354,
 18: 0.09336797023095932,
 19: 0.09508314854367717,
 20: 0.09553160500843955,
 21: 0.0972457815055955,
 22: 0.0981610015303951,
 23: 0.09958274919434452,
 24: 0.10069562323293461}}
```

In [53]:

```
for k,v in k_rmse_results.items():
    x = list(v.values())
```

```
    y = list(v.keys())
    plt.plot(x,y, label=('{}'.format(k)))
plt.xlabel('k value')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```