In [98]:
```
import pandas as pd
import numpy as np
from scipy.io.arff import loadarff
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from scipy import stats
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
from sklearn.metrics import confusion_matrix, classification_report, Confusion
MatrixDisplay
import math
```

(a)Import data

In [99]:
```
df = pd.DataFrame(loadarff('../vertebral_column_data/column_2C_weka.arff')[0])
```

In [100]:
```
df
```

Out[100]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spon
0	63.027817	22.552586	39.609117	40.475232	98.672917	
1	39.056951	10.060991	25.015378	28.995960	114.405425	
2	68.832021	22.218482	50.092194	46.613539	105.985135	
3	69.297008	24.652878	44.311238	44.644130	101.868495	
4	49.712859	9.652075	28.317406	40.060784	108.168725	
...	...	...	...	...	...	
305	47.903565	13.616688	36.000000	34.286877	117.449062	
306	53.936748	20.721496	29.220534	33.215251	114.365845	
307	61.446597	22.694968	46.170347	38.751628	125.670725	
308	45.252792	8.693157	41.583126	36.559635	118.545842	
309	33.841641	5.073991	36.641233	28.767649	123.945244	

310 rows × 7 columns

(b) Pre-Processing and Exploratory data analysis:

i. Make scatterplots of the independent variables in the dataset. Use color to show Classes 0 and 1.

In [101]:
```
encoder = LabelEncoder()
df['class'] = 1-encoder.fit_transform(df['class'])
```

In [102]:
```
df
```

Out[102]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spon
0	63.027817	22.552586	39.609117	40.475232	98.672917	
1	39.056951	10.060991	25.015378	28.995960	114.405425	
2	68.832021	22.218482	50.092194	46.613539	105.985135	
3	69.297008	24.652878	44.311238	44.644130	101.868495	
4	49.712859	9.652075	28.317406	40.060784	108.168725	
...	...	...	...	...	...	
305	47.903565	13.616688	36.000000	34.286877	117.449062	
306	53.936748	20.721496	29.220534	33.215251	114.365845	
307	61.446597	22.694968	46.170347	38.751628	125.670725	
308	45.252792	8.693157	41.583126	36.559635	118.545842	
309	33.841641	5.073991	36.641233	28.767649	123.945244	

310 rows × 7 columns

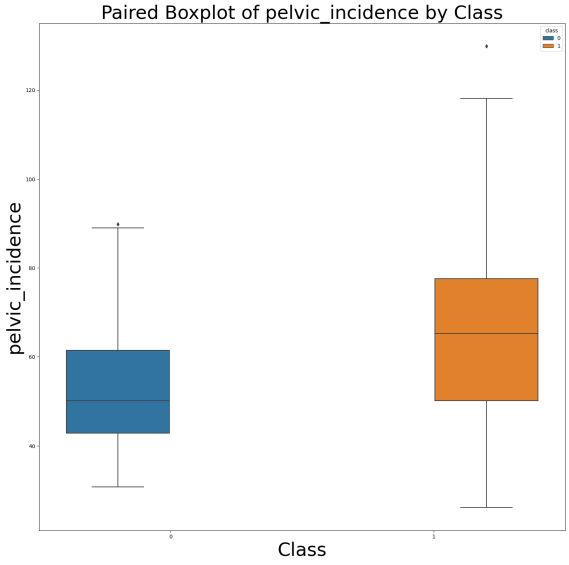
In [103]:
```
#Paired variables scatter plots
sns.pairplot(df, hue="class")
```

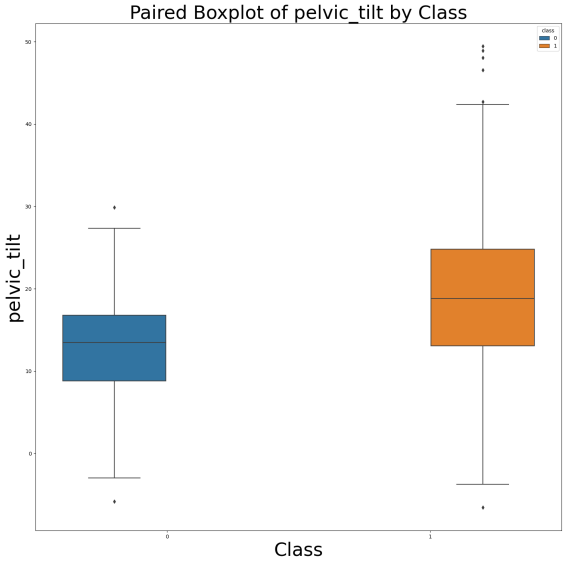
Out[103]: <seaborn.axisgrid.PairGrid at 0x2388de8e730>

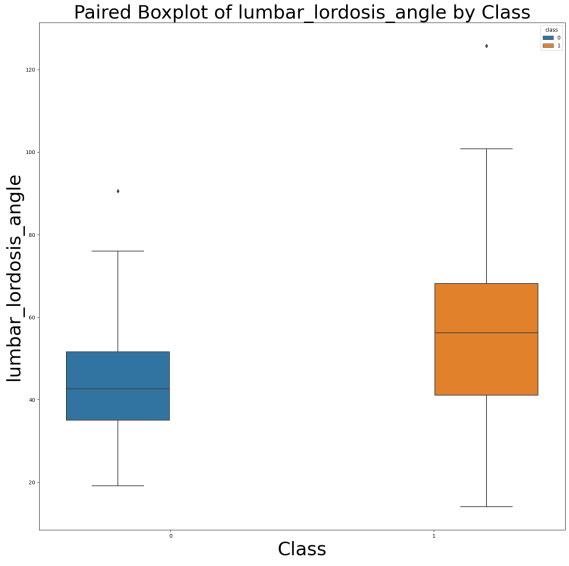
ii. Make boxplots for each of the independent variables. Use color to show Classes 0 and 1 (see ISLR p. 129).

