Question 1

1) Implement a KNN regressor using the scikit-learn conventations

In [6]:
```
from sklearn.datasets import load_diabetes,fetch_california_housing
from sklearn.base import BaseEstimator
from scipy.spatial import KDTree
class KnnRegressor(BaseEstimator):

    def __init__(self, k):
        self.k = k

    def fit(self, x, y):
        self.y_train_ = y
        self.x_train_kdtree_ = KDTree(x)
        return self

    def predict(self, x):
        _, neighbours = self.x_train_kdtree_.query(x, k=self.k)
        neighbours = neighbours.reshape(len(x), self.k)
        neighbour_labels = self.y_train_[neighbours]
        m = np.mean(neighbour_labels, axis=1)
        return m
```

To test your implementation, load the datasets diabetes and california housing through the functions load diabetes and fetch california housing, both of which are available in the module sklearn.datasets.

In [7]:
```
diabetes = load_diabetes()
california = fetch_california_housing()
```

In [8]:
```
import numpy as np

def train_test_split(x, y, train_size=0.6, random_state=None):
    RNG = np.random.default_rng(random_state)
    N = len(x)
    N_train = round(N*train_size)
    idx_train = RNG.choice(N, N_train, replace=False)
    idx_test = np.setdiff1d(np.arange(N), idx_train)
    RNG.shuffle(idx_test)
    return x[idx_train], x[idx_test], y[idx_train], y[idx_test]

x_train, x_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target,
x_train_cali, x_test_cali, y_train_cali, y_test_cali = train_test_split(california.d
```

In [16]:
```
def error_rate(y, y_hat):
    n = len(y)
    sum_of_errors = sum((y-y_hat)**2)
    return sum_of_errors/n
#finding error rate for diabtes data
knn = KnnRegressor(k=3)
knn.fit(x_train, y_train)
y_hat_train = knn.predict(x_train)
y_hat_test = knn.predict(x_test)
error_rate(y_train, y_hat_train), error_rate(y_test, y_hat_test)
```

Out[16]: (2352.791614255766, 4318.048336472066)

In [17]:
```
#finding error rate for california data data
knn = KnnRegressor(k=3)
knn.fit(x_train_cali, y_train_cali)
y_hat_train_cal = knn.predict(x_train_cali)
y_hat_test_cal = knn.predict(x_test_cali)
error_rate(y_train_cali, y_hat_train_cal), error_rate(y_test_cali, y_hat_test_cal)
```

Out[17]:
```
(0.5716849988297835, 1.2571807930454773)
```

In [19]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(3)
knn.fit(x_train_cali, y_train_cali)
y_hat_train_cal = knn.predict(x_train_cali)
y_hat_test_cal = knn.predict(x_test_cali)
error_rate(y_train_cali, y_hat_train_cal), error_rate(y_test_cali, y_hat_test_cal)
```

Out[19]:
```
(0.5716849988297835, 1.2571807930454773)
```

Question 2: L - Fold Cross Validation

In [20]:
```
class LFold:
    def __init__(self, n_splits,shuffle=None):
        self.n_splits = n_splits
        self.shuffle = shuffle
    def get_n_splits(self, x=None, y=None, groups=None):
        return self.n_splits
    def split(self, x, y=None, groups=None):
        n_samples = np.shape(x)[0]
        indices = np.arange(n_samples)
        if self.shuffle:
            indices = np.random.permutation(indices)

        fold_size = n_samples // self.n_splits
        for i in range(self.n_splits):
            start = i * fold_size
            end = (i + 1) * fold_size
            test_idx = indices[start:end]
            train_idx = np.concatenate([indices[:start], indices[end:]])
            yield train_idx, test_idx
```

In [56]:
```
for train_idx, test_idx in LFold(5,shuffle=False,random_state=101).split(list(range(
    print(train_idx, test_idx)
```

```
[ 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21] [0 1 2 3]
[ 0  1  2  3  8  9 10 11 12 13 14 15 16 17 18 19 20 21] [4 5 6 7]
[ 0  1  2  3  4  5  6  7 12 13 14 15 16 17 18 19 20 21] [ 8  9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11 16 17 18 19 20 21] [12 13 14 15]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 20 21] [16 17 18 19]
```

In [13]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

def knn_parameter_search(X, y, k_range, n_splits=5):  #defining number of splits
    results = []  #intializing llst
    for k in k_range:
        mse_train = [] #intializing llst
        mse_test = [] #intializing llst
        lfold = LFold(n_splits=n_splits, shuffle=True)
        for train_idx, test_idx in lfold.split(X, y):
            X_train, X_test = X[train_idx], X[test_idx]
```

```
            y_train, y_test = y[train_idx], y[test_idx]
            knn = KnnRegressor(k)    #intializing model
            knn.fit(X_train, y_train)  #fitting model
            y_train_pred = knn.predict(X_train)  #making predictions
            y_test_pred = knn.predict(X_test)   #making predictions
            mse_train.append(mean_squared_error(y_train, y_train_pred))  # calculati
            mse_test.append(mean_squared_error(y_test, y_test_pred))
        mean_mse_train = np.mean(mse_train)  #calculating mean mse for each k
        std_mse_train = np.std(mse_train)
        mean_mse_test = np.mean(mse_test)
        std_mse_test = np.std(mse_test)
        results.append({
            'k': k,
            'mean_mse_train': mean_mse_train,
            'std_mse_train': std_mse_train,
            'mean_mse_test': mean_mse_test,
            'std_mse_test': std_mse_test,
            "n_splits" : n_splits
        })
    best_k_result = min(results, key=lambda x: x['mean_mse_test'])   #getting the be
    return results, best_k_result

k_range = range(1, 51)
results1, best_k1 = knn_parameter_search(california.data, california.target, k_range
print("Best K for dataset 1:", best_k1['k'])
print("Mean Test MSE for best K:", best_k1['mean_mse_test'])
print("Standard Deviation of Test MSE for best K:", best_k1['std_mse_test'])

results2, best_k2 = knn_parameter_search(california.data, california.target, k_range
print("Best K for california:", best_k2['k'])
print("Mean Test MSE for best K:", best_k2['mean_mse_test'])
print("Standard Deviation of Test MSE for best K:", best_k2['std_mse_test'])
```

```
Best K for dataset 1: 14
Mean Test MSE for best K: 3202.4174628942483
Standard Deviation of Test MSE for best K: 637.5819855519782
```

