1.

From the problem, we know that

$\hat{W}_{150} = \frac{80+78+65}{3} \approx 74.33\,kg$.

$\hat{W}_{155} = \frac{80+78+65}{3} \approx 74.33\,kg$.

$\hat{W}_{165} = \frac{78+80+83}{3} \approx 80.33\,kg$.

$\hat{W}_{190} = \frac{83+80+100}{3} \approx 87.67\,kg$.

2.

$\hat{W}_{150} = 65\,kg$. (Since $d_1 = 0$, $W_1 \to \infty$)

$\hat{W}_{155} = \frac{65\times\frac{1}{5} + 78\times\frac{1}{13} + 80\times\frac{1}{16}}{\frac{1}{5}+\frac{1}{13}+\frac{1}{16}} \approx 70.71\,kg$.

$\hat{W}_{165} = \frac{78\times\frac{1}{3} + 80\times\frac{1}{6} + 83\times\frac{1}{13}}{\frac{1}{3}+\frac{1}{6}+\frac{1}{13}} \approx 79.24\,kg$.

$\hat{W}_{190} = \frac{100\times 1 + 80\times\frac{1}{8} + \frac{1}{12}\times 83}{1+\frac{1}{8}+\frac{1}{12}} \approx 96.76\,kg$.

3.

As we know, set $y = Qx$.

$$\nabla_x J(x) = \frac{d(x^T y)}{dx} + \frac{d(d^T x)}{dx} + \frac{d(c)}{dx}$$

$$= \frac{\partial(x^T y)}{\partial x} + \frac{d(y^T(x))}{dx}\frac{\partial(f(x,y))}{\partial y} + \frac{d(x^T d)}{dx}$$

$\because \frac{\partial(x^T y)}{\partial x} = y = Qx$,

$\frac{d(y^T(x))}{dx}\cdot\frac{\partial(f(x,y))}{\partial y} = \frac{d(y(x)^T)}{dx}\cdot\frac{\partial(x^T y)}{\partial y} = \frac{d((Qx)^T)}{dx}\frac{\partial(y^T\cdot x)}{\partial y} = \frac{d(x^T Q^T)}{dx}\cdot\frac{\partial(y^T x)}{\partial y} = Q^T x$.

$\Rightarrow \nabla_x J(x) = Qx + Q^T x + d = (Q+Q^T)x + d$

$\because Q = Q^T$

$\Rightarrow \nabla_x J(x) = 2Qx + d$.

$$H=\frac{\partial^2 J}{\partial x \partial x^T} = \frac{\partial\left(\frac{\partial J}{\partial x}\right)}{\partial x^T} = \frac{\partial(2Qx+d)}{\partial x^T} = \frac{\partial(2Qx)}{\partial x^T} = \frac{\partial(2x^TQ^T)}{\partial x^T} = 2Q^T.$$

$\because Q = Q^T$

$\Rightarrow H = 2Q.$



4.

As we know.

$$\hat{\beta}_{(p+1)\times 1} = (X^T_{n\times(p+1)} \cdot X_{n\times(p+1)})^{-1} \cdot X_{n\times p+1} \cdot y_{n\times 1} = [\hat{\beta}_1, \hat{\beta}_2 \cdots \hat{\beta}_p, \beta_{bias}]^T$$

we also know that.

$$\hat{y} = x'_{1\times p}\,\hat{\beta}_{p+1} + \hat{\beta}_{bias}.$$

if $x'_{1\times p}$ is augmented as $x'_{1\times(1+p)}$.

$$\Rightarrow \hat{y} = Y_{1\times(p+1)}\,\hat{\beta}_{(p+1)\times 1}$$

$$= Y'(x^Tx^{-1})x^Ty \quad \text{where } Y' = x'_{1\times p},\ x = x_{n\times(p+1)},\ y = y_{n\times 1}$$

$$= wy$$

It is a special case of knn regression when k=n.

5. Suppose $y$ is a member of the column space of $X$.

$\because \hat{y} = X(X^TX)^{-1}X^Ty$

$\because X \in R^{n\times(p+1)} \Rightarrow X$ is square and invertible.