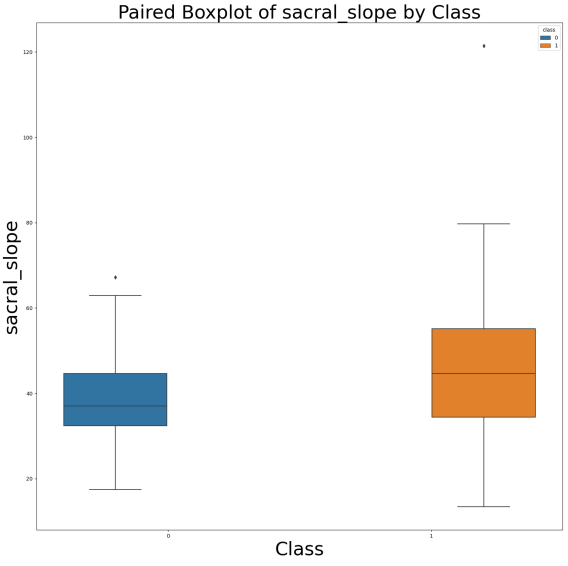
In [104]:

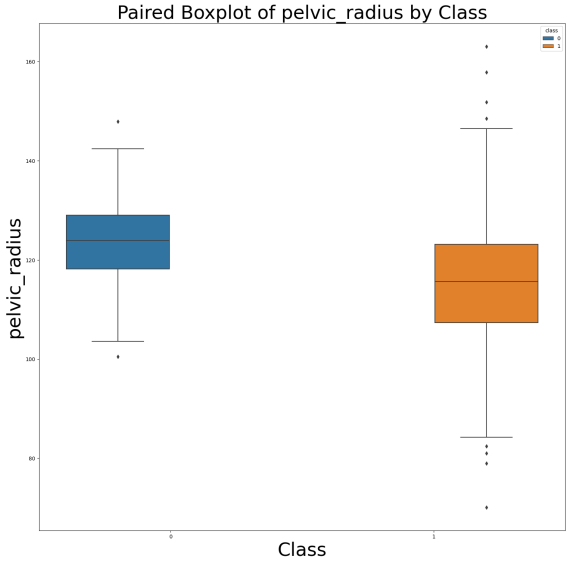
```
fig, axes = plt.subplots(3, 2, figsize=(40,60))
box_width = 0.8

for i, column in enumerate(df.columns[:-1]):
    row = i // 2
    col = i % 2

    sns.boxplot(data=df, x='class', y=column, hue='class', ax=axes[row, col],
width=box_width)
    axes[row, col].set_xlabel('Class', fontsize=36)
    axes[row, col].set_ylabel(column, fontsize=36)
    axes[row, col].set_title(f'Paired Boxplot of {column} by Class', fontsize=
36)

# Show the plots
plt.show()
```

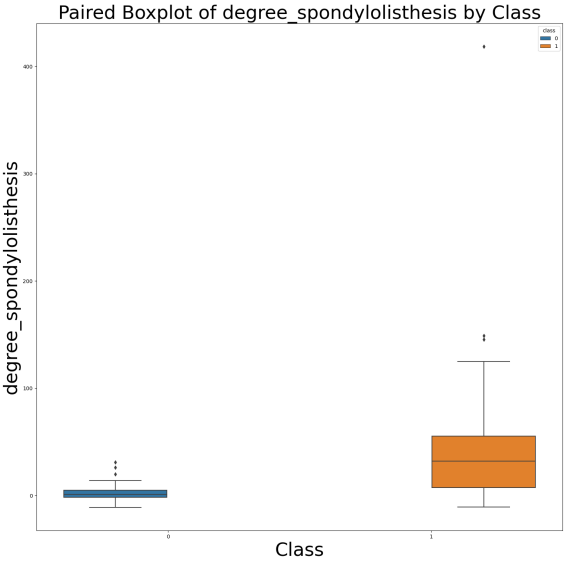

Paired Boxplot of pelvic_incidence by Class
pelvic_incidence
Class
class
0
1


Paired Boxplot of pelvic_tilt by Class
pelvic_tilt
Class
class
0
1


Paired Boxplot of lumbar_lordosis_angle by Class
lumbar_lordosis_angle
Class
class
0
1


Paired Boxplot of sacral_slope by Class
sacral_slope
Class
class
0
1


Paired Boxplot of pelvic_radius by Class
pelvic_radius
Class
class
0
1


Paired Boxplot of degree_spondylolisthesis by Class
degree_spondylolisthesis
Class
class
0
1

iii. Select the first 70 rows of Class 0 and the first 140 rows of Class 1 as the training set and the rest of the data as the test set.

In [105]:
```
train_class_0 = df[df['class'] == 0].iloc[:70]
train_class_1 = df[df['class'] == 1].iloc[:140]
train_set = pd.concat([train_class_0, train_class_1])
test_set = df.drop(train_set.index)

train_set.reset_index(drop=True, inplace=True)
test_set.reset_index(drop=True, inplace=True)
```

In [106]:
```
X_train=train_set.iloc[:,:6]
y_train=train_set.iloc[:,-1]
```

In [107]:
```
X_test=test_set.iloc[:,:6]
y_test=test_set.iloc[:,-1]
```

(c) Classification using KNN on Vertebral Column Data Set
i. Write code for k-nearest neighbors with Euclidean metric (or use a software package).

