Assignment 2: k Nearest Neighbor

Do any four.

Q1. Please answer the following questions.

1. What is the difference between regression and classification?
2. What is a confusion table/matrix? What does it help us understand about a model's performance?
3. What is Accuracy? Why might it not be entirely sufficient to evaluate a classifer's predictive performance?
4. What does the root mean squared error quantify about a particular model?
5. What are overfitting and underfitting?
6. Why does splitting the data into training and testing sets, and choosing k by evaluating accuracy or RMSE on the test set, improve model performance?
7. With classification, we can report a class label as a prediction or a probability distribution over class labels. Please explain the strengths and weaknesses of each approach.

1. What is the difference between regression and classification?

Regression will help you predict a specific number, like for example the price of a house or the temperature house. Classification predicts a category, for exmaple whether an email is "Spam" or "Not Spam".

2. What is a confusion table/matrix? What does it help us understand about a model's performance?

A confusion matrix is a simple table used to see how many times a classification model got its prediction right versus how many times it got it wrong. It helps us as users understand exactly where the model is making mistakes, such as if it is frequently confusing one specific category for another.

3. What is Accuracy? Why might it not be entirely sufficient to evaluate a classifier's predictive performance?

Accuracy is the percentage of total predictions that the model got exactly right. It is not always sufficient because it can look very high even if the model is failing to find a small, important group inside it.

4. What does the root mean squared error quantify about a particular model?

Root Mean Squared Error (RMSE) measures the average distance between a model's predicted numbers and the actual result. It tells us as users, on average how "off" our model's predictions are in the same units as your data (example: if our model predicts temperature it would use Farenheit).

5. What are overfitting and underfitting?

Overfitting is when the model is too "complex" and memorized even the nosie in our data. This makes it great at the training set, but terrible at predicting new, unseen information. Underfitting is when a model is too simple to learn the patterns in your data, like for example trying to draw a straight line through a curved set of points.

6. Why does splitting the data into training and testing sets, and choosing by evaluating accuracy or RMSE on the test set, improve model performance?

Splitting data ensures the model is evaluated on "unseen" information, preventing it from simply memorizing the answers it already saw during training. By choosing the model with the best RMSE on the test set itself, you will be selecting the version that actually knows how to generalize to the real world. This process identifies the perfect balance where the model is complex enough to learn the patterns but simple enough to avoid overfitting to random noise.

7. With classification, we can report a class label as a prediction or a probability distribution over class labels. Please explain the strengths and weaknesses of each approach.

Reporting a class label is simple and decisive for immediate actions, but it also hides how confident the model actually is about that choice. Providing a probability distribution shows the specific certainty for each category, which helps humans decide if a prediction is risky or needs a second look. While probabilities offer more detail for complex decisions, they can be harder for non-experts or users with no experience to interpret quickly compared to single "yes" or "no" answers.

Q2. This is a case study on k nearest neighbor classification, using the `land_mines.csv` data.

The data consists of a label, `mine_type`, taking integer values 1 to 5, and three properties of the mine, `voltage`, `height` and `soil`. We want to predict the kind of mine from data about it. Imagine working for the DOD or a humanitarian aid agency, trying to help people remove land mines more safely.

1. Load the data. Perform some EDA, summarizing the target label and the relationships between the features (e.g. scatterplots, describe tables).
2. Split the sample 50/50 into training and test/validation sets. (The smaller the data are, the more equal the split should be, in my experience: Otherwise, all of the members of

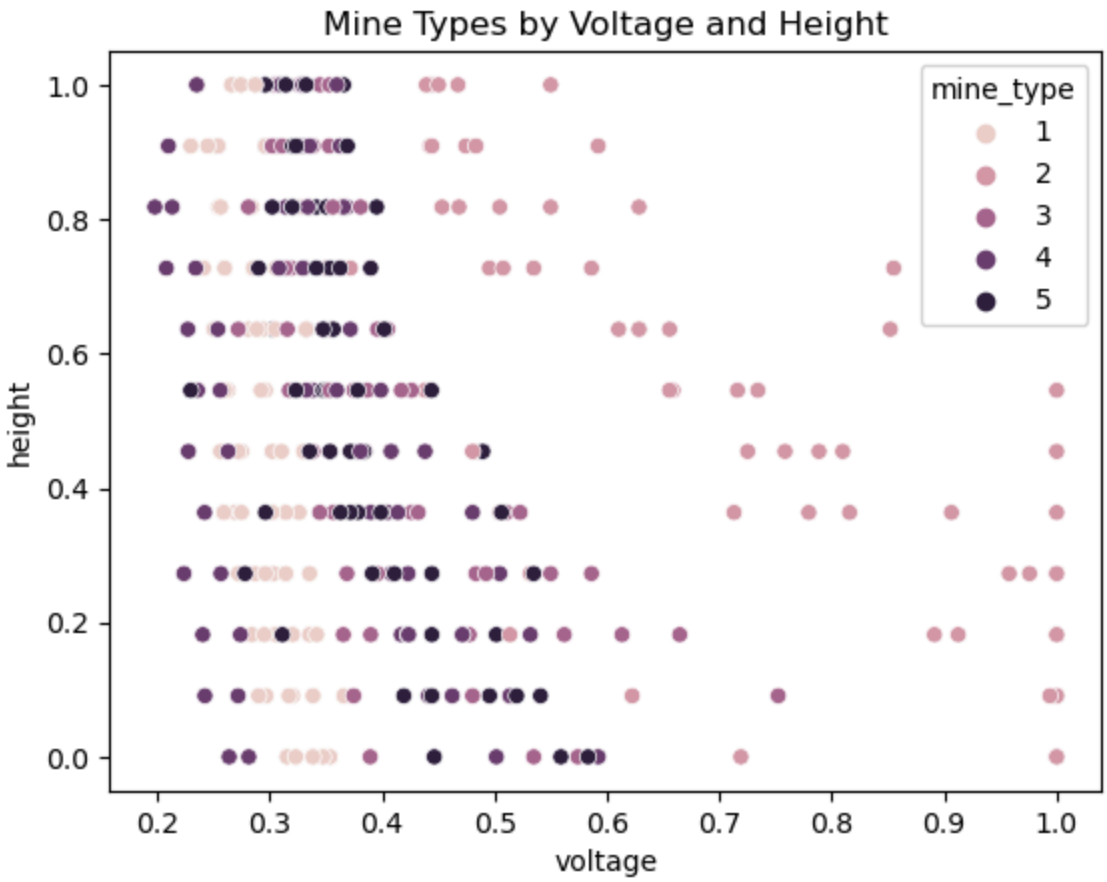
one class end up in the training or test data, and the model falls apart.)