The above code uses the self defined code and takes longer to run so another code has been added that using skikitlearn functions which are faster to run

In [25]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
import numpy as np

def knn_parameter_search(X, y, k_range, n_splits=5, random_state=None):
    results = []
    for k in k_range:
        mse_train = []
        mse_test = []
        lfold = KFold(n_splits=n_splits, shuffle=True)
        for train_idx, test_idx in lfold.split(X, y):
            X_train, X_test = X[train_idx], X[test_idx]
            y_train, y_test = y[train_idx], y[test_idx]
            knn = KNeighborsRegressor(n_neighbors=k)
            knn.fit(X_train, y_train)
            y_train_pred = knn.predict(X_train)  #making predictions
            y_test_pred = knn.predict(X_test)   #making predictions
            mse_train.append(mean_squared_error(y_train, y_train_pred))  # calculati
            mse_test.append(mean_squared_error(y_test, y_test_pred))
        mean_mse_train = np.mean(mse_train)  #calculating mean mse for each k
        std_mse_train = np.std(mse_train)
        mean_mse_test = np.mean(mse_test)
        std_mse_test = np.std(mse_test)
```

```
        results.append({
            'k': k,
            'mean_mse_train': mean_mse_train,
            'std_mse_train': std_mse_train,
            'mean_mse_test': mean_mse_test,
            'std_mse_test': std_mse_test,
            "n_splits" : n_splits
        })
    best_k_result = min(results, key=lambda x: x['mean_mse_test'])   #getting the be
    return results, best_k_result

k_range = range(1, 51)
results1, best_k1 = knn_parameter_search(diabetes.data, diabetes.target, k_range, n_
print("Best K for diabetes:", best_k1['k'])
print("Mean Test MSE for best K:", best_k1['mean_mse_test'])
print("Standard Deviation of Test MSE for best K:", best_k1['std_mse_test'])

results2, best_k2 = knn_parameter_search(california.data, california.target, k_range
print("Best K for california:", best_k2['k'])
print("Mean Test MSE for best K:", best_k2['mean_mse_test'])
print("Standard Deviation of Test MSE for best K:", best_k2['std_mse_test'])
```

```
Best K for diabetes: 16
Mean Test MSE for best K: 3229.696232703173
Standard Deviation of Test MSE for best K: 444.0201696115252
Best K for california: 7
Mean Test MSE for best K: 1.1132086274557107
Standard Deviation of Test MSE for best K: 0.01825456048460864
```

In [101…

```
results1
```

Out[101…

```
[{'k': 1,
  'mean_mse_train': 0.0,
  'std_mse_train': 0.0,
  'mean_mse_test': 6232.713636363636,
  'std_mse_test': 1571.6054026216589},
 {'k': 2,
  'mean_mse_train': 1487.481638418079,
  'std_mse_train': 118.8184009438168,
  'mean_mse_test': 4520.620454545455,
  'std_mse_test': 685.7386721065043},
 {'k': 3,
  'mean_mse_train': 2033.5202762084118,
  'std_mse_train': 101.8831684210057,
  'mean_mse_test': 3869.246212121213,
  'std_mse_test': 320.2069392127664},
 {'k': 4,
  'mean_mse_train': 2289.7313559322038,
  'std_mse_train': 74.0212535923737,
  'mean_mse_test': 3749.154403409091,
  'std_mse_test': 421.49362708295365},
 {'k': 5,
  'mean_mse_train': 2433.3196836158195,
  'std_mse_train': 108.66439831157948,
  'mean_mse_test': 3617.356272727273,
  'std_mse_test': 660.2071745702877},
 {'k': 6,
  'mean_mse_train': 2571.5554613935974,
  'std_mse_train': 87.8410786919071,
  'mean_mse_test': 3576.022601010101,
  'std_mse_test': 422.9847423636179},
 {'k': 7,
  'mean_mse_train': 2634.0579153695376,
  'std_mse_train': 201.4795439346637,
  'mean_mse_test': 3449.206771799628,
  'std_mse_test': 847.8748336971994},
 {'k': 8,
```