In [108]:
```
def knn_model(k):

    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)

    y_pred_train = knn_classifier.predict(X_train)
    y_pred_test = knn_classifier.predict(X_test)

    train_error = 1 - accuracy_score(y_train, y_pred_train)
    test_error = 1 - accuracy_score(y_test, y_pred_test)

    return train_error, test_error
```

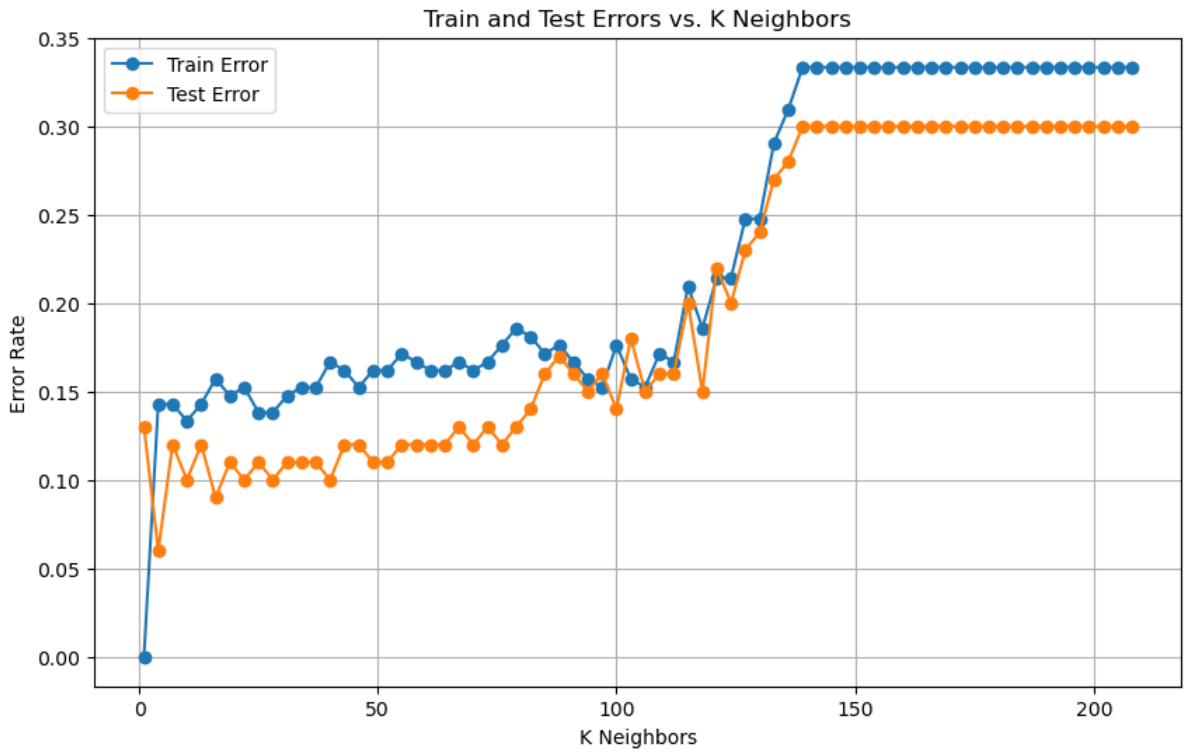
ii. Test all the data in the test database with k nearest neighbors. Take decisions by majority polling. Plot train and test errors in terms of k for k ∈ {208, 205, . . . , 7, 4, 1, } (in reverse order). You are welcome to use smaller increments of k. Which k *is the most suitable k among those values? Calculate the confusion matrix, true positive rate, true negative rate, precision,and F1-score when k = k*

In [109]:
```
k_values = list(range(1, 209, 3))

train_errors = []
test_errors = []
```

In [110]:
```
for k in k_values:
    train_error, test_error = knn_model(k)
    train_errors.append(train_error)
    test_errors.append(test_error)

best_k = k_values[test_errors.index(min(test_errors))]
min_test_error=min(test_errors)
```

In [111]:
```
# Plot train and test errors
plt.figure(figsize=(10, 6))
plt.plot(k_values, train_errors, label='Train Error', marker='o')
plt.plot(k_values, test_errors, label='Test Error', marker='o')
plt.xlabel('K Neighbors')
plt.ylabel('Error Rate')
plt.title('Train and Test Errors vs. K Neighbors')
plt.legend()
plt.grid(True)

# Print the best k
print(f"The most suitable k among the tested values is k* = {best_k}, test err
or is {min_test_error}")
```