$\Rightarrow \hat{y} = X(X^TX)^{-1}X^Ty = XX^{-1}(X^T)^{-1}X^T \cdot y = X \cdot X^{-1}y = y$.

$\Rightarrow \hat{y}$ is a number of the column space of $X$, is a linear combination of column $X$.

6. From the problem we know that:

if we prove $y-\hat{y}$ is orthogonal to the column space of $X$.

we just need to prove that $(y-\hat{y})X = 0$.

First, $y = X\beta$

$\Rightarrow RSS(\beta) = \|y - X\hat{\beta}\|^2$

$= (y-X\hat{\beta})^T(y-X\hat{\beta})$

$\because \hat{\beta}$ minimize the $RSS(\beta)$

$\Rightarrow \frac{\partial}{\partial\beta}(y-X\hat{\beta})^T(y-X\hat{\beta}) = 0$

$\Rightarrow \frac{\partial}{\partial\beta}(y^Ty - y^TX\beta - \beta^TX^Ty + \beta^TX^TX\beta = 0$

$\Rightarrow \frac{\partial}{\partial\beta}(y^Ty - 2y^TX\hat{\beta} + \hat{\beta}^TX^TX\hat{\beta}) = 0$

$\Rightarrow -2X^Ty + 2X^TX\hat{\beta} = 0$

$\because y - \hat{y} = 0$

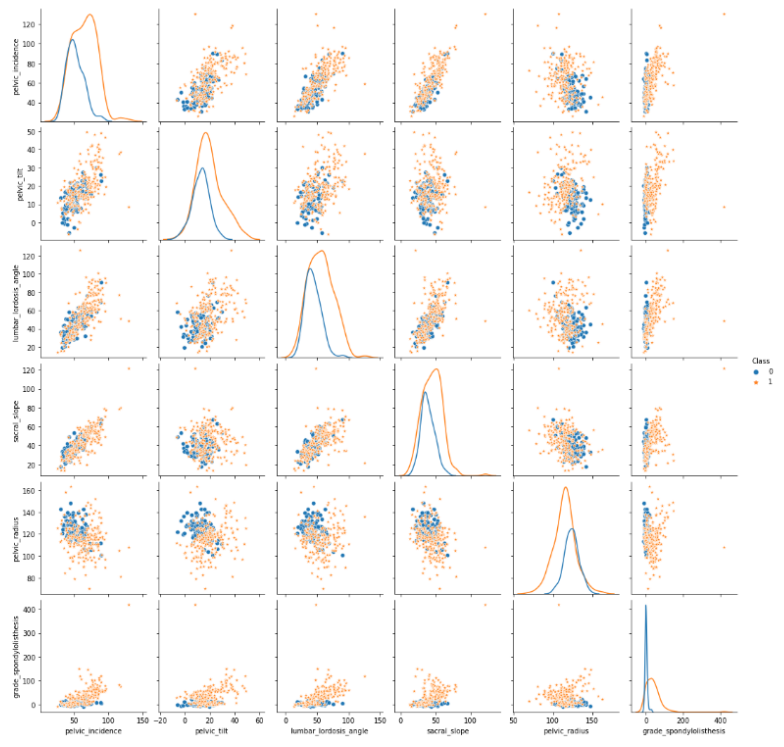
$\Rightarrow (y-\hat{y})X = 0$

$\Rightarrow y-\hat{y}$ is orthogonal to the column of the $X$. if $\hat{\beta}$ minimizes $RSS(\beta)$.

7.