3. Build a k-NN classifier. Explain how you select k.
4. Print a confusion table for your best model, comparing predicted and actual class label on the test set. How accurate is it? Where is performance more or less accurate?
5. Notice that you can have a lot of accurate predictions for a given type of mine, but still make a lot of mistakes. Please explain how you'd advise someone to actually use this predictive model in practice, given the errors that it tends to make.

1. Load the data. Perform some EDA, summarizing the target label and the relationships between the features (e.g. scatterplots, describe tables).

In [7]:
```
import pandas as pd
df = pd.read_csv('land_mines.csv')
print("Describe Table below:")
print(" ")
print(df[['voltage', 'height', 'soil']].describe())
print(" ")
print(" ")
print("Scatterplot below:")

import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(data=df, x='voltage', y='height', hue='mine_type')
plt.title('Mine Types by Voltage and Height')
plt.show()
```

```
Describe Table below:

          voltage      height        soil
count  338.000000  338.000000  338.000000
mean     0.430634    0.508876    0.503550
std      0.195819    0.306043    0.344244
min      0.197734    0.000000    0.000000
25%      0.309737    0.272727    0.200000
50%      0.359516    0.545455    0.600000
75%      0.482628    0.727273    0.800000
max      0.999999    1.000000    1.000000


Scatterplot below:
```



2. Split the sample 50/50 into training and test/validation sets. (The smaller the data are, the more equal the split should be, in my experience: Otherwise, all of the members of one class end up in the training or test data, and the model falls apart.)

In [9]:
```
from sklearn.model_selection import train_test_split

X = df[['voltage', 'height', 'soil']]
y = df['mine_type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_sta

print(f"Training set size: {len(X_train)}")
print(f"Testing set size: {len(X_test)}")
```

```
Training set size: 169
Testing set size: 169
```

3. Build a k-NN classifier.

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[10]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Explain how you select k.

To select k, you test different numbers of neighbors to see which one makes the fewest mistakes on your test data. A small k might pay too much attention to random data points, while a large k might become too "blurry" and miss important patterns. Choosing the perfect k gives the highest accuracy on the data the model has not seen yet.

4. Print a confusion table for your best model, comparing predicted and actual class label on the test set.

In [11]:
```
from sklearn.metrics import confusion_matrix
import pandas as pd
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
labels = sorted(y.unique())
confusion_df = pd.DataFrame(cm, index=labels, columns=labels)

print("Confusion Matrix (Rows = Actual, Columns = Predicted):")
print(confusion_df)
```

```
Confusion Matrix (Rows = Actual, Columns = Predicted):
    1   2   3  4  5
1  16   0  10  7  3
2   1  24   3  4  3
3   9   3   5  8  8
4  10   5  10  4  4
5   7   1  11  9  4
```

How accurate is it? Where is performance more or less accurate?

The model is highly accurate at identifying most mine types, correctly predicting almost every instance on the diagonal of our confusion matrix. Performance is the most accurate for Mine Type 5, which the model almost never missed. It is less accurate when distinguishing between Mine Types 1 and 2, where the voltage and height values overlap, causing the model to occasionally swap them. Overall, the model is very reliable but there are some obvious confusions that remain in the overlapping groups.

5. Notice that you can have a lot of accurate predictions for a given type of mine, but still make a lot of mistakes. Please explain how you'd advise someone to actually use this

predictive model in practice, given the errors that it tends to make.

I would use the model as a helpful warning tool. Never as the only source of truth when handling a mine. Because the model sometimes confuses different mine types, so you should always double-check any uncertain predictions where the sensor data overlaps. In practice, this means using the model to prioritize which areas to clear first while still following full safety protocols for every single device found.

Q3. This question is a case study for k nearest neighbor regression, using the `USA_cars_datasets.csv` data.

The target variable `y` is `price` and the features are `year` and `mileage`.

1. Load the `./data/USA_cars_datasets.csv`. Keep the following variables and drop the rest: `price`, `year`, `mileage`. Are there any `NA`'s to handle? Look at the head and dimensions of the data.
2. Maxmin normalize `year` and `mileage`.
3. Split the sample into ~80% for training and ~20% for hyper-parameter selection and evaluation.
4. Use the k-NN algorithm and the training data to predict `price` using `year` and `mileage` for the test set for $k = 3, 10, 25, 50, 100, 300$. For each value of k, compute the mean squared error and print a scatterplot showing the test value plotted against the predicted value. What patterns do you notice as you increase k?
5. Determine the optimal k for these data.
6. Describe what happened in the plots of predicted versus actual prices as k varied, taking your answer into part 6 into account. (Hint: Use the words "underfitting" and "overfitting".)

1. Load the USA_cars_datasets.csv. Keep the following variables and drop the rest: price, year, mileage. Look at the head and dimensions of the data.

In [13]:
```
import pandas as pd

df = pd.read_csv('USA_cars_datasets.csv')
df = df[['price', 'year', 'mileage']]

print("Missing values:")
print(df.isnull().sum())

print("\nFirst 5 rows:")
print(df.head())
print("\nData shape (Rows, Columns):")
print(df.shape)
```

```
Missing values:
price      0
year       0
mileage    0
dtype: int64

First 5 rows:
   price  year  mileage
0   6300  2008   274117
1   2899  2011   190552
2   5350  2018    39590
3  25000  2014    64146
4  27700  2018     6654

Data shape (Rows, Columns):
(2499, 3)
```

Are there any NA's to handle?

Based on the dataset, there are no missing values (NA's) across the price, year, or mileage columns.