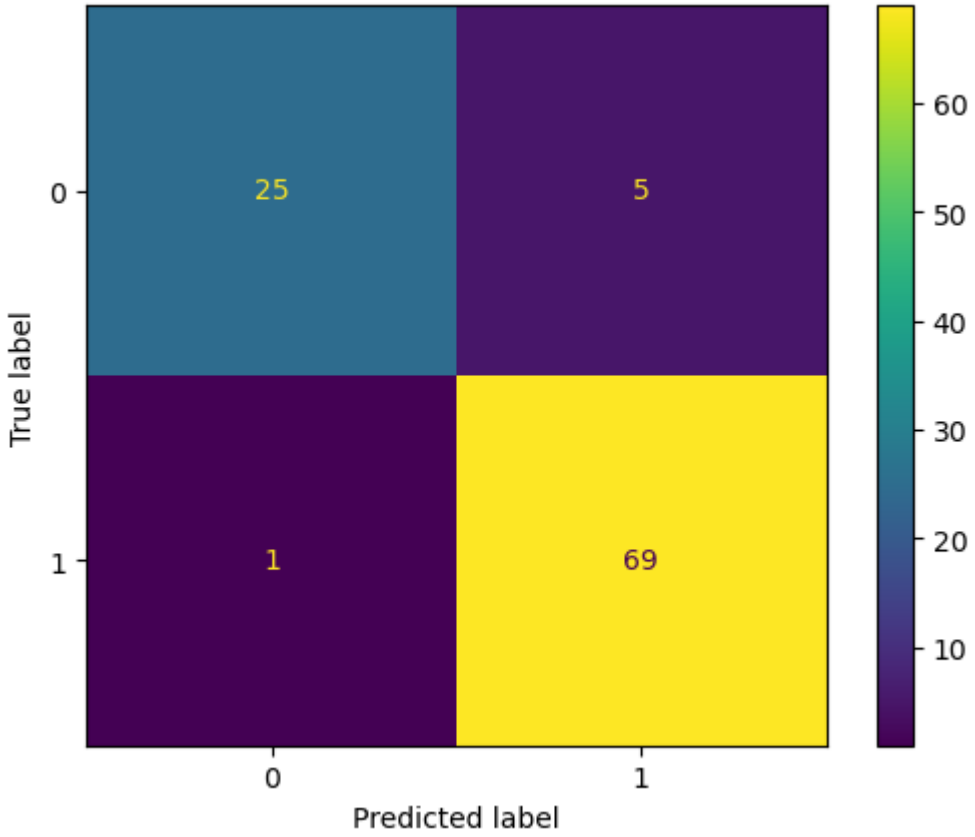
```
The most suitable k among the tested values is k* = 4, test error is 0.060000
00000000005
```



In [112]:

```
# Train the KNN model with the best k
best_knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
best_knn_classifier.fit(X_train, y_train)

# Make predictions with the best model
y_pred_best = best_knn_classifier.predict(X_test)

# Calculate confusion matrix and classification report
conf_matrix = confusion_matrix(y_test, y_pred_best)
tn, fp, fn, tp = conf_matrix.ravel()
true_positive_rate = tp / (tp + fn)
true_negative_rate = tn / (tn + fp)
precision = tp / (tp + fp)
f1_score = 2 * (precision * true_positive_rate) / (precision + true_positive_r
ate)

print("Confusion Matrix:")
ConfusionMatrixDisplay(conf_matrix).plot()
plt.show()

print(f"True Positive Rate: {true_positive_rate:.2f}")
print(f"True Negative Rate: {true_negative_rate:.2f}")
print(f"Precision: {precision:.2f}")
print(f"F1-Score: {f1_score:.2f}")
```

```
Confusion Matrix:
```




```
True Positive Rate: 0.99
True Negative Rate: 0.83
Precision: 0.93
F1-Score: 0.96
```

iii. Since the computation time depends on the size of the training set, one may only use a subset of the training set. Plot the best test error rate, which is obtained by some value of k, against the size of training set, when the size of training set is N ∈ {10, 20, 30, . . . , 210}.

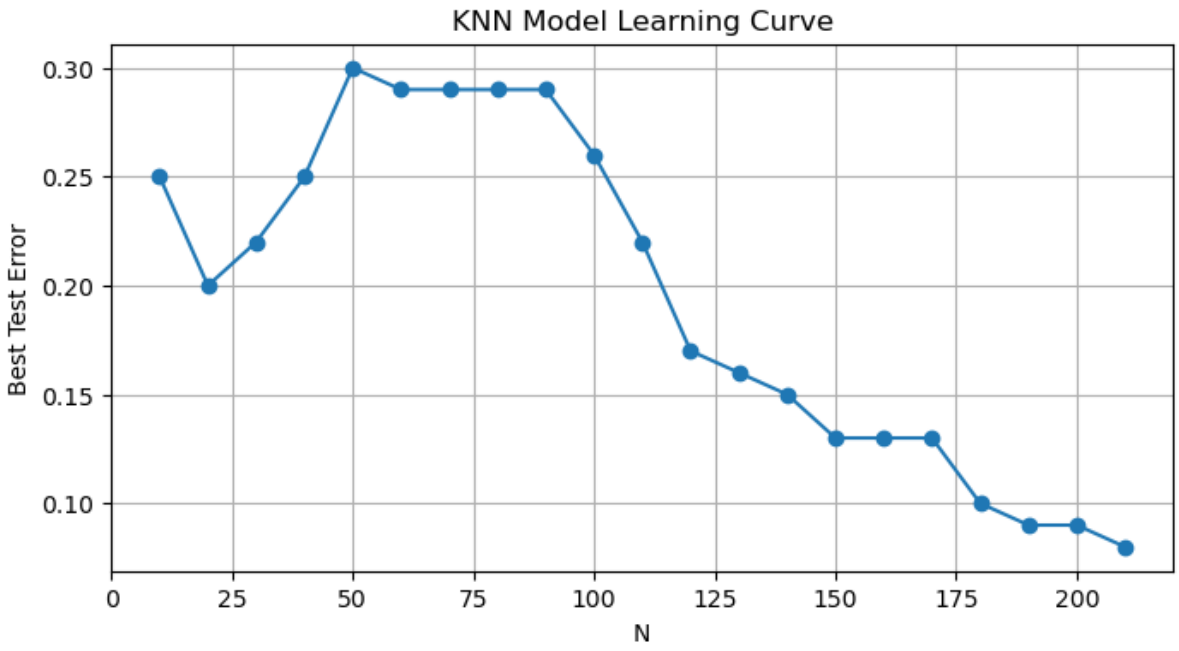
Note: for each N, select your training set by choosing the first N/3 rows of Class 0 and the first N − N/3 rows of Class 1 in the training set you created in 1(b)iii. Also, for each N, select the optimal k from a set starting from k = 1, increasing by 5. For example, if N = 200, the optimal k is selected from {1, 6, 11, . . . , 196}. This plot is called a Learning Curve.