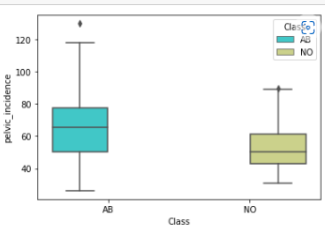
b. c(i)

```
In [109]: g=sns.pairplot(df_binary, hue='Class', markers=["o", "*"],kind='scatter',diag_kws={"shade": False})
```

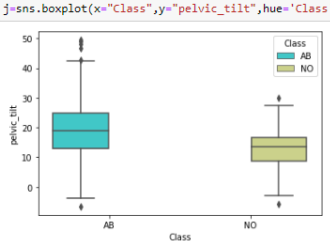



c (ii)

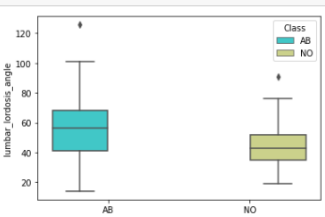
```
In [110]: h=sns.boxplot(x="Class",y="pelvic_incidence",hue='Class',data=df, palette="rainbow")
```



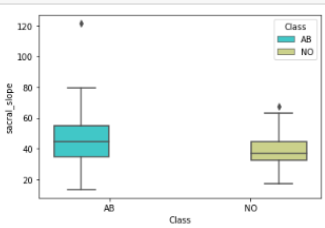

```
In [111]: j=sns.boxplot(x="Class",y="pelvic_tilt",hue='Class',data=df, palette="rainbow")
```



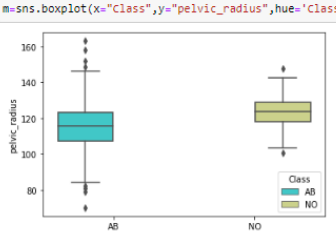

```
In [112]: k=sns.boxplot(x="Class",y="lumbar_lordosis_angle",hue='Class',data=df, palette="rainbow")
```



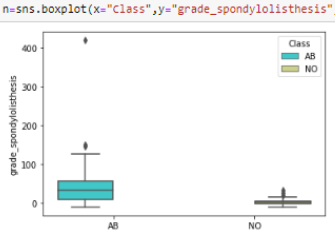

```
In [113]: l=sns.boxplot(x="Class",y="sacral_slope",hue='Class',data=df, palette="rainbow")
```




```
In [114]: m=sns.boxplot(x="Class",y="pelvic_radius",hue='Class',data=df, palette="rainbow")
```




```
In [115]: n=sns.boxplot(x="Class",y="grade_spondylolisthesis",hue='Class',data=df, palette="rainbow")
```



(iii)

:

| | pelvic_incidence | pelvic_tilt | lumbar_lordosis_angle | sacral_slope | pelvic_radius | grade_spondylolisthesis | Class |
|---|---|---|---|---|---|---|---|
| **0** | 38.51 | 16.96 | 35.11 | 21.54 | 127.63 | 7.99 | 0 |
| **1** | 54.92 | 18.97 | 51.60 | 35.95 | 125.85 | 2.00 | 0 |
| **2** | 44.36 | 8.95 | 46.90 | 35.42 | 129.22 | 4.99 | 0 |
| **3** | 48.32 | 17.45 | 48.00 | 30.87 | 128.98 | -0.91 | 0 |
| **4** | 45.70 | 10.66 | 42.58 | 35.04 | 130.18 | -3.39 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **205** | 77.12 | 30.35 | 77.48 | 46.77 | 110.61 | 82.09 | 1 |
| **206** | 88.02 | 39.84 | 81.77 | 48.18 | 116.60 | 56.77 | 1 |
| **207** | 83.40 | 34.31 | 78.42 | 49.09 | 110.47 | 49.67 | 1 |
| **208** | 72.05 | 24.70 | 79.87 | 47.35 | 107.17 | 56.43 | 1 |
| **209** | 85.10 | 21.07 | 91.73 | 64.03 | 109.06 | 38.03 | 1 |

210 rows × 7 columns

C.