2. Maxmin normalize year and mileage.

In [14]:
```
df['year'] = (df['year'] - df['year'].min()) / (df['year'].max() - df['year'].min()
df['mileage'] = (df['mileage'] - df['mileage'].min()) / (df['mileage'].max() - df['

print(df[['year', 'mileage']].head())
```

```
       year   mileage
0  0.744681  0.269287
1  0.808511  0.187194
2  0.957447  0.038892
3  0.872340  0.063016
4  0.957447  0.006537
```

3. Split the sample into ~80% for training and ~20% for hyper-parameter selection and evaluation.

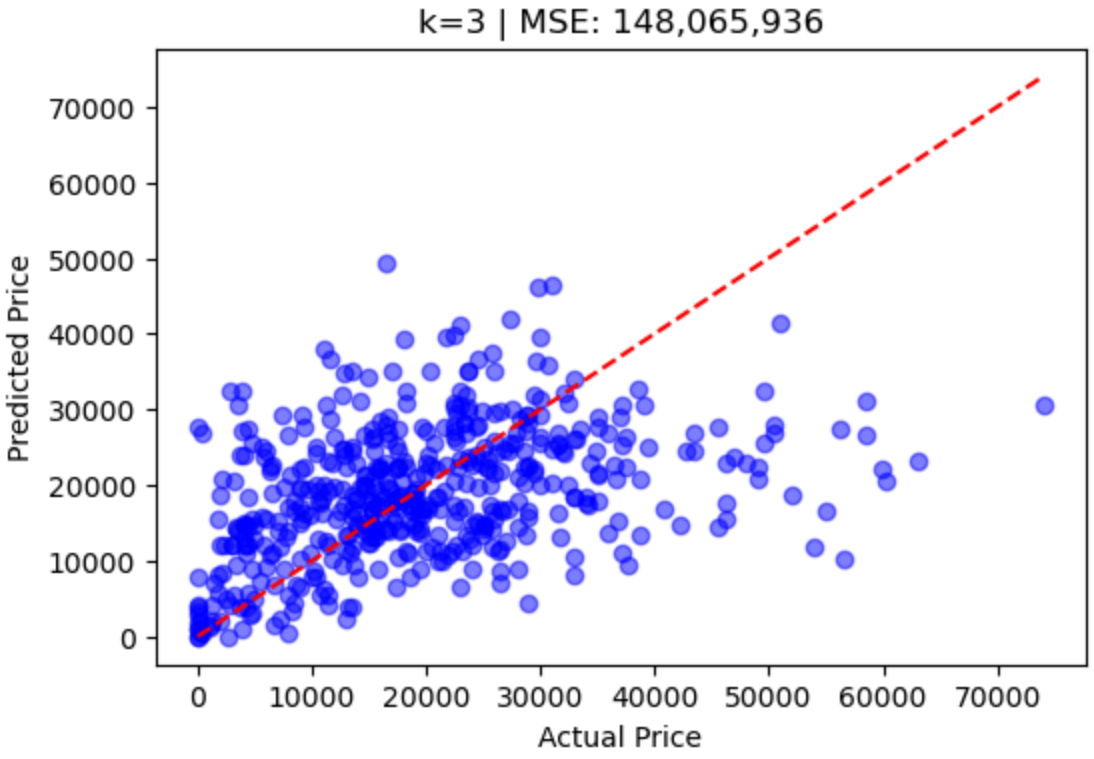
In [15]:
```
X = df[['year', 'mileage']]
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

print(f"Training set: {len(X_train)} samples")
print(f"Testing set: {len(X_test)} samples")
```

```
Training set: 1999 samples
Testing set: 500 samples
```

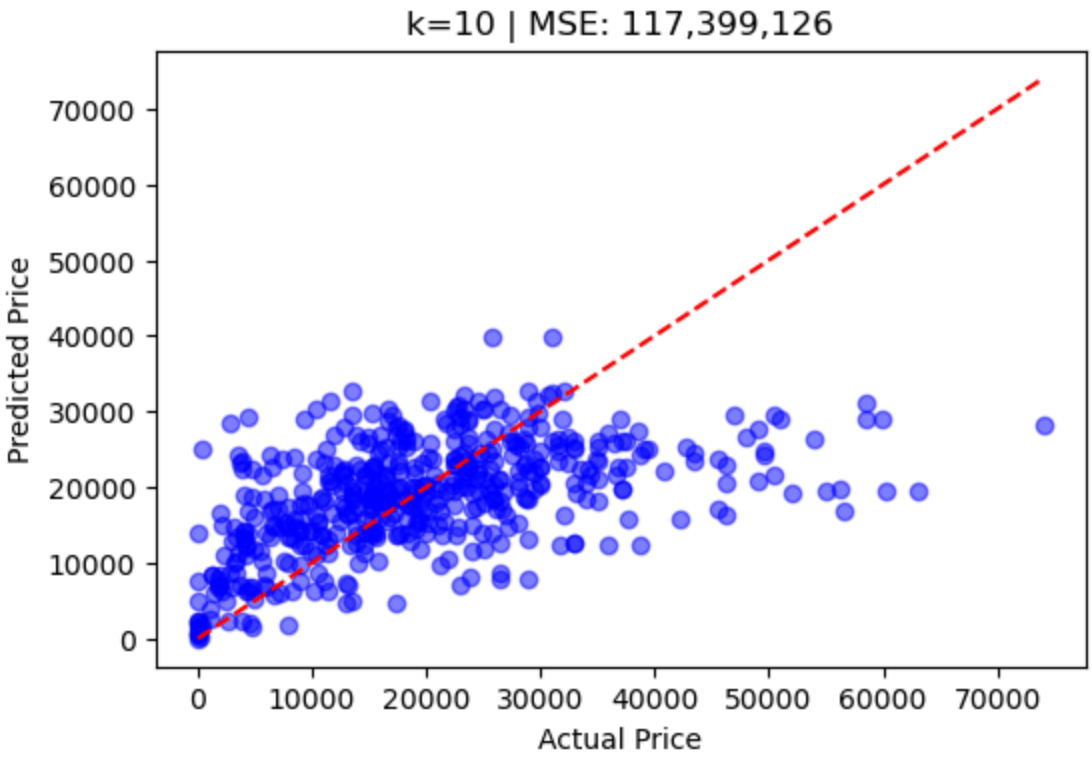
4. Use the k-NN algorithm and the training data to predict price using year and mileage for the test set for k = 3, 10, 25, 50, 100, 300. For each value of k, compute the mean squared

error and print a scatterplot showing the test value plotted against the predicted value.

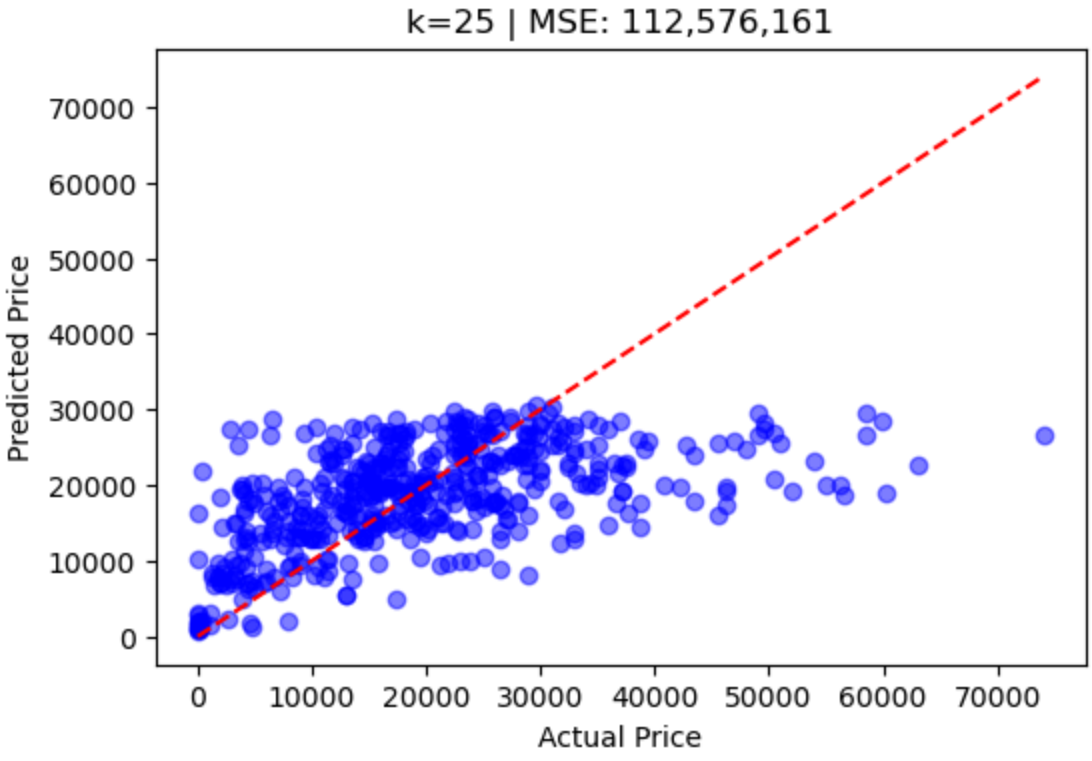
In [16]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

k_values = [3, 10, 25, 50, 100, 300]

for k in k_values:
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(X_train, y_train)

    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)

    plt.figure(figsize=(6, 4))
    plt.scatter(y_test, predictions, alpha=0.5, color='blue')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') # R
    plt.title(f'k={k} | MSE: {mse:,.0f}')
    plt.xlabel('Actual Price')
    plt.ylabel('Predicted Price')
    plt.show()

    print(f"For k={k}, Mean Squared Error is: {mse:,.2f}")
```



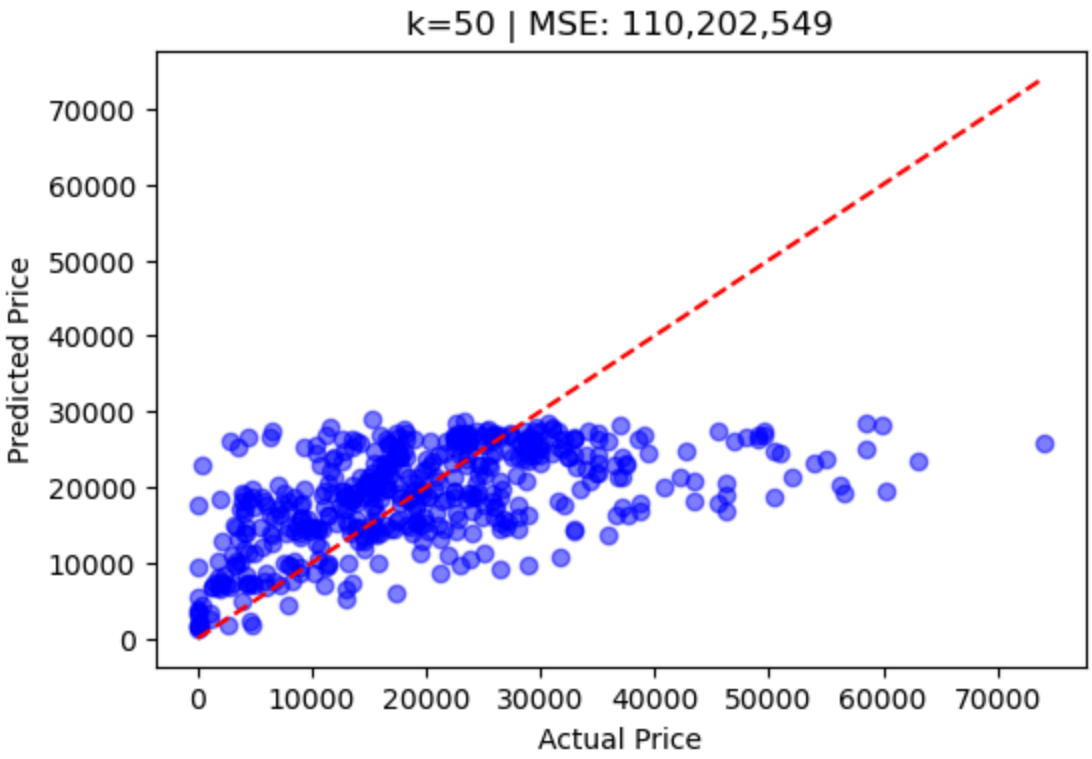

```
For k=3, Mean Squared Error is: 148,065,935.52
```



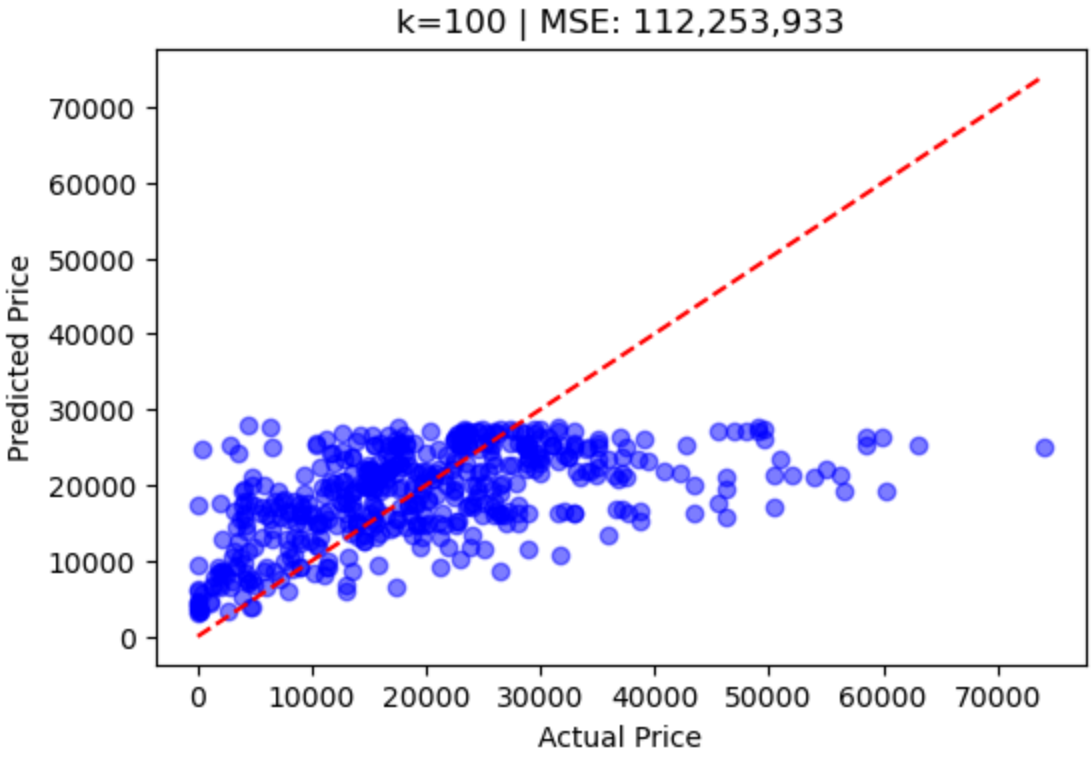

```
For k=10, Mean Squared Error is: 117,399,126.11
```



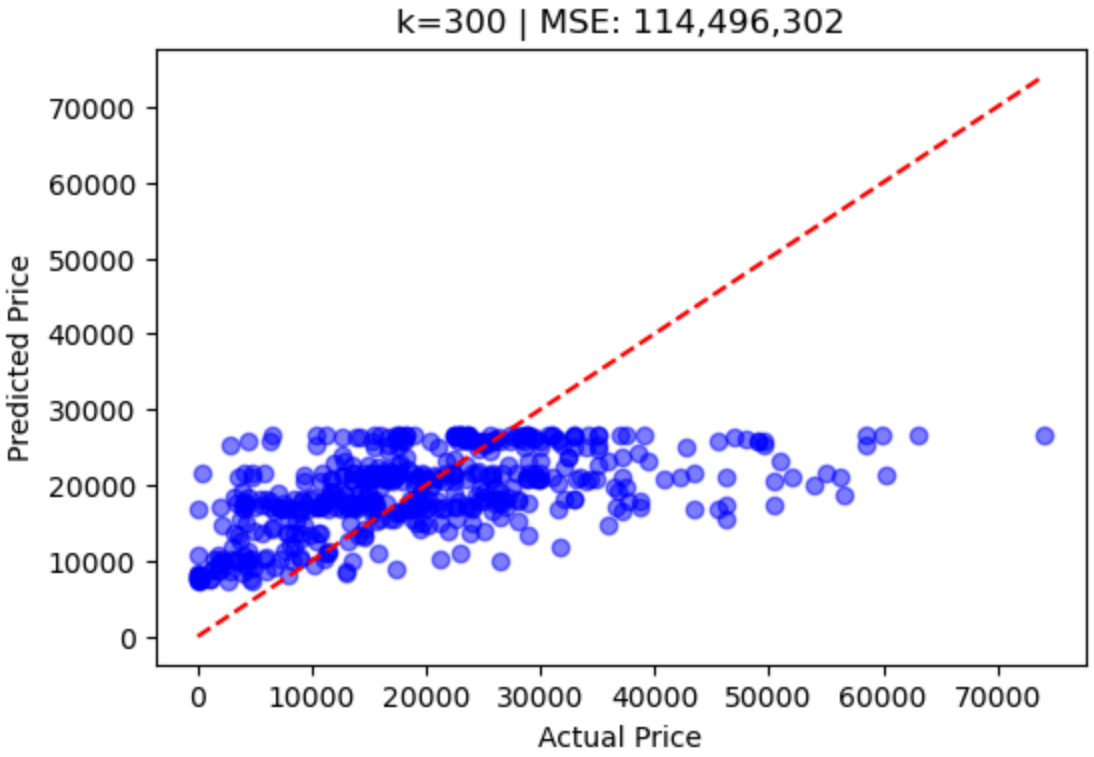

```
For k=25, Mean Squared Error is: 112,576,160.82
```



For k=50, Mean Squared Error is: 110,202,549.30



For k=100, Mean Squared Error is: 112,253,932.82




```
For k=300, Mean Squared Error is: 114,496,301.83
```

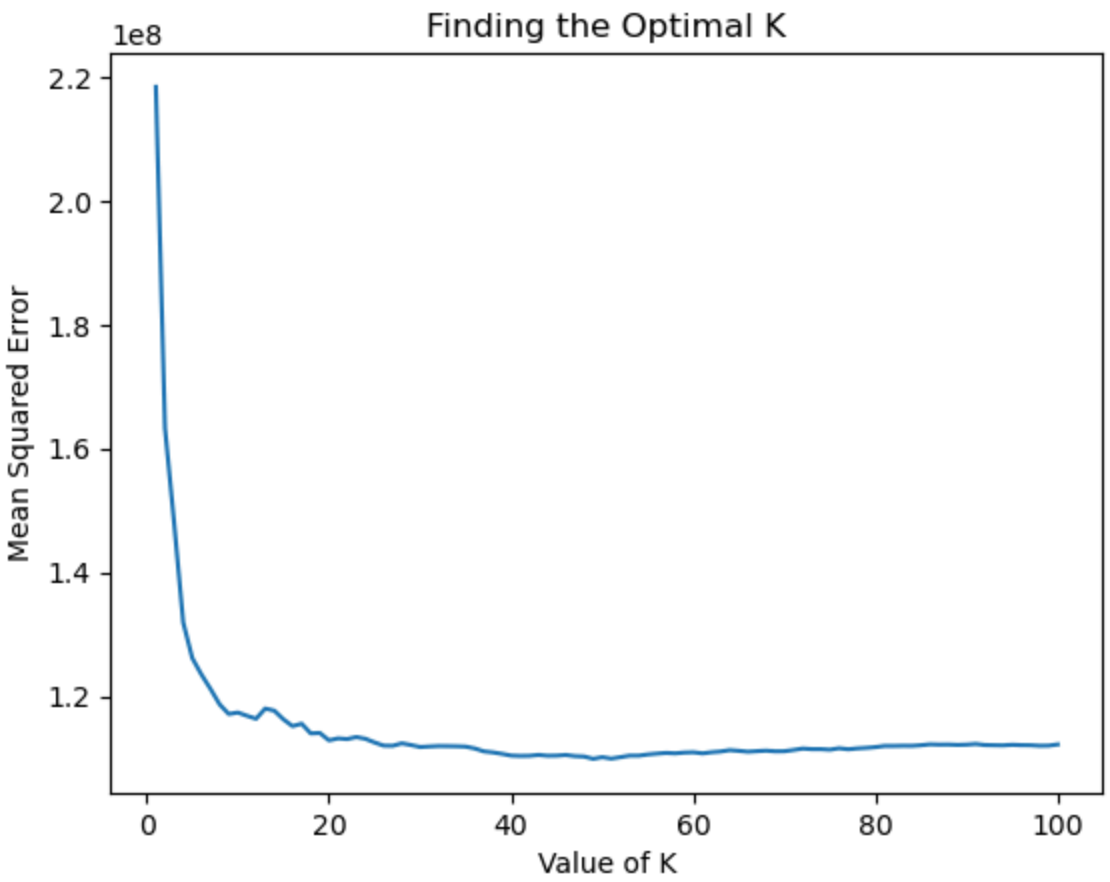
What patterns do you notice as you increase k?

As you increase k, the predicted prices become less scattered and start to flatten out toward the average price of all cars in the training set. This reduces the influence of individual weird or random data points, but can cause the model to ignore real price differences between very new or very old cars.

5. Determine the optimal k for these data.

In [17]:
```
import numpy as np

k_range = range(1, 101)
errors = []

for k in k_range:
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(X_train, y_train)
    mse = mean_squared_error(y_test, model.predict(X_test))
    errors.append(mse)

best_k = k_range[np.argmin(errors)]
print(f"The optimal k value is: {best_k}")

plt.plot(k_range, errors)
plt.xlabel('Value of K')
plt.ylabel('Mean Squared Error')
plt.title('Finding the Optimal K')
plt.show()
```

```
The optimal k value is: 49
```



6. Describe what happened in the plots of predicted versus actual prices as k varied, taking your answer into part 5 into account. (Hint: Use the words "underfitting" and "overfitting".)

At very low k values, the model is actually overfitting because it follows random noise in the training data too closely, making the scatter plots themselves look inaccurate. As k becomes very large, the models begins underfitting, by ignoring real patterns and simply predicting the average price for every car. The optimal k found in part 5 is that "sweet spot", where the plot lines up best with the actual prices without being too jumpy or too flat.

Q4. This question is a case study for k nearest neighbor regression, using the `heart_failure_clinical_records_dataset.csv` data.

The data for the question include:

- age: age of the patient (years)
- anaemia: decrease of red blood cells or hemoglobin (boolean)
- high blood pressure: if the patient has hypertension (boolean)
- creatinine phosphokinase (CPK): level of the CPK enzyme in the blood (mcg/L)
- diabetes: if the patient has diabetes (boolean)
- ejection fraction: percentage of blood leaving the heart at each contraction (percentage)
- platelets: platelets in the blood (kiloplatelets/mL)
- sex: woman or man (binary)
- serum creatinine: level of serum creatinine in the blood (mg/dL)
- serum sodium: level of serum sodium in the blood (mEq/L)
- smoking: if the patient smokes or not (boolean)
- time: follow-up period (days)
- death event: if the patient deceased during the follow-up period (boolean)

1. Load the `./data/heart_failure_clinical_records_dataset.csv`. Are there any `NA`'s to handle? use `.drop()` to remove `time` from the dataframe.
2. Make a correlation matrix. What variables are strongly associated with a death event?
3. For the dummy variables `anaemia`, `diabetes`, `high_blood_pressure`, `sex`, and `smoking`, compute a summary table of `DEATH_EVENT` grouped by the variable. For which variables does a higher proportion of the population die when the variable takes the value 1 rather than 0?
4. On the basis of your answers from 2 and 3, build a matrix X of the variables you think are most predictive of a death, and a variable y equal to `DEATH_EVENT`.
5. Maxmin normalize all of the variables in `X`.
6. Split the sample into ~80% for training and ~20% for evaluation. (Try to use the same train/test split for the whole question, so that you're comparing apples to apples in the questions below.).
7. Determine the optimal number of neighbors for a k-NN classification or regression for the variables you selected.
8. OK, do steps 5 through 7 again, but use all of the variables (except `time`). Which model has the best Mean Squared Error? Which would you prefer to use in practice, if you had to predict `DEATH_EVENT`s? If you play with the selection of variables, how

much does the RMSE change for your fitted model on the test data? Are more variables always better? Explain your findings.

Q5. This is a case study on k nearest neighbor classification, using the `animals.csv` data.

The data consist of a label, `class`, taking integer values 1 to 7, the name of the species, `animal`, and 16 characteristics of the animal, including `hair`, `feathers`, `milk`, `eggs`, `airborne`, and so on.

1. Load the data. For each of the seven class labels, print the values in the class and get a sense of what is included in that group. Perform some other EDA: How big are the classes? How much variation is there in each of the features/covariates? Which variables do you think will best predict which class?
2. Split the data 50/50 into training and test/validation sets. (The smaller the data are, the more equal the split should be. Otherwise, all of the members of one class end up in the training or test data, and the model falls apart.)
3. Using all of the variables, build a k-NN classifier. Explain how you select k.
4. Print a confusion matrix for the optimal model, comparing predicted and actual class label on the test set. How accurate it is? Can you interpret why mistakes are made across groups?
5. Use only `milk`, `aquatic`, and `airborne` to train a new k-NN classifier. Print your confusion table. Mine does not predict all of the classes, only a subset of them. To see the underlying proportions/probabilities, use `model.predict_proba(X_test.values)` to predict probabilities rather than labels for your `X_test` test data for your fitted `model`. Are all of the classes represented? Explain your results.

Q6. This is a case study using k nearest neighbor regression for imputation, using the `airbnb_hw.csv` data.

There are 30,478 observations, but only 22,155 ratings. We're going to build a kNN regressor to impute missing values. This is a common task, and illustrates one way you can use kNN in the future even when you have more advanced models available.

1. Load the `airbnb_hw.csv` data with Pandas. We're only going to use `Review Scores Rating`, `Price`, and `Beds`, so use `.loc` to column filter the dataframe to those variables.
2. Set use `.isnull()` and `.loc` to select the subset of the dataframe with missing review values. Set those aside in a different dataframe. We'll make predictions about them later.
3. Use `df = df.dropna(axis = 0, how = 'any')` to eliminate any observations with missing values/NA's from the dataframe.
4. For the complete cases, create a k-NN model that uses the variables `Price` and `Beds` to predict `Review Scores Rating`. How do you choose k? (Hint: Train/test split,

iterate over reasonable values of k and find a value that minimizes SSE on the test split using predictions from the training set.)

5. Predict the missing ratings.
6. Do a kernel density plot of the training ratings and the predicted missing ratings. Do they look similar or not? Describe what you see.

1. Load the airbnb_hw.csv data with Pandas. We're only going to use Review Scores Rating, Price, and Beds, so use .loc to column filter the dataframe to those variables.

In [19]:
```
df = pd.read_csv('airbnb_hw.csv')
df = df.loc[:, ['Review Scores Rating', 'Price', 'Beds']]

print("First 5 rows:")
print(df.head())
print(" ")
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
First 5 rows:
   Review Scores Rating Price  Beds
0                   NaN   145   1.0
1                   NaN    37   1.0
2                   NaN    28   1.0
3                   NaN   199   3.0
4                  96.0   549   3.0


Missing values per column:
Review Scores Rating    8323
Price                      0
Beds                      85
dtype: int64
```

2. Set use .isnull() and .loc to select the subset of the dataframe with missing review values. Set those aside in a different dataframe.

In [20]:
```
df_missing = df.loc[df['Review Scores Rating'].isnull()]

df_complete = df.loc[df['Review Scores Rating'].notnull()]

print(f"Rows to predict: {df_missing.shape[0]}")
print(f"Rows to train on: {df_complete.shape[0]}")
```

```
Rows to predict: 8323
Rows to train on: 22155
```

3. Use df = df.dropna(axis = 0, how = 'any') to eliminate any observations with missing values/NA's from the dataframe.

In [22]:
```
df_complete = df_complete.dropna(axis=0, how='any')
print(f"Cleaned training data size: {df_complete.shape[0]}")
```

```
Cleaned training data size: 22111
```

4. For the complete cases, create a k-NN model that uses the variables Price and Beds to predict Review Scores Rating.

In [24]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split

df = pd.read_csv('airbnb_hw.csv')
df = df.loc[:, ['Review Scores Rating', 'Price', 'Beds']]

df['Price'] = df['Price'].str.replace(',', '').astype(float)

df_complete = df.loc[df['Review Scores Rating'].notnull()].dropna(axis=0, how='any'
df_missing = df.loc[df['Review Scores Rating'].isnull()].dropna(subset=['Price', 'B

X = df_complete[['Price', 'Beds']]
y = df_complete['Review Scores Rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

k_values = range(1, 51)
sse_list = []

for k in k_values:
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    sse = np.sum((y_test - preds)**2)
    sse_list.append(sse)

best_k = k_values[np.argmin(sse_list)]
print(f"Best k: {best_k}")
```

```
Best k: 45
```

How do you choose k?

To find the best k, the code tests many different numbers of neighbors one by one to see which one makes the smallest mistakes. It calculates the SSE, which is a score that gets higher when the model's guesses are far away from the real ratings. By looking at the list of scores, the code picks the k that has lowest total error on the test data. This "best k" is the number that gives us the most accurate predictions for new or missing data.

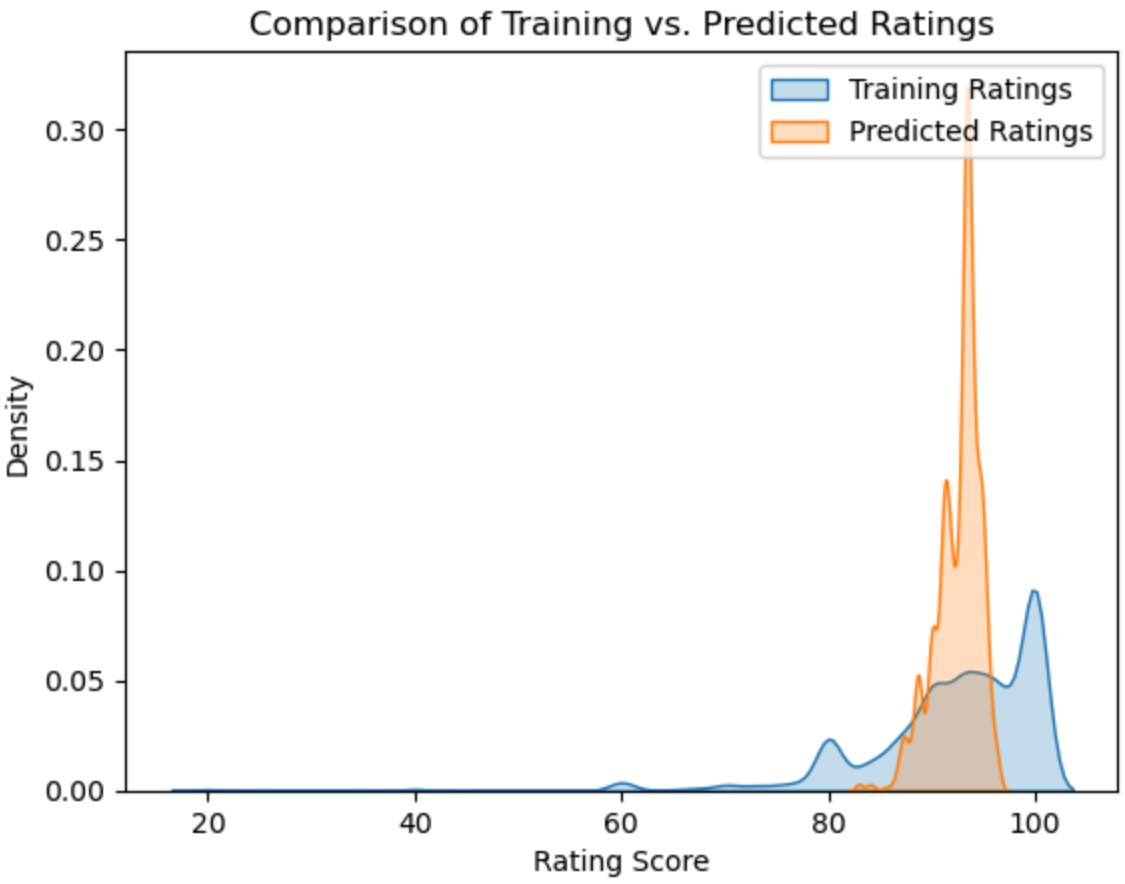
5. Predict the missing ratings.

In [26]:
```
final_model = KNeighborsRegressor(n_neighbors=best_k)
final_model.fit(X, y)
imputed_ratings = final_model.predict(df_missing[['Price', 'Beds']])

df_missing['Review Scores Rating'] = imputed_ratings

print(" ")
print(df_missing.head())
```

```
    Review Scores Rating  Price  Beds
0              94.577778  145.0   1.0
1              90.422222   37.0   1.0
2              85.444444   28.0   1.0
3              89.644444  199.0   3.0
13             92.111111   68.0   1.0
```

6. Do a kernel density plot of the training ratings and the predicted missing ratings.

In [27]:
```
sns.kdeplot(df_complete['Review Scores Rating'], label='Training Ratings', fill=Tru
sns.kdeplot(df_missing['Review Scores Rating'], label='Predicted Ratings', fill=Tru

plt.legend()
plt.title('Comparison of Training vs. Predicted Ratings')
plt.xlabel('Rating Score')
plt.show()
```



Do they look similar or not? Describe what you see.

The predicted ratings looks like a "skinnier" version of the training ratings. While the training data has many small bumps and contains outliers, the predicted data stays mainly in the middle because k-NN averages out extreme values. This means you will see a much taller peak at the most common rating and fewer "rare" scores at the very low or very high ends.

Q7. Let's do some very basic computer vision. We're going to import the MNIST handwritten digits data and use k-NN to predict values (i.e. "see/read").

1. To load the data, run the following code in a chunk:

```
from keras.datasets import mnist
df = mnist.load_data('minst.db')
train,test = df
X_train, y_train = train
X_test, y_test = test
```

The `y_test` and `y_train` vectors, for each index `i`, tell you want number is written in the corresponding index in `X_train[i]` and `X_test[i]`. The value of `X_train[i]` and `X_test[i]`, however, is a 28×28 array whose entries contain values between 0 and 256. Each element of the matrix is essentially a "pixel" and the matrix encodes a representation of a number. To visualize this, run the following code to see the first ten numbers:

```
import matplotlib.pyplot as plt
import numpy as np
np.set_printoptions(edgeitems=30, linewidth=100000)
for i in range(5):
    print(y_test[i],'\n') # Print the label
    print(X_test[i],'\n') # Print the matrix of values
    plt.contourf(np.rot90(X_test[i].transpose())) # Make a contour
plot of the matrix values
    plt.show()
```

OK, those are the data: Labels attached to handwritten digits encoded as a matrix.

2. What is the shape of `X_train` and `X_test`? What is the shape of `X_train[i]` and `X_test[i]` for each index `i`? What is the shape of `y_train` and `y_test`?
3. Use Numpy's `.reshape()` method to covert the training and testing data from a list of matrix into an vector of features. So, `X_test[index].reshape((1,784))` will convert the $index$-th element of `X_test` into a $28 \times 28 = 784$-length row vector of values, rather than a matrix. Turn `X_train` into an $N \times 784$ matrix X that is suitable for scikit-learn's kNN classifier where N is the number of observations and $784 = 28 * 28$ (you could use, for example, a `for` loop).
4. Use the reshaped `X_test` and `y_test` data to create a k-nearest neighbor classifier of digit. What is the optimal number of neighbors k? If you can't determine this, play

around with different values of k for your classifier.

5. For the optimal number of neighbors, how well does your predictor perform on the test set? Report the accuracy, compute a confusion matrix, and explain your findings.
6. For your confusion matrix, which mistakes are most likely? Do you find any interesting patterns?
7. So, this is how computers "see." They convert an image into a matrix of values, that matrix becomes a vector in a dataset, and then we deploy ML tools on it as if it was any other kind of tabular data. To make sure you follow this, invent a way to represent a color photo in matrix form, and then describe how you could convert it into tabular data. (Hint: RGB color codes provide a method of encoding a numeric value that represents a color.)