In [113]:
```
n_set=list(range(10, 211, 10))
```

In [114]:
```
def knn_model_sub(k, X_train_subset, y_train_subset):

    knn_classifier_sub = KNeighborsClassifier(n_neighbors=k)
    knn_classifier_sub.fit(X_train_subset, y_train_subset)

    y_pred_test_sub = knn_classifier_sub.predict(X_test)
    test_sub_error = 1 - accuracy_score(y_test, y_pred_test_sub)

    return test_sub_error
```

In [115]:
```
best_test_error_list=[]

for n in n_set:

    train_sub_0 = train_set[train_set['class'] == 0].iloc[:math.floor(n/3)]
    train_sub_1 = train_set[train_set['class'] == 1].iloc[:(n-math.floor(n/
3))]

    train_sub_set = pd.concat([train_sub_0, train_sub_1])
    train_sub_set.reset_index(drop=True, inplace=True)
    X_train_subset=train_sub_set.iloc[:,:6]
    y_train_subset=train_sub_set.iloc[:,-1]

    errors_n=[]
    k_set=list(range(1,n+1,5))

    for k in k_set:
        test_sub_error=knn_model_sub(k, X_train_subset, y_train_subset)
        errors_n.append(test_sub_error)

    best_test_error = min(errors_n)
    best_test_error_list.append(best_test_error)
```

In [116]:
```
plt.figure(figsize=(8, 4))
plt.plot(n_set, best_test_error_list, marker='o')
plt.xlabel('N')
plt.ylabel('Best Test Error')
plt.title('KNN Model Learning Curve')
plt.grid(True)
```



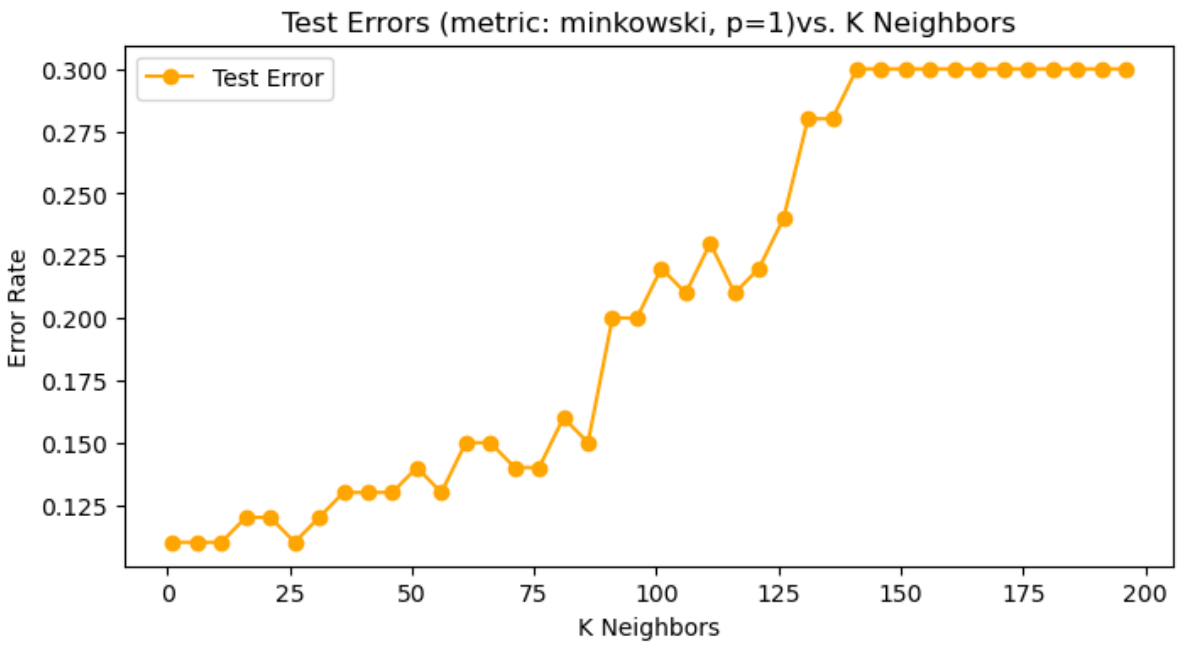
(d) Replace the Euclidean metric with the following metrics and test them. Summarize the test errors (i.e., when k = k∗) in a table. Use all of your training data and select the best k when {1, 6, 11, . . . , 196}.
i. Minkowski Distance:
A. which becomes Manhattan Distance with p = 1.
B. with log10(p) ∈ {0.1, 0.2, 0.3, . . . , 1}. In this case, use the k∗ you found for the Manhattan distance in 1(d)iA. What is the best log10(p)?
C. which becomes Chebyshev Distance with p → ∞