(i)

In [117]:
```
classifier=KNeighborsClassifier(n_neighbors=208,metric='euclidean')
Train_error=[]
alternative_k = np.arange(208, 0, -3)
#choose the train and errors in terms of k belongs to {208,205,....,4,1}
for i in alternative_k:
    knn=KNeighborsClassifier(n_neighbors=i,metric='euclidean')
    knn.fit(X_T, y_T)
    pred_i = knn.predict(X_T)
    Train_error.append(np.mean(pred_i != y_T))
print("Minimum train error:",min(Train_error),"at K =",208-3*Train_error.index(min(Train_error)))
Test_error = []
for i in alternative_k:
    knn=KNeighborsClassifier(n_neighbors=i,metric='euclidean')
    knn.fit(X_T, y_T)
    pred_i = knn.predict(X_test)
    Test_error.append(np.mean(pred_i != Y_test))
```

```
Minimum train error: 0.0 at K = 1
```

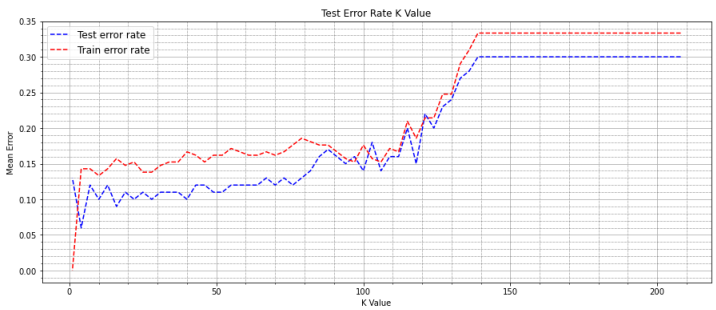
(ii)


In [140]:
```
knn = KNeighborsClassifier(n_neighbors = 4, metric = "euclidean")
knn.fit(X_T, y_T)
pred = knn.predict(X_test)
conf_matrix = confusion_matrix(Y_test, pred)
average_precision = precision_score(Y_test, pred)
fscore = f1_score(Y_test, pred)
print("Confusion matrix is:\n",conf_matrix)
print('**Classification report is:**')
print(classification_report(Y_test, y_pred))
print("True Positive Rate - ", conf_matrix[1][1]/(conf_matrix[1][0] + conf_matrix[1][1]))
print("True Negative Rate - ", conf_matrix[0][0]/(conf_matrix[0][0] + conf_matrix[0][1]))
print("Precision : ", average_precision)
print("F1-Score at k=4 : ", fscore)
```

```
Confusion matrix is:
 [[25  5]
 [ 1 69]]
**Classification report is:**
              precision    recall  f1-score   support

           0       0.87      0.67      0.75        30
           1       0.87      0.96      0.91        70

    accuracy                           0.87       100
   macro avg       0.87      0.81      0.83       100
weighted avg       0.87      0.87      0.86       100

True Positive Rate -  0.9857142857142858
True Negative Rate -  0.8333333333333334
Precision :  0.9324324324324325
F1-Score at k=4 :  0.9583333333333333
```

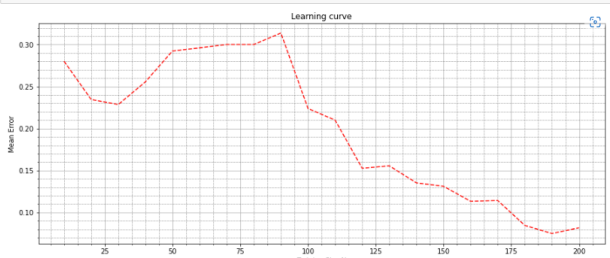
In [118]:
```
plt.figure(figsize=(15, 6))
plt.plot(alternative_k, Test_error, color='blue', linestyle='dashed',markerfacecolor='blue', markersize=10,label='Test error rate
plt.plot(alternative_k, Train_error, color='red', linestyle='dashed',markerfacecolor='blue', markersize=10,label='Train error rat
plt.title('Test Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.grid(True)
plt.minorticks_on()
plt.legend(prop={'size': 12})
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
print("Minimum test error:",min(Test_error),"at K =",208-3*Test_error.index(min(Test_error)))
```

```
Minimum test error: 0.06 at K = 4
```




c(iii)

In [120]:
```
alternative_N = np.arange(10, 210, 10)
Train_K_min=[]
for N in alternative_N:
    training_N_set=pd.merge(df_NO.iloc[0:N//3,:],df_AB.iloc[0:N-N//3,:],how='outer') #selec training set
    Test_N_set= pd.merge(df_NO.iloc[N//3:,:],df_AB.iloc[N-N//3:,:],how='outer')
    alternative_N_k=np.arange(1,N,5)
    X_N_Train = training_N_set.iloc[:, :-1].values
    Y_N_Train = training_N_set.iloc[:, -1].values
    X_N_Test = Test_N_set.iloc[:, :-1].values
    Y_N_Test = Test_N_set.iloc[:, -1].values
    Train_N_error=[]
    for k in alternative_N_k:
            classifier=KNeighborsClassifier \
            (n_neighbors=k,metric='euclidean')
            classifier.fit(X_N_Train, Y_N_Train)
            pred_i=classifier.predict(X_N_Test)
            Train_N_error.append(np.mean(pred_i != Y_N_Test))
    Train_K_min.append((min(Train_N_error)))
x=range(10,211,10)
plt.figure(figsize=(15, 6))
plt.plot(alternative_N, Train_K_min, color='red', linestyle='dashed',markerfacecolor='blue', markersize=10)
plt.title('Learning curve')
plt.xlabel('Training Size N')
plt.ylabel('Mean Error')
plt.grid(True)
plt.minorticks_on()
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
```




d (ii)

A

In [121]:
```
print("Metric : ", "Manhattan")
train_error_throughtout = []
k = np.arange(1,200,5)
test_error_array= {}
train_error_array = {}
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,p=1, metric = "minkowski")
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\ntest error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))
optimal_k= [key for m in [min(test_error_array.values())] for key,val in test_error_array.items() if val == m][-1]
print("\nerror:",min(test_error_array.values()))
print("\noptimal_k:",optimal_k)
```

```
Metric :  Manhattan

test error array
 {1: 0.12, 6: 0.10999999999999999, 11: 0.10999999999999999, 16: 0.12, 21: 0.12, 26: 0.10999999999999999, 31: 0.12, 36: 0.13, 4
1: 0.13, 46: 0.13, 51: 0.14, 56: 0.13, 61: 0.15000000000000002, 66: 0.15000000000000002, 71: 0.14, 76: 0.14, 81: 0.160000000000
00003, 86: 0.15000000000000002, 91: 0.19999999999999996, 96: 0.19999999999999996, 101: 0.21999999999999997, 106: 0.209999999999
99996, 111: 0.22999999999999998, 116: 0.20999999999999996, 121: 0.21999999999999997, 126: 0.24, 131: 0.28, 136: 0.28, 141: 0.30
000000000000004, 146: 0.30000000000000004, 151: 0.30000000000000004, 156: 0.30000000000000004, 161: 0.30000000000000004, 166:
0.30000000000000004, 171: 0.30000000000000004, 176: 0.30000000000000004, 181: 0.30000000000000004, 186: 0.3000000000000004, 19
1: 0.30000000000000004, 196: 0.30000000000000004}

error: 0.10999999999999999

optimal_k: 26
```

B.

In [122]:
```
p_values = []
for i in range(1,11):
    p_values.append(10**(i/10))
print("\np_value\n",p_values)

test_error_array = {}
train_error_array = {}
for i in p_values:
    knn = KNeighborsClassifier(n_neighbors=optimal_k,p=i,metric = "minkowski")
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\n test error array\n",test_error_array)

train_error_throughtout.append(min(train_error_array.values()))
optimal_p = [key for m in [min(test_error_array.values())] for key,val in test_error_array.items() if val == m]
optimal_p
import math
best_p = [math.log10(x) for x in optimal_p]
print("\nbest p_value:",best_p)
```

```
p_value
 [1.2589254117941673, 1.5848931924611136, 1.9952623149688795, 2.51188643150958, 3.1622776601683795, 3.9810717055349722, 5.01187
2336272722, 6.309573444801933, 7.943282347242816, 10.0]

 test error array
 {1.2589254117941673: 0.09999999999999998, 1.5848931924611136: 0.09999999999999998, 1.9952623149688795: 0.10999999999999999, 2.
51188643150958: 0.09999999999999998, 3.1622776601683795: 0.10999999999999999, 3.9810717055349722: 0.09999999999999998, 5.011872
336272722: 0.10999999999999999, 6.309573444801933: 0.10999999999999999, 7.943282347242816: 0.10999999999999999, 10.0: 0.1099999
9999999999}

best p_value: [0.10000000000000002, 0.20000000000000004, 0.4, 0.6
```

C.

In [123]:
```
k = np.arange(1,200,5)
test_error_array= {}
train_error_array= {}
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,metric='chebyshev')
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\n test error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))

optimal_k = [key for m in [min(test_error_array.values())] for key,val in test_error_array.items() if val == m][0]
print("\noptimal_k:",optimal_k)
```

```
 test error array
 {1: 0.13, 6: 0.09999999999999998, 11: 0.12, 16: 0.07999999999999996, 21: 0.10999999999999999, 26: 0.12, 31: 0.12, 36: 0.099999
99999999998, 41: 0.13, 46: 0.12, 51: 0.12, 56: 0.12, 61: 0.14, 66: 0.13, 71: 0.14, 76: 0.10999999999999999, 81: 0.14, 86: 0.12,
91: 0.15000000000000002, 96: 0.14, 101: 0.18999999999999995, 106: 0.18000000000000005, 111: 0.18000000000000005, 116: 0.1800000
0000000005, 121: 0.20999999999999996, 126: 0.20999999999999996, 131: 0.25, 136: 0.28, 141: 0.30000000000000004, 146: 0.30000000
000000004, 151: 0.30000000000000004, 156: 0.30000000000000004, 161: 0.30000000000000004, 166: 0.30000000000000004, 171: 0.30000
000000000004, 176: 0.30000000000000004, 181: 0.30000000000000004, 186: 0.30000000000000004, 191: 0.30000000000000004, 196: 0.30
000000000000004}

optimal_k: 16
```

d(ii)

In [124]:
```
k = np.arange(1,200,5)
test_error_array = {}
train_error_array = {}
covarience_x = np.cov(X_T)
covarience_x_inverse = np.linalg.inv(covarience_x)
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,algorithm='brute',
                               metric='mahalanobis',
                               metric_params={'VI': covarience_x_inverse})
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\n test error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))
optimal_k = [key for m in [min(test_error_array.values())] for key,val in test_error_array.items() if val == m][0]
print("\noptimal_k:",optimal_k)
```

```
 test error array
 {1: 0.22999999999999998, 6: 0.25, 11: 0.20999999999999996, 16: 0.20999999999999996, 21: 0.21999999999999997, 26: 0.22999999999
999998, 31: 0.24, 36: 0.24, 41: 0.25, 46: 0.26, 51: 0.27, 56: 0.30000000000000004, 61: 0.29000000000000004, 66: 0.27, 71: 0.290
00000000000004, 76: 0.31999999999999995, 81: 0.32999999999999996, 86: 0.32999999999999996, 91: 0.32999999999999996, 96: 0.33999
999999999997, 101: 0.33999999999999997, 106: 0.33999999999999997, 111: 0.32999999999999996, 116: 0.32999999999999996, 121: 0.32
999999999999996, 126: 0.31999999999999995, 131: 0.31999999999999995, 136: 0.30000000000000004, 141: 0.30000000000000004, 146:
0.30000000000000004, 151: 0.30000000000000004, 156: 0.30000000000000004, 161: 0.30000000000000004, 166: 0.30000000000000004, 17
1: 0.30000000000000004, 176: 0.30000000000000004, 181: 0.30000000000000004, 186: 0.30000000000000004, 191: 0.30000000000000004,
196: 0.30000000000000004}

optimal_k: 11
```

e.

In [125]:
```
k = np.arange(1,200,5)
test_error_array = {}
train_error_array = {}
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,weights='distance')
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\ntest error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))
best_error = min(test_error_array.values())
print("\n Euclidean best error",best_error)
```

```
test error array
 {1: 0.13, 6: 0.09999999999999998, 11: 0.12, 16: 0.10999999999999999, 21: 0.10999999999999999, 26: 0.10999999999999999, 31: 0.1
0999999999999999, 36: 0.10999999999999999, 41: 0.10999999999999999, 46: 0.10999999999999999, 51: 0.10999999999999999, 56: 0.109
99999999999999, 61: 0.10999999999999999, 66: 0.10999999999999999, 71: 0.10999999999999999, 76: 0.12, 81: 0.13, 86: 0.13, 91: 0.
14, 96: 0.14, 101: 0.12, 106: 0.14, 111: 0.14, 116: 0.13, 121: 0.13, 126: 0.13, 131: 0.14, 136: 0.14, 141: 0.17000000000000004,
146: 0.18999999999999995, 151: 0.18999999999999995, 156: 0.19999999999999996, 161: 0.22999999999999998, 166: 0.25, 171: 0.27, 1
76: 0.27, 181: 0.27, 186: 0.27, 191: 0.28, 196: 0.28}

 Euclidean best error 0.09999999999999998
```

In [126]:
```
k = np.arange(1,200,5)
test_error_array = {}
train_error_array = {}
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,weights='distance',p=1)
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\ntest error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))
best_error = min(test_error_array.values())
print("\n Manhattan best error",best_error)
```

```
test error array
 {1: 0.12, 6: 0.10999999999999999, 11: 0.10999999999999999, 16: 0.12, 21: 0.10999999999999999, 26: 0.09999999999999998, 31: 0.1
0999999999999999, 36: 0.10999999999999999, 41: 0.10999999999999999, 46: 0.10999999999999999, 51: 0.12, 56: 0.12, 61: 0.13, 66:
0.13, 71: 0.10999999999999999, 76: 0.12, 81: 0.14, 86: 0.14, 91: 0.13, 96: 0.13, 101: 0.15000000000000002, 106: 0.1500000000000
0002, 111: 0.15000000000000002, 116: 0.15000000000000002, 121: 0.15000000000000002, 126: 0.15000000000000002, 131: 0.1500000000
0000002, 136: 0.16000000000000003, 141: 0.17000000000000004, 146: 0.18000000000000005, 151: 0.18999999999999995, 156: 0.2099999
9999999996, 161: 0.24, 166: 0.25, 171: 0.27, 176: 0.27, 181: 0.27, 186: 0.27, 191: 0.28, 196: 0.29000000000000004}

 Manhattan best error 0.09999999999999998
```

In [127]:
```
k = np.arange(1,200,5)
test_error_array = {}
train_error_array = {}
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,metric='chebyshev',weights='distance')
    knn.fit(X_T, y_T)
    y_test_pred = knn.predict(X_test)
    test_error_array.update({i: 1 - accuracy_score(Y_test, y_test_pred)})
    y_train_pred = knn.predict(X_T)
    train_error_array.update({i: 1 - accuracy_score(y_T, y_train_pred)})
print("\ntest error array\n",test_error_array)
train_error_throughtout.append(min(train_error_array.values()))
best_error = min(test_error_array.values())
print("\n  Chebyshev best error",best_error)
```

```
test error array
 {1: 0.13, 6: 0.13, 11: 0.12, 16: 0.10999999999999999, 21: 0.12, 26: 0.12, 31: 0.10999999999999999, 36: 0.10999999999999999, 4
1: 0.10999999999999999, 46: 0.12, 51: 0.12, 56: 0.12, 61: 0.10999999999999999, 66: 0.13, 71: 0.12, 76: 0.12, 81: 0.12, 86: 0.1
4, 91: 0.13, 96: 0.14, 101: 0.14, 106: 0.14, 111: 0.14, 116: 0.14, 121: 0.15000000000000002, 126: 0.14, 131: 0.1600000000000000
3, 136: 0.16000000000000003, 141: 0.18000000000000005, 146: 0.18999999999999995, 151: 0.20999999999999996, 156: 0.2099999999999
9996, 161: 0.22999999999999998, 166: 0.26, 171: 0.26, 176: 0.26, 181: 0.27, 186: 0.27, 191: 0.27, 196: 0.28}

  Chebyshev best error 0.10999999999999999
```

f.

In [134]:
```
lt_error= min(train_error_throughtout)
print("\n\nlowest_train_error:",lt_error)
```

```
lowest_train_error: 0.0
```