```
 'mean_mse_train': 2682.0830861581917,
 'std_mse_train': 69.55577788485503,
 'mean_mse_test': 3423.274680397727,
 'std_mse_test': 308.5105215923481},
{'k': 9,
 'mean_mse_train': 2774.544151496129,
 'std_mse_train': 115.39141609636708,
 'mean_mse_test': 3439.177918069585,
 'std_mse_test': 455.6205639980049},
{'k': 10,
 'mean_mse_train': 2804.687474576271,
 'std_mse_train': 105.08112481624488,
 'mean_mse_test': 3419.913590909091,
 'std_mse_test': 347.2201609011447},
{'k': 11,
 'mean_mse_train': 2808.940360461316,
 'std_mse_train': 39.11374638848341,
 'mean_mse_test': 3493.7462434259955,
 'std_mse_test': 209.59884523612104},
{'k': 12,
 'mean_mse_train': 2814.1654425612055,
 'std_mse_train': 71.54241791298318,
 'mean_mse_test': 3308.1056660353543,
 'std_mse_test': 97.11478607537039},
{'k': 13,
 'mean_mse_train': 2834.886032828536,
 'std_mse_train': 91.2277757686463,
 'mean_mse_test': 3334.725053792362,
 'std_mse_test': 307.4823336133376},
{'k': 14,
 'mean_mse_train': 2875.797728582959,
 'std_mse_train': 143.23354146372415,
 'mean_mse_test': 3249.9875463821895,
 'std_mse_test': 609.0294134330908},
{'k': 15,
 'mean_mse_train': 2879.1592266164466,
 'std_mse_train': 121.85594845880088,
 'mean_mse_test': 3231.121393939394,
 'std_mse_test': 371.2924928437827},
{'k': 16,
 'mean_mse_train': 2901.9680239230224,
 'std_mse_train': 100.73062105828589,
 'mean_mse_test': 3266.868394886364,
 'std_mse_test': 281.8686956823103},
{'k': 17,
 'mean_mse_train': 2897.323853928411,
 'std_mse_train': 118.43189067893789,
 'mean_mse_test': 3332.537716262976,
 'std_mse_test': 481.07367624906107},
{'k': 18,
 'mean_mse_train': 2927.8147049591967,
 'std_mse_train': 55.74497531425312,
 'mean_mse_test': 3205.595173961841,
 'std_mse_test': 243.92834465934476},
{'k': 19,
 'mean_mse_train': 2935.8520603471216,
 'std_mse_train': 82.48931584613207,
 'mean_mse_test': 3296.9503147821715,
 'std_mse_test': 352.2133229731162},
{'k': 20,
 'mean_mse_train': 2957.748382768362,
 'std_mse_train': 96.96067455664112,
 'mean_mse_test': 3241.3609943181814,
 'std_mse_test': 473.9605809167786},
{'k': 21,
 'mean_mse_train': 2970.162424894628,
 'std_mse_train': 111.17614360350996,
 'mean_mse_test': 3331.802159348588,
 'std_mse_test': 172.42277104635227},
```

```
{'k': 22,
 'mean_mse_train': 2987.9835761311106,
 'std_mse_train': 155.55647890967953,
 'mean_mse_test': 3276.1895144628097,
 'std_mse_test': 474.80038409057124},
{'k': 23,
 'mean_mse_train': 3006.393266262963,
 'std_mse_train': 50.476999528265814,
 'mean_mse_test': 3308.600850661625,
 'std_mse_test': 342.241556696924},
{'k': 24,
 'mean_mse_train': 3019.6202497253607,
 'std_mse_train': 63.662462586414335,
 'mean_mse_test': 3267.48948074495,
 'std_mse_test': 261.05296869267715},
{'k': 25,
 'mean_mse_train': 3039.667879774011,
 'std_mse_train': 69.24950261928967,
 'mean_mse_test': 3314.2272836363636,
 'std_mse_test': 334.13644469619794},
{'k': 26,
 'mean_mse_train': 3051.2012837227962,
 'std_mse_train': 48.94762389571196,
 'mean_mse_test': 3294.865092119419,
 'std_mse_test': 130.15863041302066},
{'k': 27,
 'mean_mse_train': 3063.1803422380317,
 'std_mse_train': 122.78993352944406,
 'mean_mse_test': 3324.754489337823,
 'std_mse_test': 658.3671887451478},
{'k': 28,
 'mean_mse_train': 3084.516768275107,
 'std_mse_train': 64.49581221945805,
 'mean_mse_test': 3350.3824762291274,
 'std_mse_test': 332.46334253806305},
{'k': 29,
 'mean_mse_train': 3104.3133886884725,
 'std_mse_train': 119.884775786941,
 'mean_mse_test': 3282.1769511404173,
 'std_mse_test': 506.80333144464646},
{'k': 30,
 'mean_mse_train': 3102.937148148148,
 'std_mse_train': 155.75507597771207,
 'mean_mse_test': 3414.457707070707,
 'std_mse_test': 473.48557322616824},
{'k': 31,
 'mean_mse_train': 3147.448772171173,
 'std_mse_train': 117.7468278664487,
 'mean_mse_test': 3300.8287437328545,
 'std_mse_test': 415.3932015161917},
{'k': 32,
 'mean_mse_train': 3123.3057341322383,
 'std_mse_train': 114.70301907462384,
 'mean_mse_test': 3412.5051513671874,
 'std_mse_test': 426.27442035624205},
{'k': 33,
 'mean_mse_train': 3156.1811338863727,
 'std_mse_train': 99.74944528689748,
 'mean_mse_test': 3307.926183320812,
 'std_mse_test': 391.5557867879551},
{'k': 34,
 'mean_mse_train': 3151.613961546732,
 'std_mse_train': 74.62556908959523,
 'mean_mse_test': 3373.730567788613,
 'std_mse_test': 325.60215102927907},
{'k': 35,
 'mean_mse_train': 3172.7007014873743,
 'std_mse_train': 79.57077184809917,
 'mean_mse_test': 3329.9352949907234,
```

```
 'std_mse_test': 144.62148667597236},
{'k': 36,
 'mean_mse_train': 3177.8196942351956,
 'std_mse_train': 121.83013064824502,
 'mean_mse_test': 3326.8840049803594,
 'std_mse_test': 337.04962732223044},
{'k': 37,
 'mean_mse_train': 3159.276539847222,
 'std_mse_train': 64.9902799596884,
 'mean_mse_test': 3442.237059233681,
 'std_mse_test': 163.97157364118144},
{'k': 38,
 'mean_mse_train': 3176.6407073102027,
 'std_mse_train': 93.71837055396931,
 'mean_mse_test': 3325.4528944220597,
 'std_mse_test': 285.35638984812863},
{'k': 39,
 'mean_mse_train': 3211.328494485861,
 'std_mse_train': 92.52731620660998,
 'mean_mse_test': 3303.6908747235675,
 'std_mse_test': 370.9683378175694},
{'k': 40,
 'mean_mse_train': 3215.5397814265534,
 'std_mse_train': 95.89872684307036,
 'mean_mse_test': 3356.7135738636366,
 'std_mse_test': 485.6716534838466},
{'k': 41,
 'mean_mse_train': 3217.318852781335,
 'std_mse_train': 102.7775575024354,
 'mean_mse_test': 3422.160464550322,
 'std_mse_test': 440.3462459208854},
{'k': 42,
 'mean_mse_train': 3221.492000397146,
 'std_mse_train': 50.36739392264093,
 'mean_mse_test': 3390.7989344980415,
 'std_mse_test': 214.6241067703686},
{'k': 43,
 'mean_mse_train': 3222.483215541765,
 'std_mse_train': 13.797375232662716,
 'mean_mse_test': 3445.191125424062,
 'std_mse_test': 220.67725698142456},
{'k': 44,
 'mean_mse_train': 3235.8219259233315,
 'std_mse_train': 60.147828287650306,
 'mean_mse_test': 3423.327735255447,
 'std_mse_test': 253.40213740344345},
{'k': 45,
 'mean_mse_train': 3249.3443261491248,
 'std_mse_train': 126.6785072322872,
 'mean_mse_test': 3323.6813187429857,
 'std_mse_test': 410.29253939510284},
{'k': 46,
 'mean_mse_train': 3233.322392212148,
 'std_mse_train': 83.53409869369561,
 'mean_mse_test': 3451.6142893968035,
 'std_mse_test': 472.3176294822649},
{'k': 47,
 'mean_mse_train': 3238.7257736072,
 'std_mse_train': 76.16441827010412,
 'mean_mse_test': 3391.422548253015,
 'std_mse_test': 432.9050542039892},
{'k': 48,
 'mean_mse_train': 3237.7549560087095,
 'std_mse_train': 69.84597020711524,
 'mean_mse_test': 3385.387606534091,
 'std_mse_test': 205.16244899382767},
{'k': 49,
 'mean_mse_train': 3253.9323191608014,
 'std_mse_train': 35.56006103705857,
```

```
 'mean_mse_test': 3407.3489568740297,
 'std_mse_test': 351.7246460215051},
{'k': 50,
 'mean_mse_train': 3246.6699943502826,
 'std_mse_train': 127.51350229110584,
 'mean_mse_test': 3377.5057209090905,
 'std_mse_test': 607.5373885792384}]
```

The effects of parameter K:

1. Underfitting: When the value of K is really large then the model becomes less flexible and tends to underfit to the data. This is seen when the mean test error is high because the model's perdictions are overly smooth and it fails to capture locak patterns in the data.
2. Overfitting: When the value of K is very small like 1,2,3 then the model becomes highly sentitive to noise and individual data points which leads to overfitting. This is due to the fact that the model fits the training data too closely resulting in poor generalization to new data. This is evident as the mean test error starts increasing.

Effects of the parameter L:

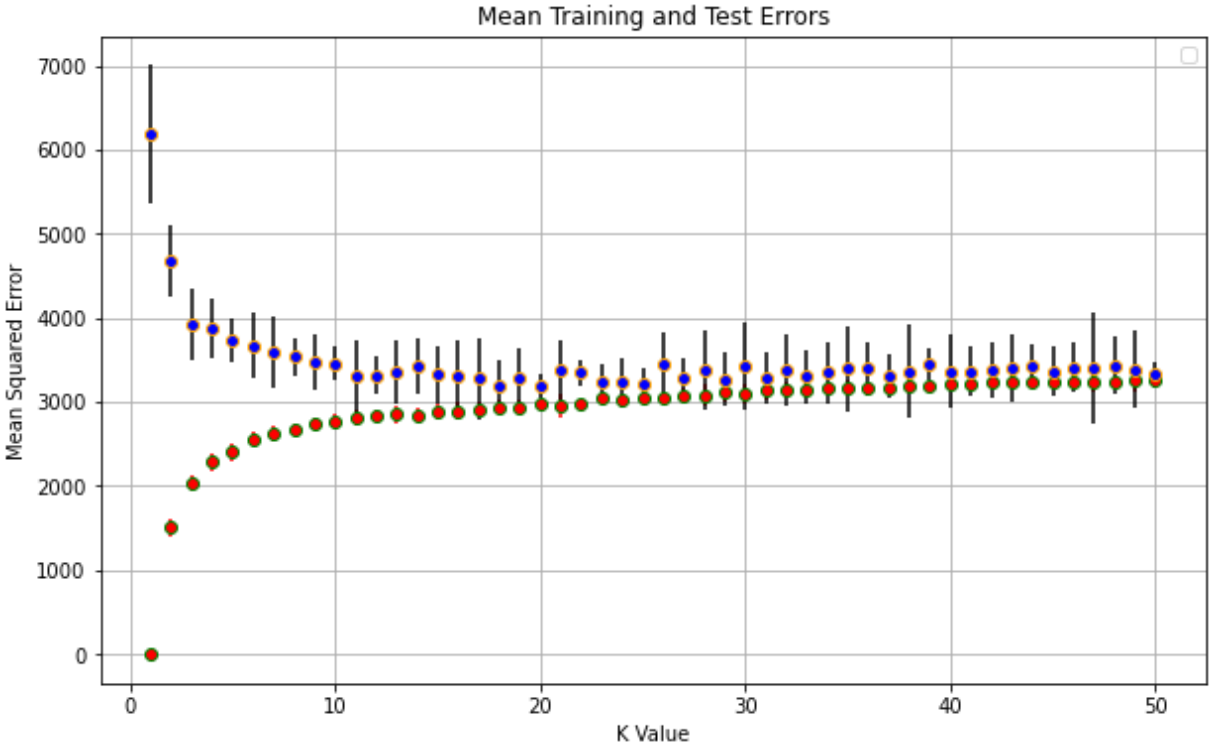
1. Low L: When the value of number of folds is small there is less variability in the cross validation procedure. This can reslt in a more stable estimate of the model performance but may have high bias. The error bars which show the standard error of the mean, are narrower which indicates lower uncertainty.
2. High L: With a larger number of folds there is more variance in the cross-validation procedure. This can lead to wider error bars indicating higher uncertainity in the estimated errors. However it provides a more accurate estimate of the model's generalization performance.

In [26]:
```
import matplotlib.pyplot as plt
def plot_knn_results(results,l):
    plt.figure(figsize=(16, 8))
    for result in results:
        k_values = result['k']
        mean_mse_train = result['mean_mse_train']
        mean_mse_test = result['mean_mse_test']
        s = result['std_mse_test']
        ste = [1.96 * s / (l ** 0.5)]
        s1 = result['std_mse_train']
        ste1 = [1.96 * s1 / (l ** 0.5)]
        plt.errorbar(
                    k_values,
                    mean_mse_train,
                    yerr=ste1,
                    marker='o',
                    ecolor = 'Red',mfc='red',
         mec='green')
        plt.errorbar(
                    k_values,
                    mean_mse_test,
                    yerr=ste,
                    marker='o',
                    ecolor = 'black',mfc='blue',
         mec='orange')
    plt.title(f'Mean Training and Test Errors')
    plt.xlabel('K Value')
    plt.ylabel('Mean Squared Error')
    plt.legend()
```

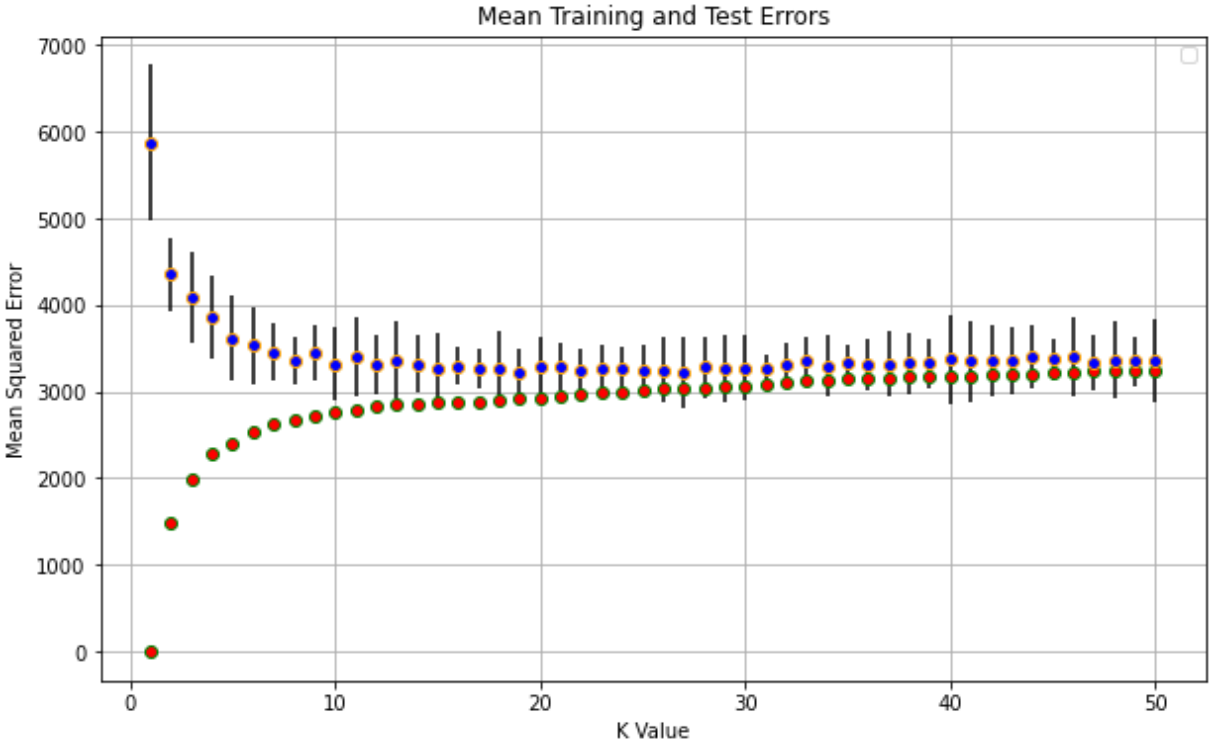
```
    plt.grid(True)
    plt.show()
```

In [34]:
```
l_values = [5,10,15]
for l in l_values:
    results1, best_k1 = knn_parameter_search(diabetes.data, diabetes.target, k_range
    plot_knn_results(results1, dataset_name="diabetes data", l=l)
```

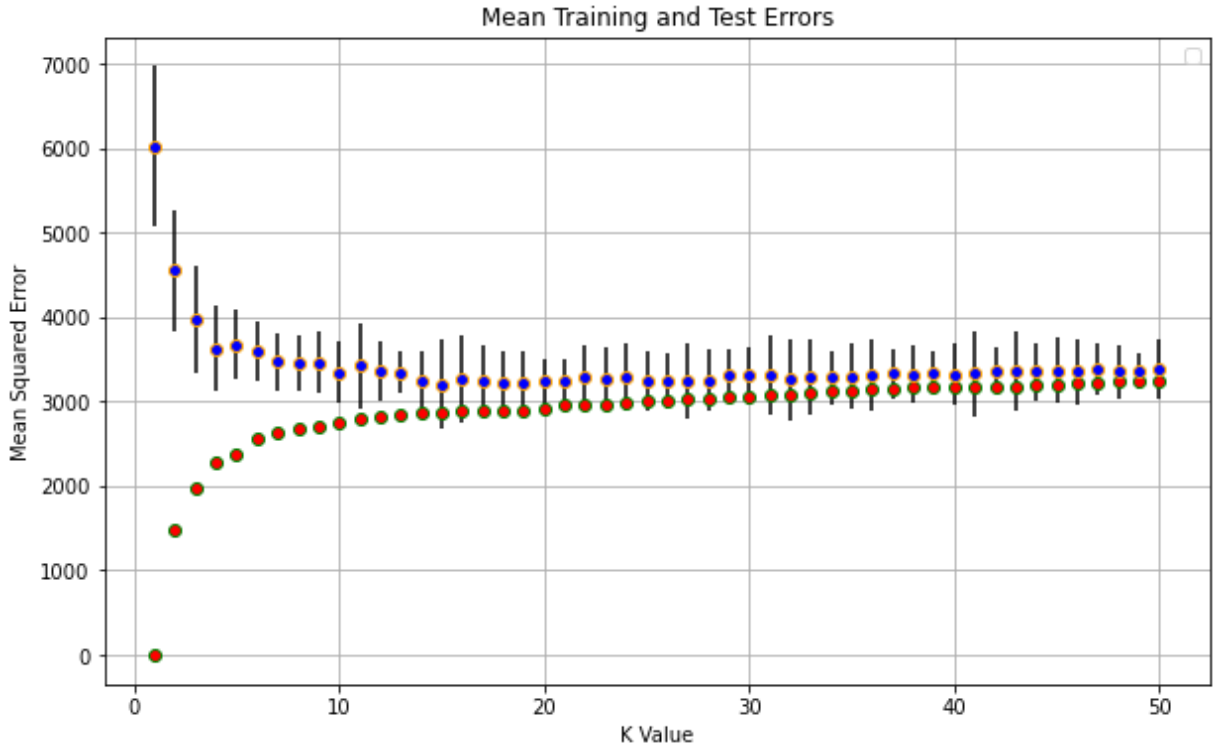
```
No handles with labels found to put in legend.
```




```
No handles with labels found to put in legend.
```

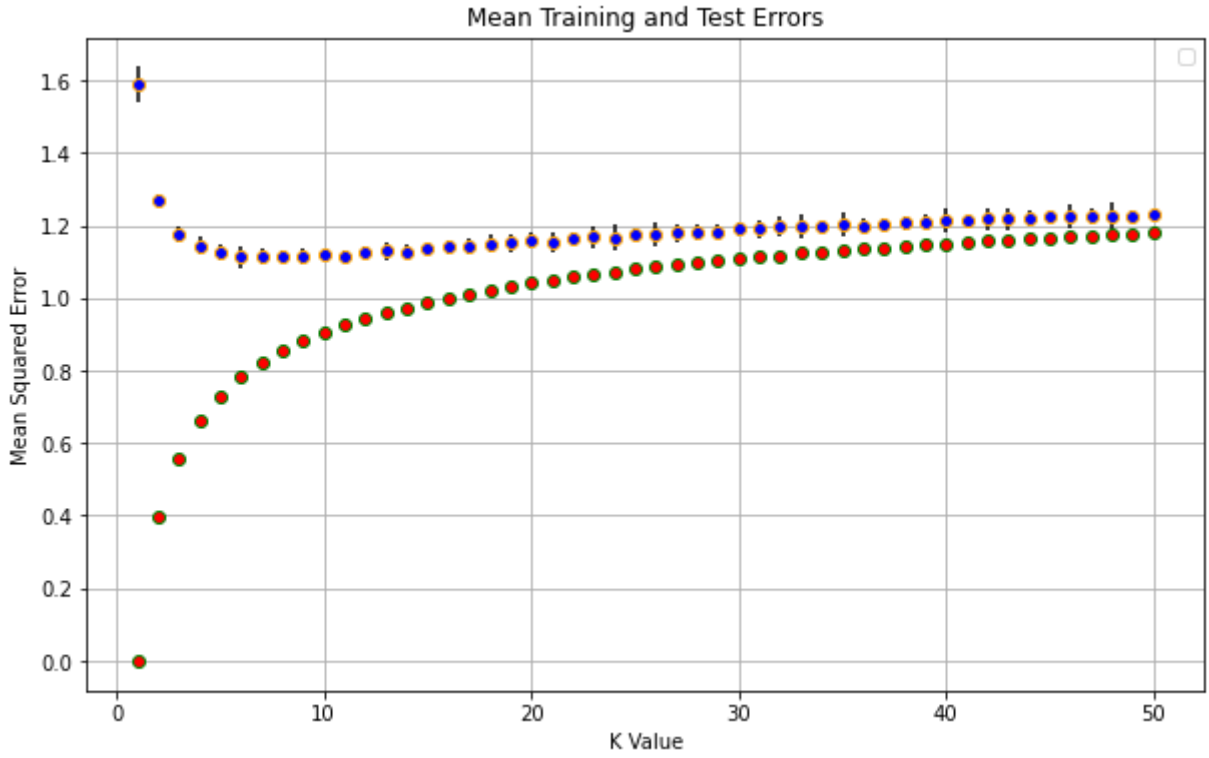



```
No handles with labels found to put in legend.
```

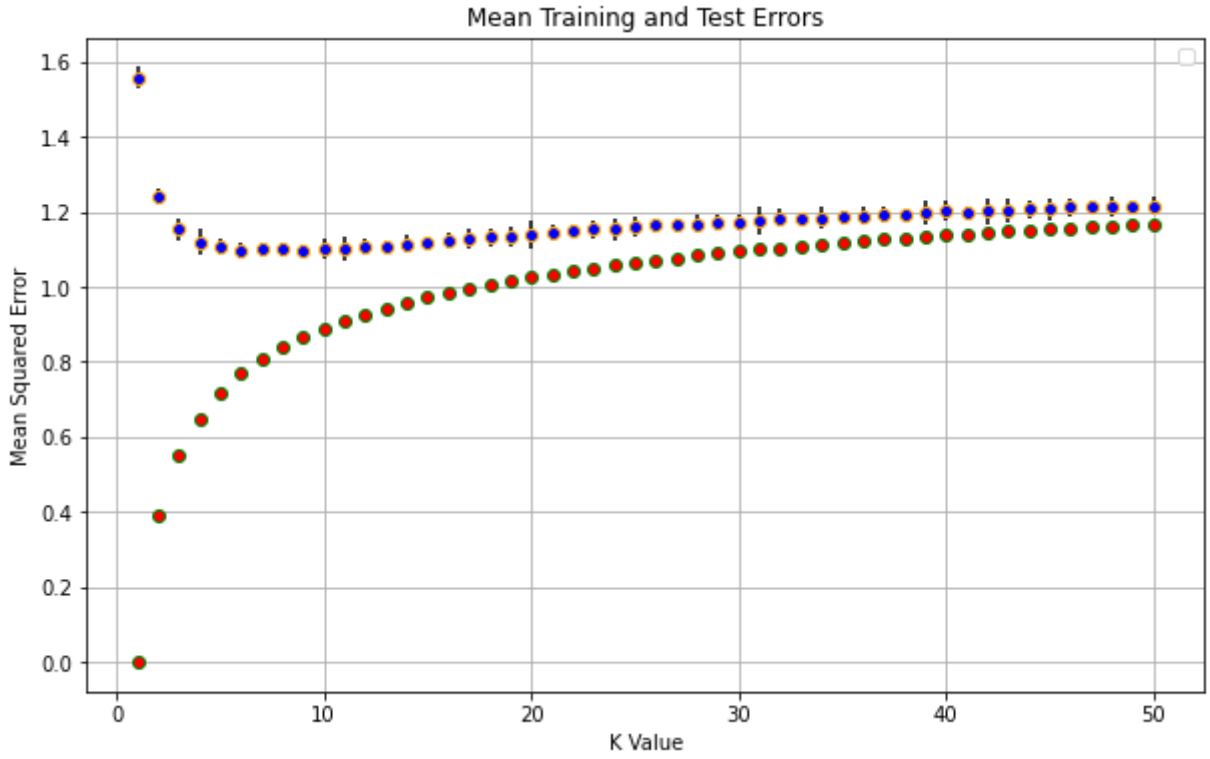


In [46]:
```
l_values = [5,10]
for l in l_values:
    results2, best_k2 = knn_parameter_search(california.data, california.target, k_r
    plot_knn_results(results2, dataset_name="california data", l=l)
```

```
No handles with labels found to put in legend.
```




```
No handles with labels found to put in legend.
```



Question 3 Nested CV

In [37]:
```
from sklearn.base import BaseEstimator
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
class KnnRegressorCV(BaseEstimator):
    def __init__(self, ks=list(range(1, 21)), cv=KFold(5)):
        self.ks = ks
        self.cv = cv
        self.best_k_ = None
        self.model_ = None

    def fit(self, X, y):
        best_score = float('-inf')
        for ki in self.ks:
            knn = KNeighborsRegressor(n_neighbors=ki)
            scores = cross_val_score(knn, X, y, cv=self.cv, scoring='neg_mean_square
            mean_score = scores.mean()
            if mean_score > best_score:
                best_score = mean_score
                self.best_k_ = ki

        self.model_ = KNeighborsRegressor(n_neighbors=self.best_k_)
        self.model_.fit(X, y)
        return self

    def predict(self, X):
        return self.model_.predict(X)
```

In [38]:
```
k1 = KnnRegressorCV()
```

In [40]:
```
X = diabetes.data
y= diabetes.target
```

In [41]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

outer_cv = KFold(n_splits=5)
selected_k_values = []

for train_index, test_index in outer_cv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    knn_cv = KnnRegressorCV(ks=list(range(1, 21)), cv=LFold(5))
    knn_cv.fit(X_train, y_train)
    selected_k = knn_cv.best_k_
    selected_k_values.append(selected_k)
    y_pred = knn_cv.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean squared error for outer test set: {mse}")

mean_selected_k = sum(selected_k_values) / len(selected_k_values)

print(f"Mean selected K value across outer cross-validation splits: {mean_selected_k
```

```
Mean squared error for outer test set: 3365.91468997087
Mean squared error for outer test set: 3239.704055052292
Mean squared error for outer test set: 3518.0419823232323
Mean squared error for outer test set: 2848.7391868512113
Mean squared error for outer test set: 3084.4355955678675
Mean selected K value across outer cross-validation splits: 16.6
```

In [42]:
```
selected_k_values
```

Out[42]:
```
[18, 17, 12, 17, 19]
```

In [43]:
```
X = california.data
y= california.target
```

In [44]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

outer_cv = KFold(n_splits=5)

selected_k_values = []

for train_index, test_index in outer_cv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    knn_cv = KnnRegressorCV(ks=list(range(1, 21)), cv=LFold(5))
    knn_cv.fit(X_train, y_train)
    selected_k = knn_cv.best_k_
    selected_k_values.append(selected_k)
    y_pred = knn_cv.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean squared error for outer test set: {mse}")

mean_selected_k = sum(selected_k_values) / len(selected_k_values)
print(f"Mean selected K value across outer cross-validation splits: {mean_selected_k
```

```
Mean squared error for outer test set: 1.0985673983935011
Mean squared error for outer test set: 1.034657332348302
Mean squared error for outer test set: 1.4045020207720065
Mean squared error for outer test set: 1.1875299594976432
Mean squared error for outer test set: 1.37360032561288
Mean selected K value across outer cross-validation splits: 9.6
```

In [45]:
```
selected_k_values
```

Out[45]:
```
[9, 10, 11, 8, 10]
```

Factors determine whether the internal cross-validation procedure is successful in approximately selecting the best model

1. Size and Quality of the Dataset: A larger dataset provides more reliable estimates of model performance and it becomes easier to identify the best hyperparameters. High-quality data also contributes to better model selection.
2. Choice of K Range: The range of K values tested in the inner cross-validation loop is critical. If the range is too narrow or doesn't include the true optimal K the procedure may not select the best model.
3. Number of Inner Folds (L in your previous questions): The number of inner cross-validation folds can impact the reliability of hyperparameter selection. Smaller values may result in more variability while larger values increase computational cost.
4. Data Variability: If the dataset is highly variable it may be challenging to find a single optimal K value that works well for all data partitions.
5. Model Complexity: The complexity of the underlying model (e.g., KNN with different K values) can affect the success of the procedure. Simpler models may be less sensitive to hyperparameter choices.
6. Randomness: The outcome of the cross-validation procedure can have an element of randomness due to the random splitting of data into folds. Repeating the process multiple times and observing consistent results can increase confidence in the chosen K value.

In [ ]:
```
```