A. which becomes Manhattan Distance with p = 1.

In [117]:
```
def knn_model_metrics(k, this_metric):

    knn_classifier_m = KNeighborsClassifier(n_neighbors=k, metric=this_metric)
    knn_classifier_m.fit(X_train, y_train)

    y_pred_test = knn_classifier_m.predict(X_test)
    test_error = 1 - accuracy_score(y_test, y_pred_test)

    return test_error
```

In [118]:
```
k_set_new=list(range(1,197,5))
test_error_list_minkowski=[]

for k in k_set_new:
    test_error_minkowski = knn_model_metrics(k, 'manhattan')
    test_error_list_minkowski.append(test_error_minkowski)

best_k_minkowski = k_set_new[test_error_list_minkowski.index(min(test_error_li
st_minkowski))]
min_test_error_minkowski=min(test_error_list_minkowski)

plt.figure(figsize=(8,4))
plt.plot(k_set_new, test_error_list_minkowski, label='Test Error', color='oran
ge', marker='o')
plt.xlabel('K Neighbors')
plt.ylabel('Error Rate')
plt.title('Test Errors (metric: minkowski, p=1)vs. K Neighbors')
plt.legend()

# Print the best k
print(f"The most suitable k among the tested values is k* = {best_k_minkowsk
i}, test error is {min_test_error_minkowski}")
```

```
The most suitable k among the tested values is k* = 1, test error is 0.109999
99999999999
```



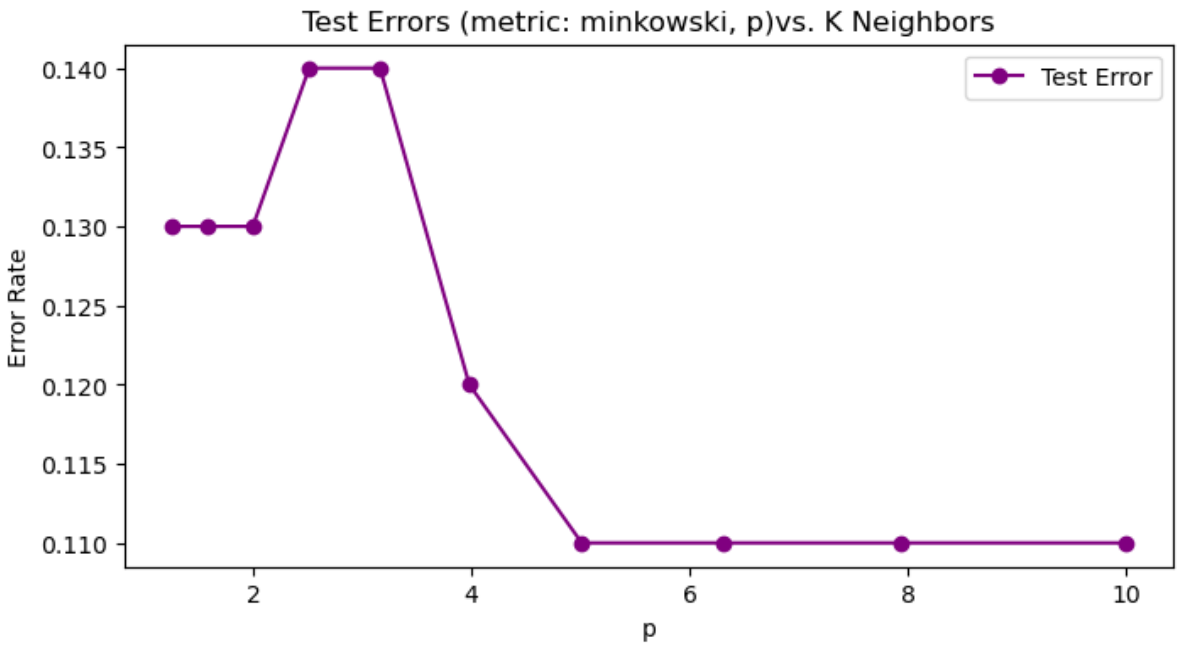
B. with log10(p) ∈ {0.1, 0.2, 0.3, . . . , 1}. In this case, use the k∗ you found for the Manhattan distance in 1(d)iA. What is the best log10(p)?

In [119]:
```
#k=1
def knn_model_metrics_p(p):

    knn_classifier_m = KNeighborsClassifier(n_neighbors=1,
                                            algorithm='brute',
                                            metric="minkowski",
                                            p=p)
    knn_classifier_m.fit(X_train, y_train)

    y_pred_test = knn_classifier_m.predict(X_test)
    test_error = 1 - accuracy_score(y_test, y_pred_test)

    return test_error
```

In [120]:
```
p_list=[10**0.1, 10**0.2, 10**0.3, 10**0.4, 10**0.5, 10**0.6, 10**0.7, 10**0.
8, 10**0.9, 10**1]
test_error_list_p=[]
```

In [121]:
```
for p in p_list:
    test_error_p=knn_model_metrics_p(p)
    test_error_list_p.append(test_error_p)

best_k_p = k_set_new[test_error_list_p.index(min(test_error_list_p))]
min_test_error_p=min(test_error_list_p)

plt.figure(figsize=(8,4))
plt.plot(p_list, test_error_list_p, label='Test Error', color='purple', marker
='o')
plt.xlabel('p')
plt.ylabel('Error Rate')
plt.title('Test Errors (metric: minkowski, p)vs. K Neighbors')
plt.legend()

# Print the best k
print(f"The most suitable p = {(test_error_list_p.index(min(test_error_list_
p))+1)/10}, test error is {min_test_error_p}")
```

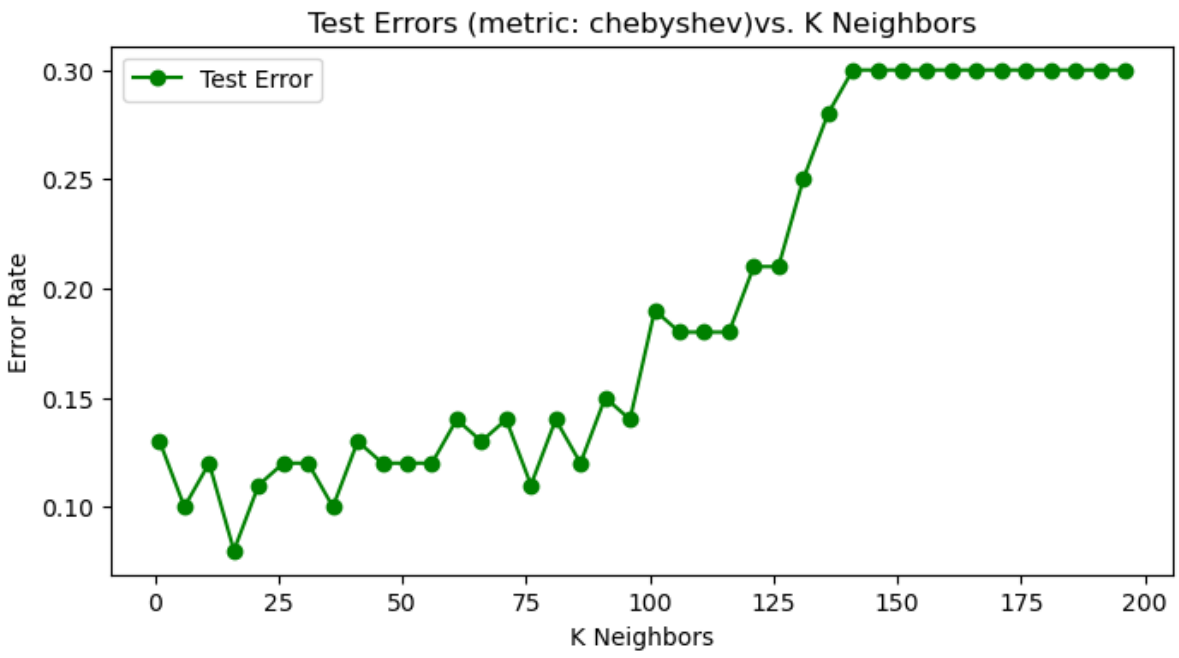
```
The most suitable p = 0.7, test error is 0.10999999999999999
```



C. which becomes Chebyshev Distance with p → ∞

In [122]:
```
k_set_new=list(range(1,197,5))
test_error_list_chebyshev=[]

for k in k_set_new:
    test_error_chebyshev = knn_model_metrics(k, 'chebyshev')
    test_error_list_chebyshev.append(test_error_chebyshev)

best_k_chebyshev = k_set_new[test_error_list_chebyshev.index(min(test_error_li
st_chebyshev))]
min_test_error_chebyshev=min(test_error_list_chebyshev)

plt.figure(figsize=(8,4))
plt.plot(k_set_new, test_error_list_chebyshev, label='Test Error', color='gree
n', marker='o')
plt.xlabel('K Neighbors')
plt.ylabel('Error Rate')
plt.title('Test Errors (metric: chebyshev)vs. K Neighbors')
plt.legend()

# Print the best k
print(f"The most suitable k among the tested values is k* = {best_k_chebyshe
v}, test error is {min_test_error_chebyshev}")
```

```
The most suitable k among the tested values is k* = 16, test error is 0.07999
999999999996
```



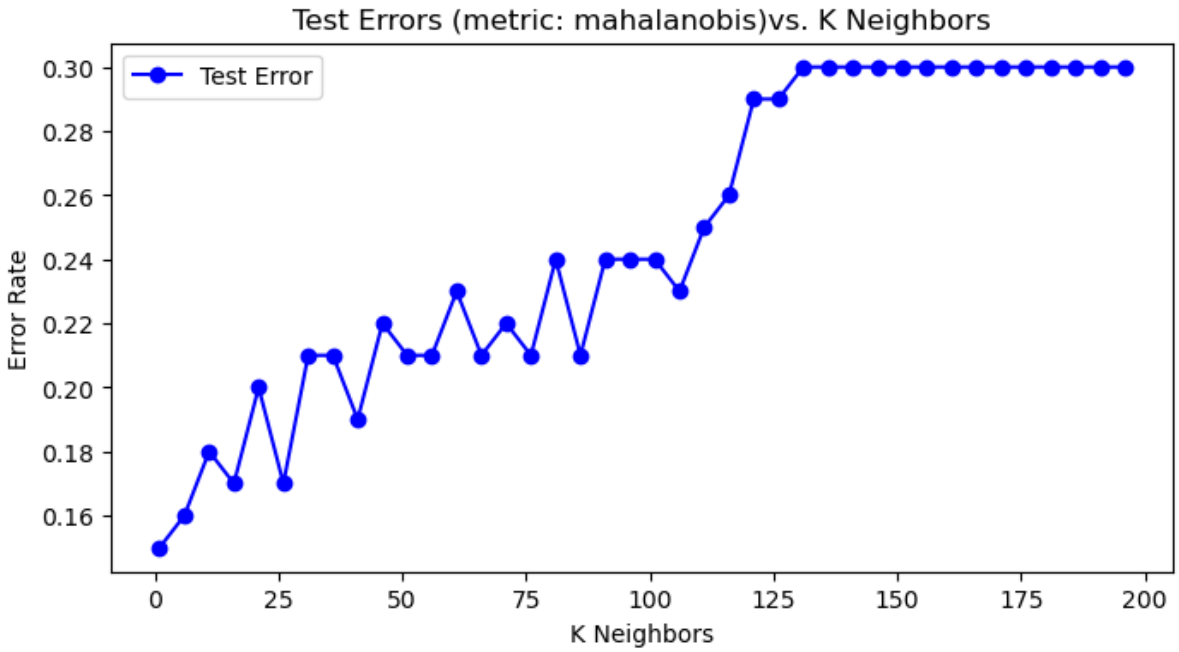
ii. Mahalanobis Distance.

In [123]:

```
def knn_model_metrics2(k):

    cov_matrix = np.cov(X_train, rowvar=False)
    cov_matrix_pseudoinv = np.linalg.pinv(cov_matrix)

    knn_classifier_m = KNeighborsClassifier(n_neighbors=k,
                                            algorithm='brute',
                                            metric='mahalanobis',
                                            metric_params={'VI': cov_matrix_ps
eudoinv})
    knn_classifier_m.fit(X_train, y_train)

    y_pred_test = knn_classifier_m.predict(X_test)
    test_error = 1 - accuracy_score(y_test, y_pred_test)

    return test_error
```

In [124]:
```
test_error_list_mahalanobis=[]

for k in k_set_new:
    test_error_mahalanobis = knn_model_metrics2(k)
    test_error_list_mahalanobis.append(test_error_mahalanobis)

best_k_mahalanobis = k_set_new[test_error_list_mahalanobis.index(min(test_erro
r_list_mahalanobis))]
min_test_error_mahalanobis=min(test_error_list_mahalanobis)

plt.figure(figsize=(8,4))
plt.plot(k_set_new, test_error_list_mahalanobis, label='Test Error', color='bl
ue', marker='o')
plt.xlabel('K Neighbors')
plt.ylabel('Error Rate')
plt.title('Test Errors (metric: mahalanobis)vs. K Neighbors')
plt.legend()

print(f"The most suitable k among the tested values is k* = {best_k_mahalanobi
s}, test error is {min_test_error_mahalanobis}")
```

```
The most suitable k among the tested values is k* = 1, test error is 0.150000
00000000002
```



Summary

In [125]:
```
metrics = ["Manhattan (minkowski, p=1)", "Minkowski, lg(p)=0.7", "Chebyshev (m
inkowski, p → ∞)", "Mahalanobis"]
best_k_list=[1,1,16,1]
best_test_error_list=[0.1099999999999999,0.1099999999999999,0.07999999999999
996,0.15000000000000002]
summary=pd.DataFrame({"Metrics": metrics, "k*":best_k_list, "Test Error":best_
test_error_list})
summary
```

Out[125]:

	Metrics	k*	Test Error
0	Manhattan (minkowski, p=1)	1	0.11
1	Minkowski, lg(p)=0.7	1	0.11
2	Chebyshev (minkowski, p → ∞)	16	0.08
3	Mahalanobis	1	0.15

(e) The majority polling decision can be replaced by weighted decision, in which the weight of each point in voting is inversely proportional to its distance from the query/test data point. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. Use weighted voting with Euclidean, Manhattan, and Chebyshev distances and report the best test errors when k ∈ {1, 6, 11, 16, . . . , 196}.

In [126]:
```
def knn_model_weights(k, this_metric):

    knn_classifier_w = KNeighborsClassifier(n_neighbors=k, metric=this_metric,
weights='distance')
    knn_classifier_w.fit(X_train, y_train)

    y_pred_test = knn_classifier_w.predict(X_test)
    y_pred_train = knn_classifier_w.predict(X_train)

    test_error = 1 - accuracy_score(y_test, y_pred_test)
    train_error = 1 - accuracy_score(y_train, y_pred_train)

    return test_error, train_error
```

In [ ]:
```
k_list_e=list(range(1,197,5))

e_list_train=[]
m_list_train=[]
c_list_train=[]

e_list_test=[]
m_list_test=[]
c_list_test=[]

for k in k_list_e:
    test_error_euclidean, train_error_euclidean = knn_model_weights (k, "eucli
dean")
    test_error_manhattan, train_error_manhattan = knn_model_weights (k, "manha
ttan")
    test_error_chebyshev, train_error_chebyshev = knn_model_weights (k, "cheby
shev")

    e_list_train.append(train_error_euclidean)
    m_list_train.append(train_error_manhattan)
    c_list_train.append(train_error_chebyshev)

    e_list_test.append(test_error_euclidean)
    m_list_test.append(test_error_manhattan)
    c_list_test.append(test_error_chebyshev)
```

In [ ]:
```
best_k_e_test = k_list_e[e_list_test.index(min(e_list_test))]

print(f"The most suitable k for euclidean metric is k* = {best_k_e_test}, test
error is {min(e_list_test)}")

best_k_m_test = k_list_e[m_list_test.index(min(m_list_test))]

print(f"The most suitable k for manhattan metric is k* = {best_k_m_test}, test
error is {min(m_list_test)}")

best_k_c_test = k_list_e[c_list_test.index(min(c_list_test))]

print(f"The most suitable k for chebyshev metric is k* = {best_k_c_test}, test
error is {min(c_list_test)}")
```

(f) What is the lowest training error rate you achieved in this homework?

The lowest training error rate is 0.
When k=1, which means every point itself is a classification. The test error should be 0 at this time because the prediction point is the training point at this situation.

In [ ]: