Group 16 : Chandana Budaati(1002087323), Sanjana Potluri(1002147971), Deepthi Burada(1002034183), Varshith Konduru(1002132051)

In [43]:
```
# Import the required library
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import precision_score, recall_score
import warnings
warnings.filterwarnings("ignore")
```

Loading Dataset

In [44]:
```
# Import the h5py
import h5py

# Load the dataset
with h5py.File('usps.h5', 'r') as hf:
        train = hf.get('train')
        X_train = train.get('data')[:]
        y_train = train.get('target')[:]
        test = hf.get('test')
        X_test = test.get('data')[:]
        y_test = test.get('target')[:]
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[44]:
```
((7291, 256), (7291,), (2007, 256), (2007,))
```

Changing Different K for KNN Classifier

In [45]:
```
sc = [] # List to store accuracy
l = []      # List to store k values
#Checking with different k values
for k in range(1,25):
    l.append(k)
    #Create KNN Classifier
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train,y_train) # Train the KNN Classifier on the training set
    y_pred = neigh.predict(X_test) # Predict the X_test score and store in y_pred
    accuracy = accuracy_score(y_test, y_pred=y_pred)
    sc.append(accuracy)# Accuracy Calculation
```

In [46]:
```
#Display of accuracys
for k, accuracys in zip(l, sc):
    print(f"For k={k}, the accuracy is {accuracys}")
```

```
For k=1, the accuracy is 0.9436970602889886
For k=2, the accuracy is 0.9412057797708022
For k=3, the accuracy is 0.9446935724962631
For k=4, the accuracy is 0.9431988041853513
For k=5, the accuracy is 0.9446935724962631
For k=6, the accuracy is 0.9387144992526159
For k=7, the accuracy is 0.9417040358744395
For k=8, the accuracy is 0.9407075236671649
For k=9, the accuracy is 0.9372197309417041
For k=10, the accuracy is 0.9357249626307922
For k=11, the accuracy is 0.9312406576980568
For k=12, the accuracy is 0.9307424015944196
For k=13, the accuracy is 0.929745889387145
For k=14, the accuracy is 0.9292476332835077
For k=15, the accuracy is 0.9302441454907823
For k=16, the accuracy is 0.9272546088689586
For k=17, the accuracy is 0.9257598405580468
For k=18, the accuracy is 0.9217737917289487
For k=19, the accuracy is 0.9227703039362232
For k=20, the accuracy is 0.9177877428998505
For k=21, the accuracy is 0.9192825112107623
For k=22, the accuracy is 0.9192825112107623
For k=23, the accuracy is 0.9192825112107623
For k=24, the accuracy is 0.9172894867962132
```

Cross Validation

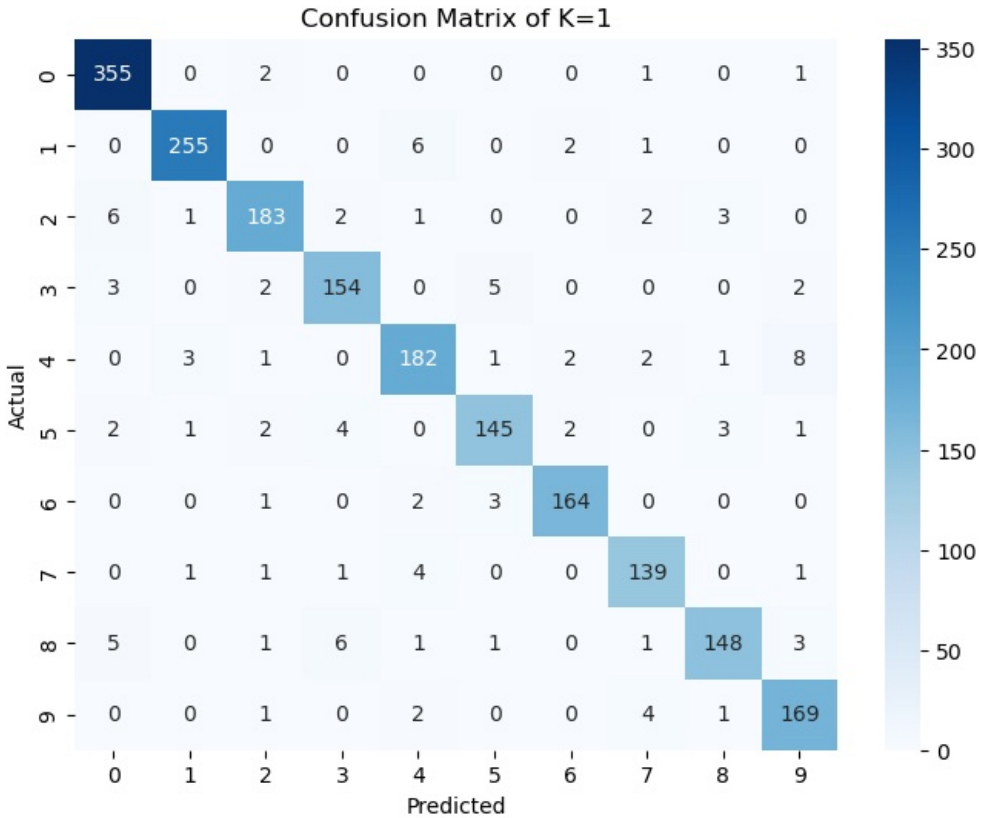
Cross Validation

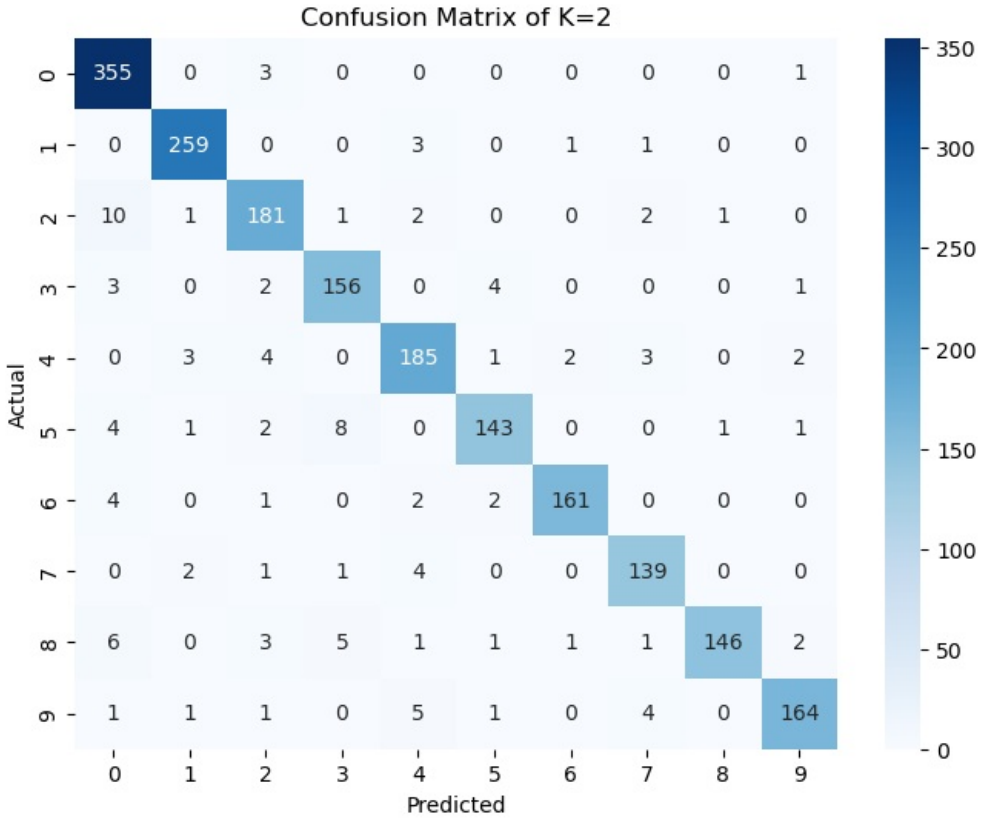
In [47]:
```
# Cross Validation
cv_values = []
for i in range(2,25):
    neigh = KNeighborsClassifier(n_neighbors = 10)
    cv_values = cross_val_score(estimator=neigh, X=X_train, y=y_train, scoring='accuracy', cv=i, n_jobs=-1)
    mean = cv_values.mean()
    print(f"Cross_validation results for {i} folds is {mean}")
```

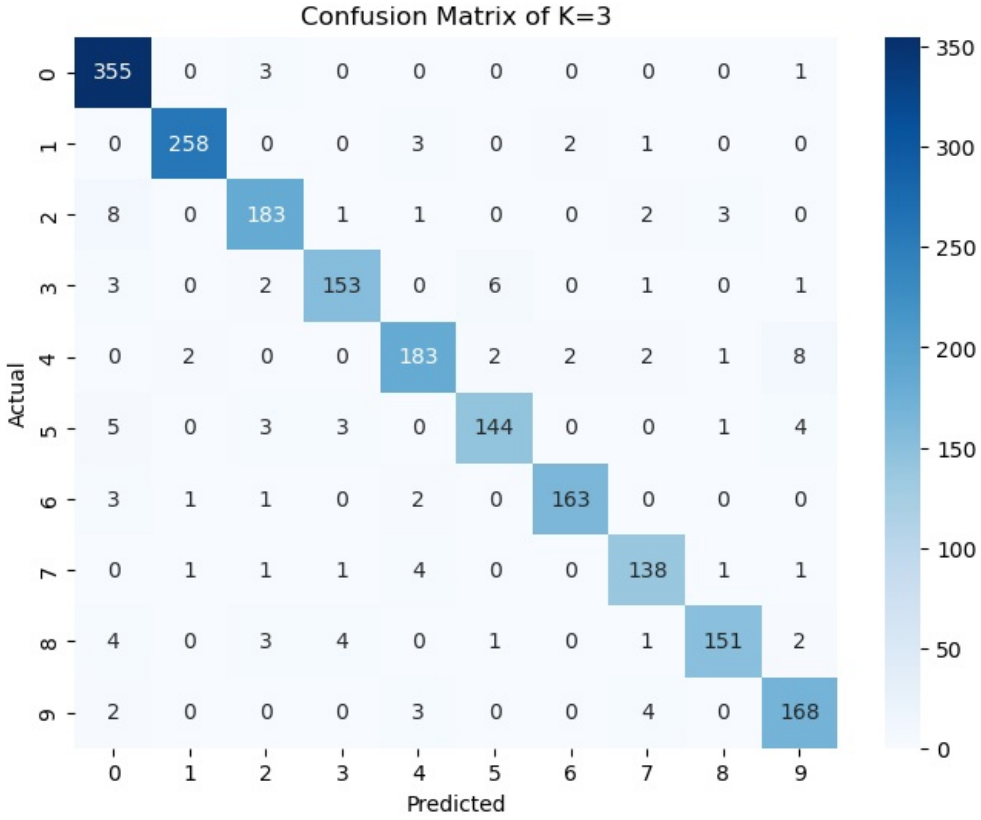
```
Cross_validation results for 2 folds is 0.9422572569522042
Cross_validation results for 3 folds is 0.9496634407309789
Cross_validation results for 4 folds is 0.9518577566922957
Cross_validation results for 5 folds is 0.9546006011596345
Cross_validation results for 6 folds is 0.9551489964623494
Cross_validation results for 7 folds is 0.9563832945215456
Cross_validation results for 8 folds is 0.9572063004313748
Cross_validation results for 9 folds is 0.9580301039039679
Cross_validation results for 10 folds is 0.9585784617697353
Cross_validation results for 11 folds is 0.9567971274031343
Cross_validation results for 12 folds is 0.9578930207520447
Cross_validation results for 13 folds is 0.9595407043936455
Cross_validation results for 14 folds is 0.9595418257366435
Cross_validation results for 15 folds is 0.9591271551420613
Cross_validation results for 16 folds is 0.9595385699826489
Cross_validation results for 17 folds is 0.9598161596512339
Cross_validation results for 18 folds is 0.9598130241169832
Cross_validation results for 19 folds is 0.9591277884659428
Cross_validation results for 20 folds is 0.9599491946409755
Cross_validation results for 21 folds is 0.9598107320704956
Cross_validation results for 22 folds is 0.9600872592264138
Cross_validation results for 23 folds is 0.9599506240570568
Cross_validation results for 24 folds is 0.9594072071680854
```

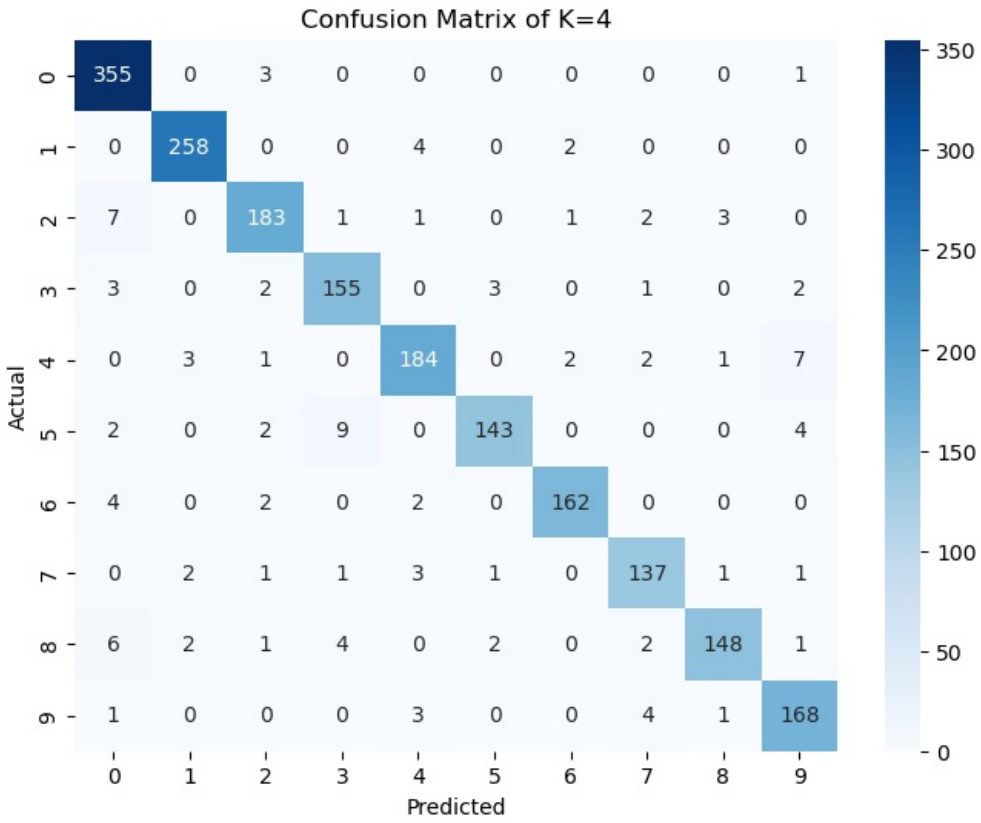
Confusion Matrix

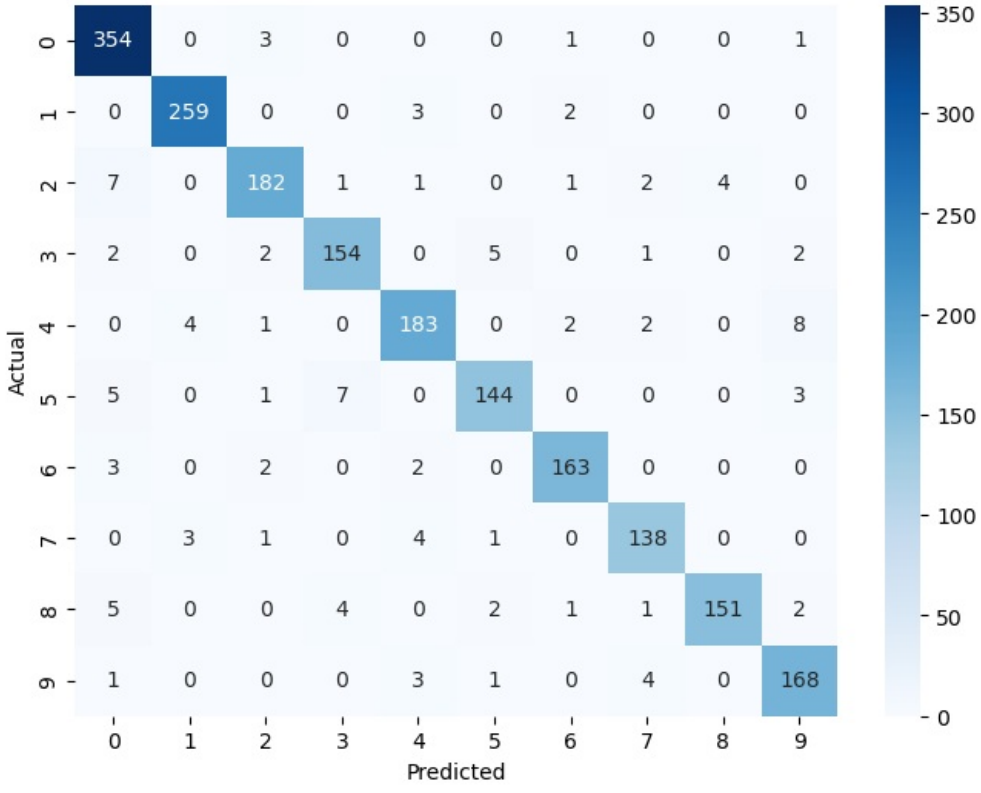
In [48]:
```
for k in range(1,25):
    #Create KNN Classifier
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train,y_train) # Train the KNN Classifier on the training set
    y_pred = neigh.predict(X_test) # Predict the X_test score and store in y_pred
    cm = confusion_matrix(y_test, y_pred) # Calculate the confusion_matrix
    class_labels = np.unique(np.concatenate((y_test, y_pred))) # Find the class_labels
    # Plot the confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix of K={k}")
    plt.show()
```

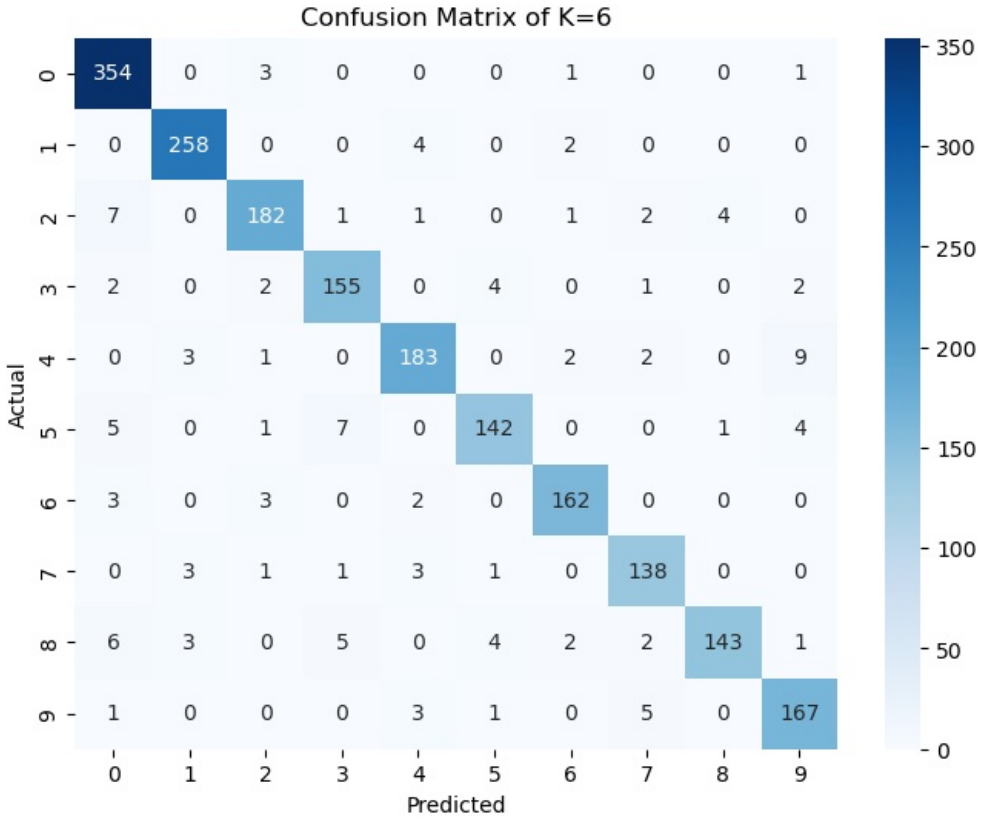


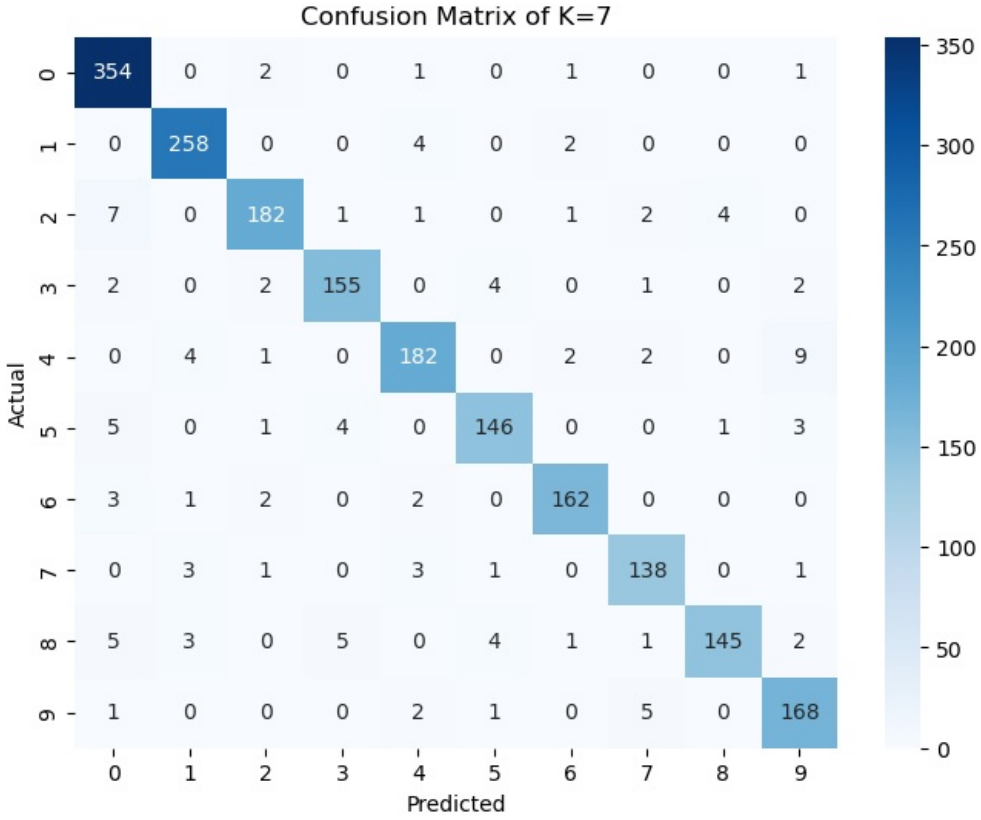

Confusion Matrix of K=2
Actual
Predicted
0 355 0 3 0 0 0 0 0 0 1
1 0 259 0 0 3 0 1 1 0 0
2 10 1 181 1 2 0 0 2 1 0
3 3 0 2 156 0 4 0 0 0 1
4 0 3 4 0 185 1 2 3 0 2
5 4 1 2 8 0 143 0 0 1 1
6 4 0 1 0 2 2 161 0 0 0
7 0 2 1 1 4 0 0 139 0 0
8 6 0 3 5 1 1 1 1 146 2
9 1 1 1 0 5 1 0 4 0 164
0 1 2 3 4 5 6 7 8 9
350 300 250 200 150 100 50 0

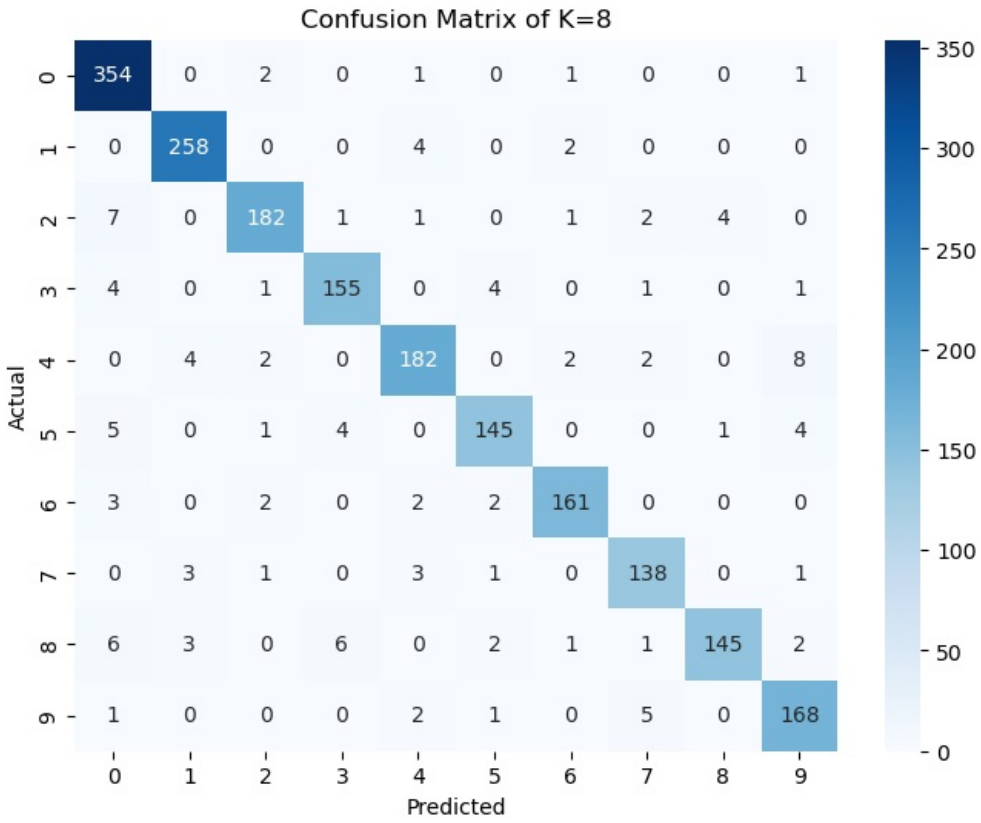

Confusion Matrix of K=3
Actual
Predicted
0 355 0 3 0 0 0 0 0 0 1
1 0 258 0 0 3 0 2 1 0 0
2 8 0 183 1 1 0 0 2 3 0
3 3 0 2 153 0 6 0 1 0 1
4 0 2 0 0 183 2 2 2 1 8
5 5 0 3 3 0 144 0 0 1 4
6 3 1 1 0 2 0 163 0 0 0
7 0 1 1 1 4 0 0 138 1 1
8 4 0 3 4 0 1 0 1 151 2
9 2 0 0 0 3 0 0 4 0 168
0 1 2 3 4 5 6 7 8 9
350 300 250 200 150 100 50 0

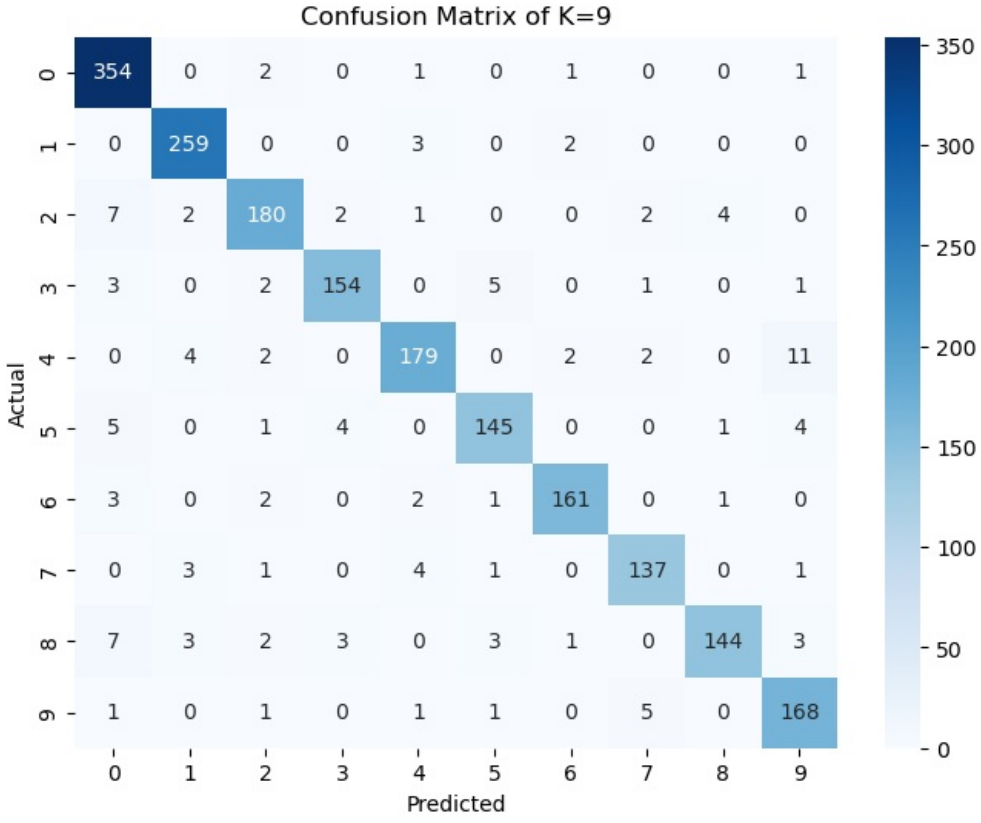

Confusion Matrix of K=4
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
355 0 3 0 0 0 0 0 0 1
0 258 0 0 4 0 2 0 0 0
7 0 183 1 1 0 1 2 3 0
3 0 2 155 0 3 0 1 0 2
0 3 1 0 184 0 2 2 1 7
2 0 2 9 0 143 0 0 0 4
4 0 2 0 2 0 162 0 0 0
0 2 1 1 3 1 0 137 1 1
6 2 1 4 0 2 0 2 148 1
1 0 0 0 3 0 0 4 1 168
350 300 250 200 150 100 50 0

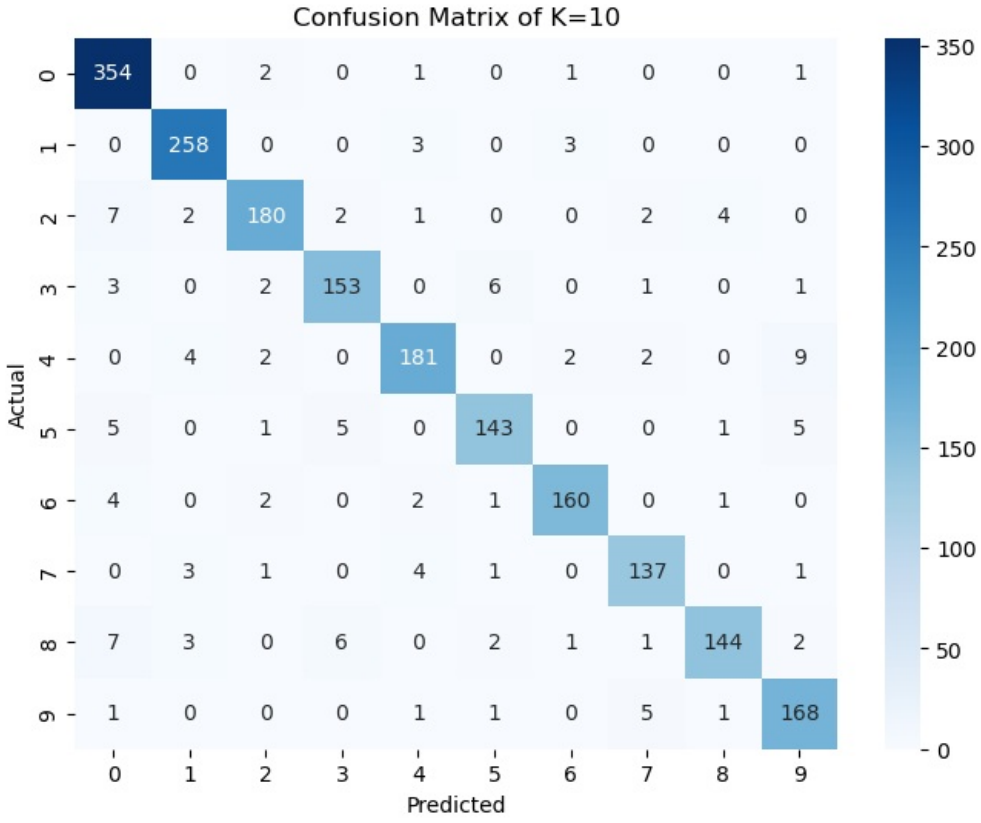

Confusion Matrix of K=5
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 3 0 0 0 1 0 0 1
0 259 0 0 3 0 2 0 0 0
7 0 182 1 1 0 1 2 4 0
2 0 2 154 0 5 0 1 0 2
0 4 1 0 183 0 2 2 0 8
5 0 1 7 0 144 0 0 0 3
3 0 2 0 2 0 163 0 0 0
0 3 1 0 4 1 0 138 0 0
5 0 0 4 0 2 1 1 151 2
1 0 0 0 3 1 0 4 0 168
350 300 250 200 150 100 50 0

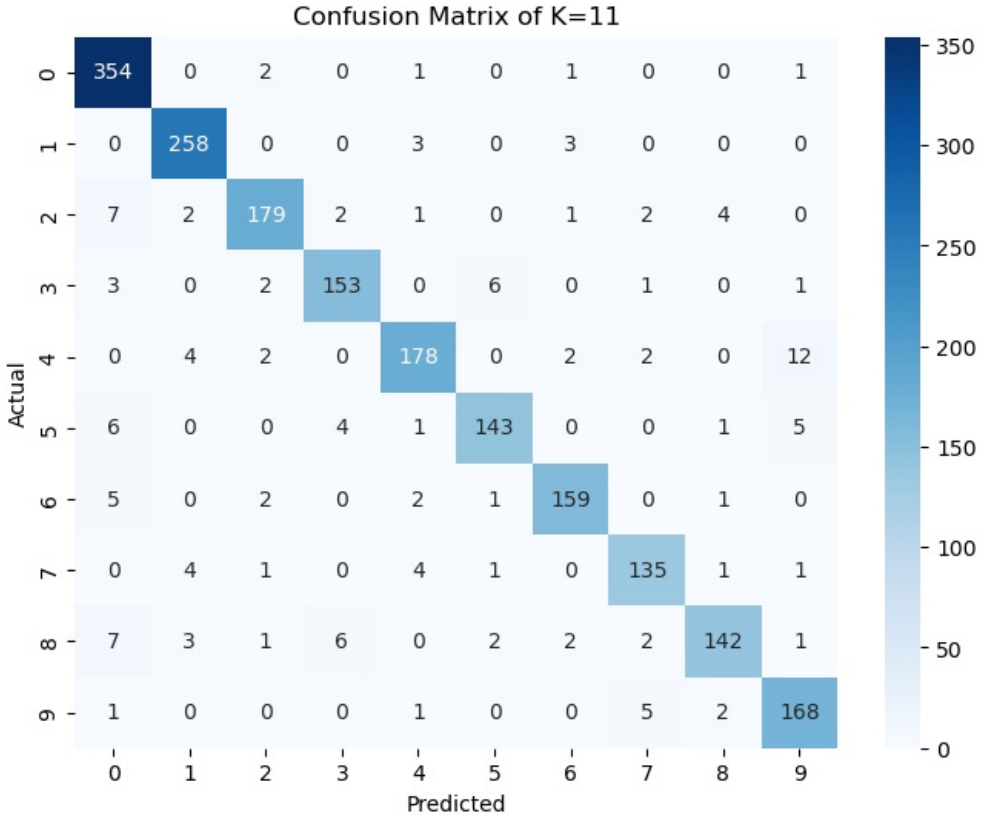

Confusion Matrix of K=6
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 354 0 3 0 0 0 1 0 0 1
1 0 258 0 0 4 0 2 0 0 0
2 7 0 182 1 1 0 1 2 4 0
3 2 0 2 155 0 4 0 1 0 2
4 0 3 1 0 183 0 2 2 0 9
5 5 0 1 7 0 142 0 0 1 4
6 3 0 3 0 2 0 162 0 0 0
7 0 3 1 1 3 1 0 138 0 0
8 6 3 0 5 0 4 2 2 143 1
9 1 0 0 0 3 1 0 5 0 167
350 300 250 200 150 100 50 0

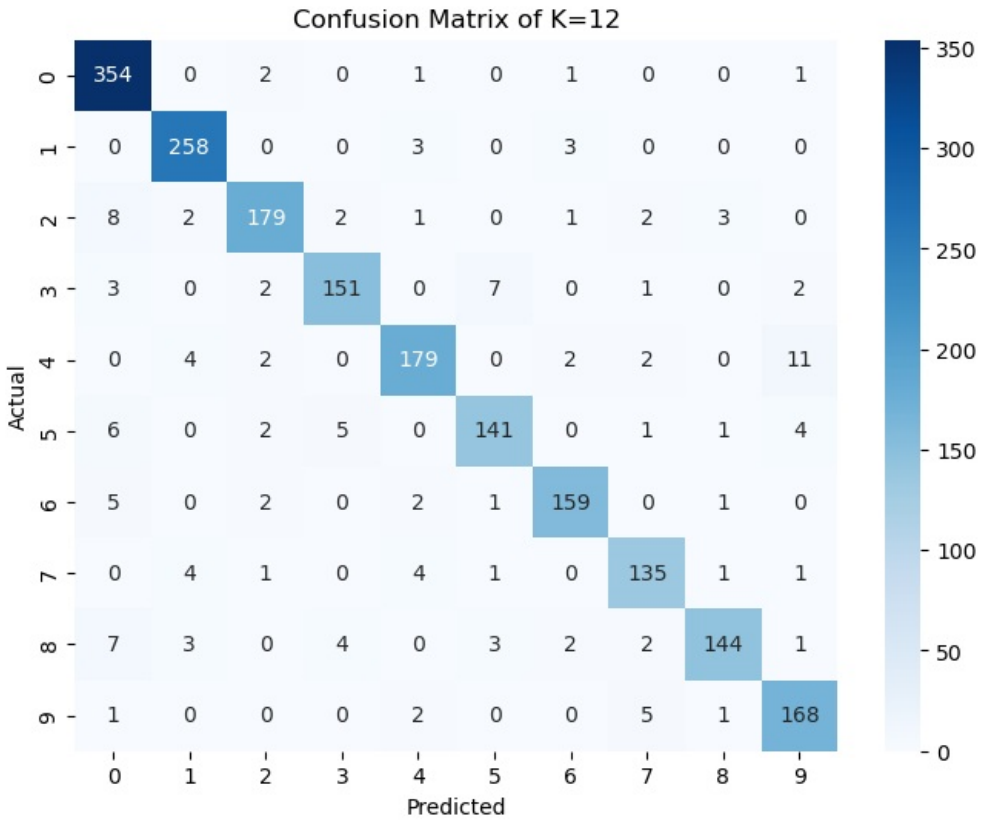

Confusion Matrix of K=7
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 354 0 2 0 1 0 1 0 0 1
1 0 258 0 0 4 0 2 0 0 0
2 7 0 182 1 1 0 1 2 4 0
3 2 0 2 155 0 4 0 1 0 2
4 0 4 1 0 182 0 2 2 0 9
5 5 0 1 4 0 146 0 0 1 3
6 3 1 2 0 2 0 162 0 0 0
7 0 3 1 0 3 1 0 138 0 1
8 5 3 0 5 0 4 1 1 145 2
9 1 0 0 0 2 1 0 5 0 168
350 300 250 200 150 100 50 0

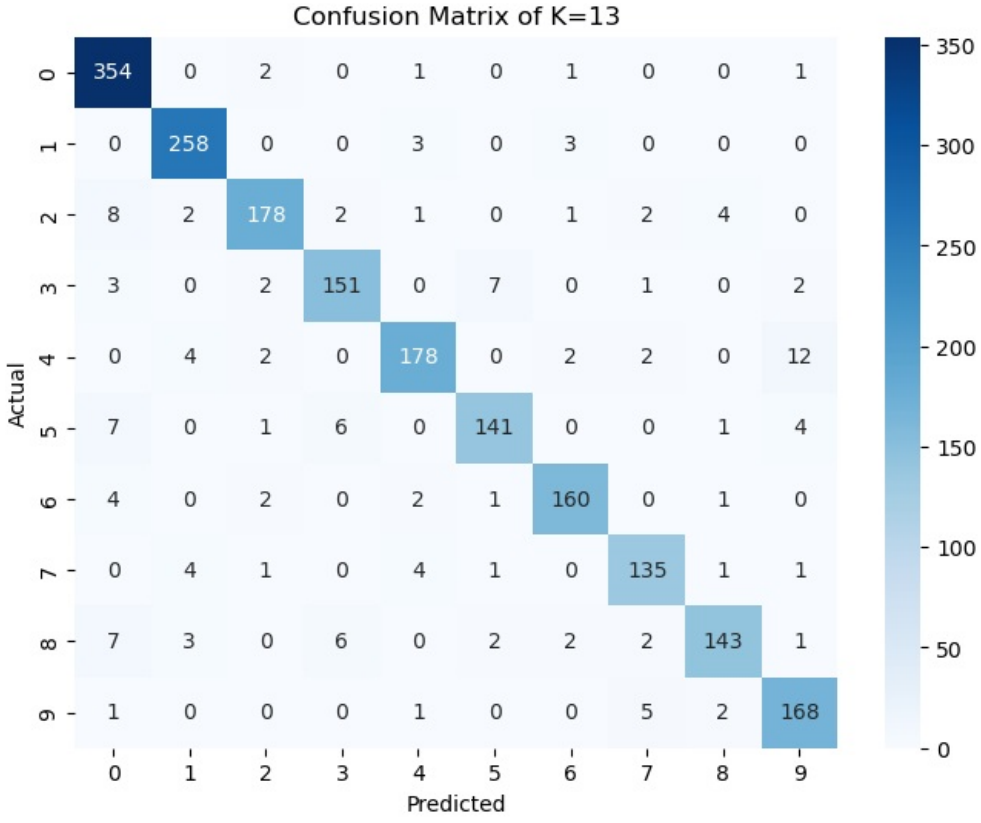

Confusion Matrix of K=8
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 354 0 2 0 1 0 1 0 0 1
1 0 258 0 0 4 0 2 0 0 0
2 7 0 182 1 1 0 1 2 4 0
3 4 0 1 155 0 4 0 1 0 1
4 0 4 2 0 182 0 2 2 0 8
5 5 0 1 4 0 145 0 0 1 4
6 3 0 2 0 2 2 161 0 0 0
7 0 3 1 0 3 1 0 138 0 1
8 6 3 0 6 0 2 1 1 145 2
9 1 0 0 0 2 1 0 5 0 168
350 300 250 200 150 100 50 0

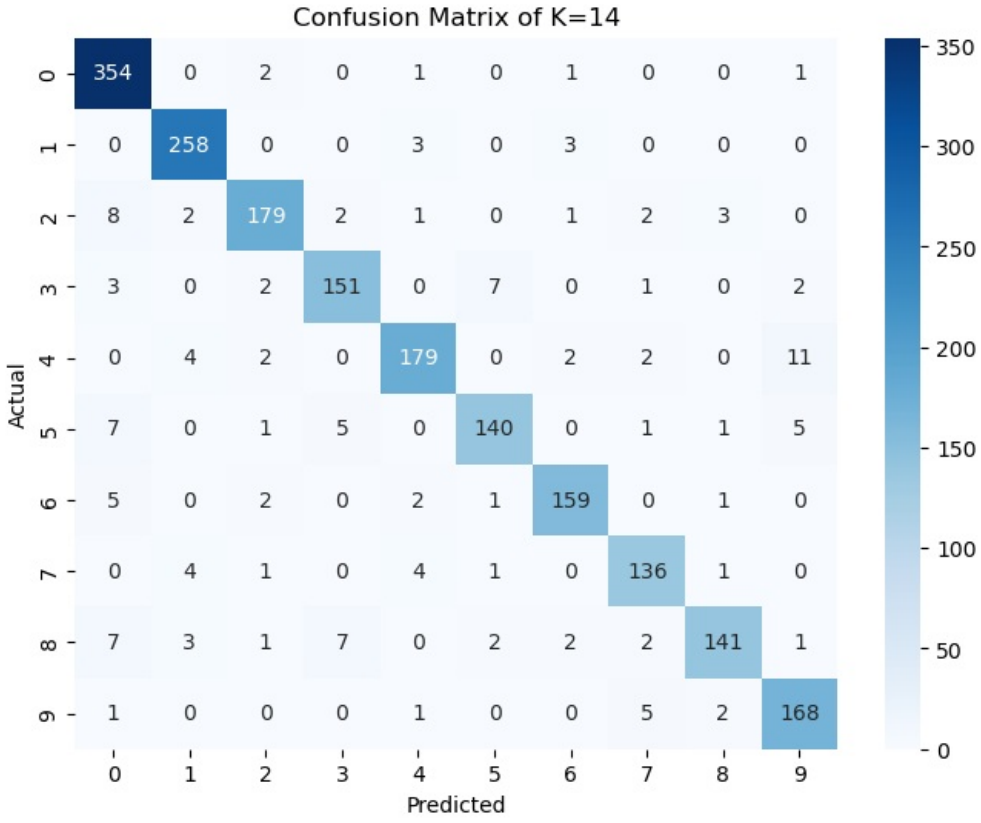

Confusion Matrix of K=9
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 354 0 2 0 1 0 1 0 0 1
1 0 259 0 0 3 0 2 0 0 0
2 7 2 180 2 1 0 0 2 4 0
3 3 0 2 154 0 5 0 1 0 1
4 0 4 2 0 179 0 2 2 0 11
5 5 0 1 4 0 145 0 0 1 4
6 3 0 2 0 2 1 161 0 1 0
7 0 3 1 0 4 1 0 137 0 1
8 7 3 2 3 0 3 1 0 144 3
9 1 0 1 0 1 1 0 5 0 168
350 300 250 200 150 100 50 0

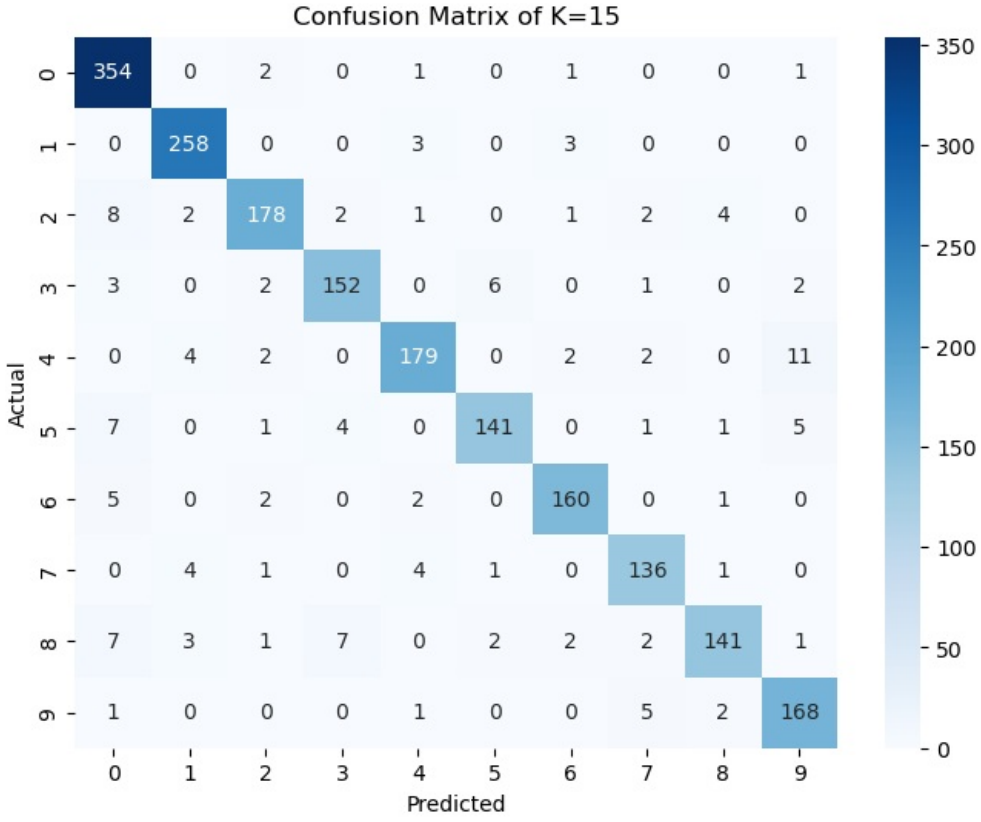

Confusion Matrix of K=10
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
7 2 180 2 1 0 0 2 4 0
3 0 2 153 0 6 0 1 0 1
0 4 2 0 181 0 2 2 0 9
5 0 1 5 0 143 0 0 1 5
4 0 2 0 2 1 160 0 1 0
0 3 1 0 4 1 0 137 0 1
7 3 0 6 0 2 1 1 144 2
1 0 0 0 1 1 0 5 1 168
350 300 250 200 150 100 50 0

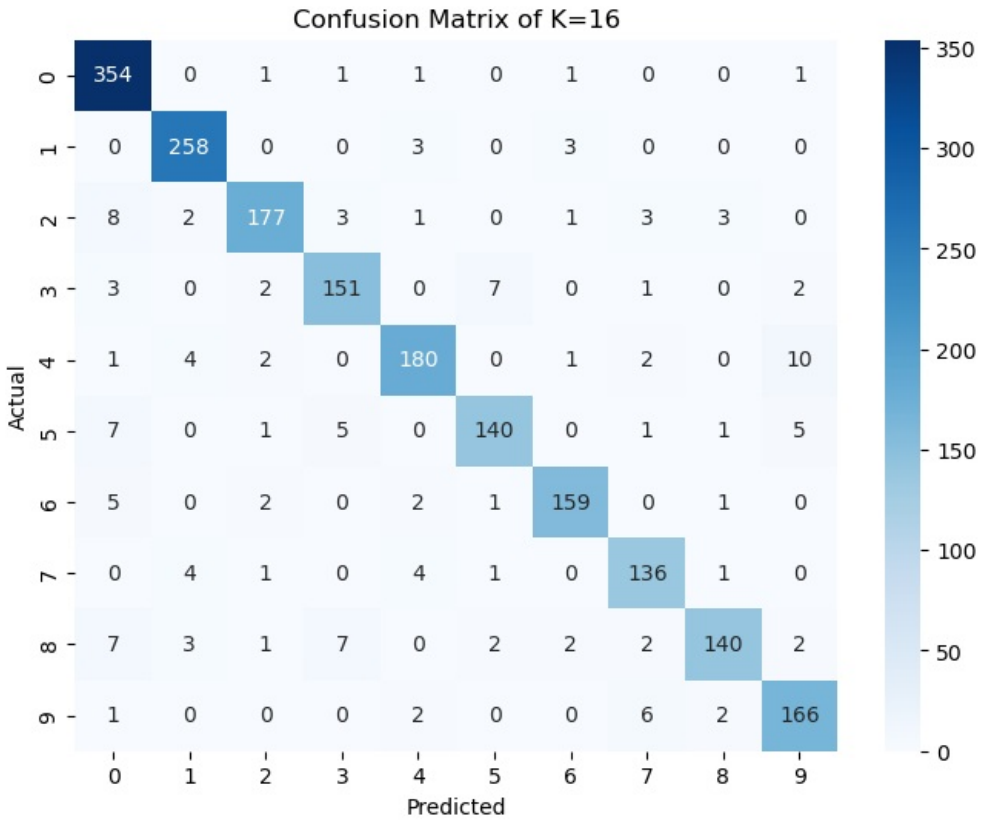

Confusion Matrix of K=11
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
7 2 179 2 1 0 1 2 4 0
3 0 2 153 0 6 0 1 0 1
0 4 2 0 178 0 2 2 0 12
6 0 0 4 1 143 0 0 1 5
5 0 2 0 2 1 159 0 1 0
0 4 1 0 4 1 0 135 1 1
7 3 1 6 0 2 2 2 142 1
1 0 0 0 1 0 0 5 2 168
350 300 250 200 150 100 50 0

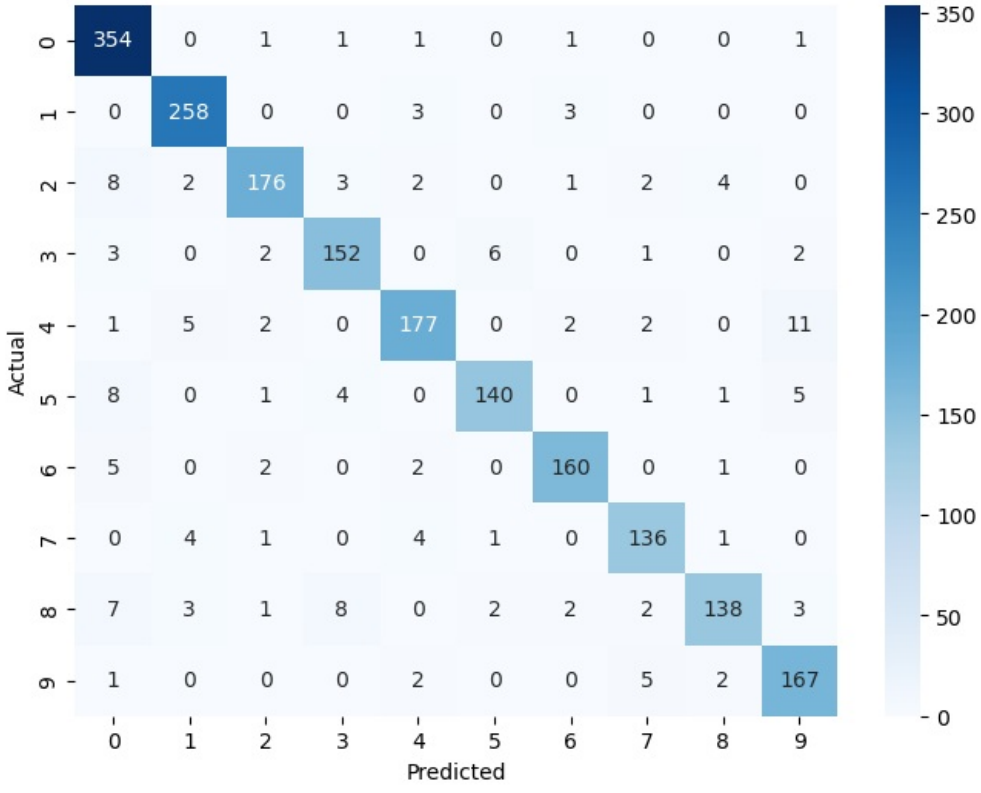

Confusion Matrix of K=12
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 179 2 1 0 1 2 3 0
3 0 2 151 0 7 0 1 0 2
0 4 2 0 179 0 2 2 0 11
6 0 2 5 0 141 0 1 1 4
5 0 2 0 2 1 159 0 1 0
0 4 1 0 4 1 0 135 1 1
7 3 0 4 0 3 2 2 144 1
1 0 0 0 2 0 0 5 1 168
350 300 250 200 150 100 50 0

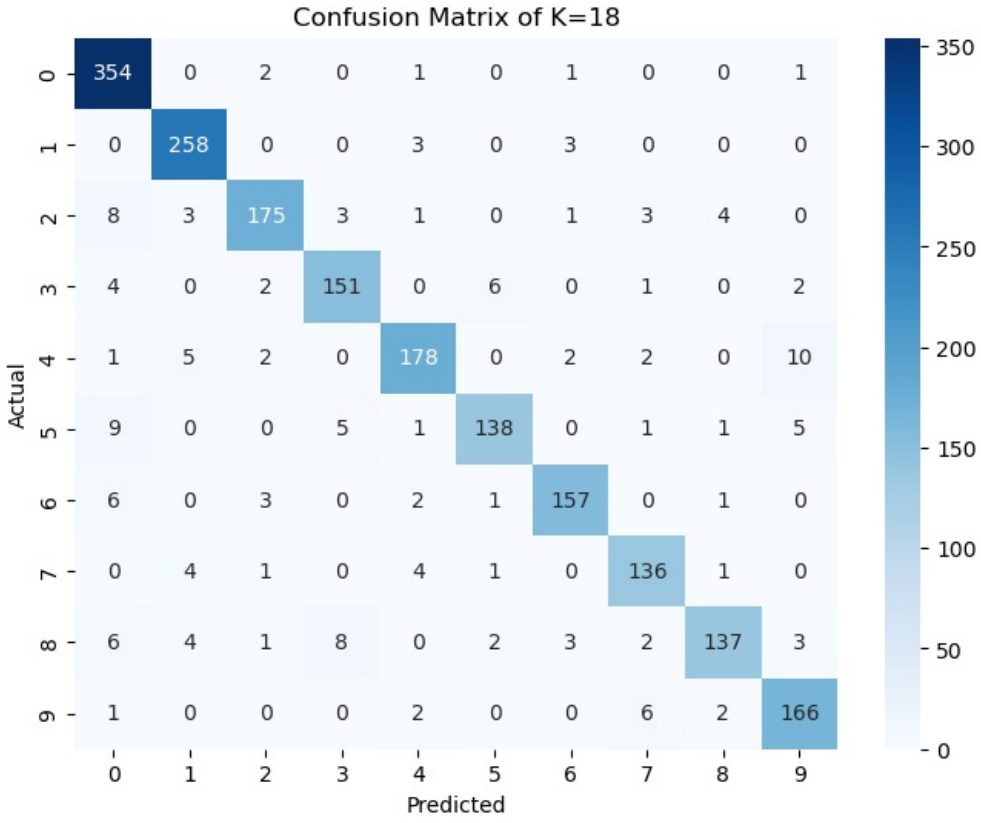

Confusion Matrix of K=13
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 178 2 1 0 1 2 4 0
3 0 2 151 0 7 0 1 0 2
0 4 2 0 178 0 2 2 0 12
7 0 1 6 0 141 0 0 1 4
4 0 2 0 2 1 160 0 1 0
0 4 1 0 4 1 0 135 1 1
7 3 0 6 0 2 2 2 143 1
1 0 0 0 1 0 0 5 2 168
350 300 250 200 150 100 50 0

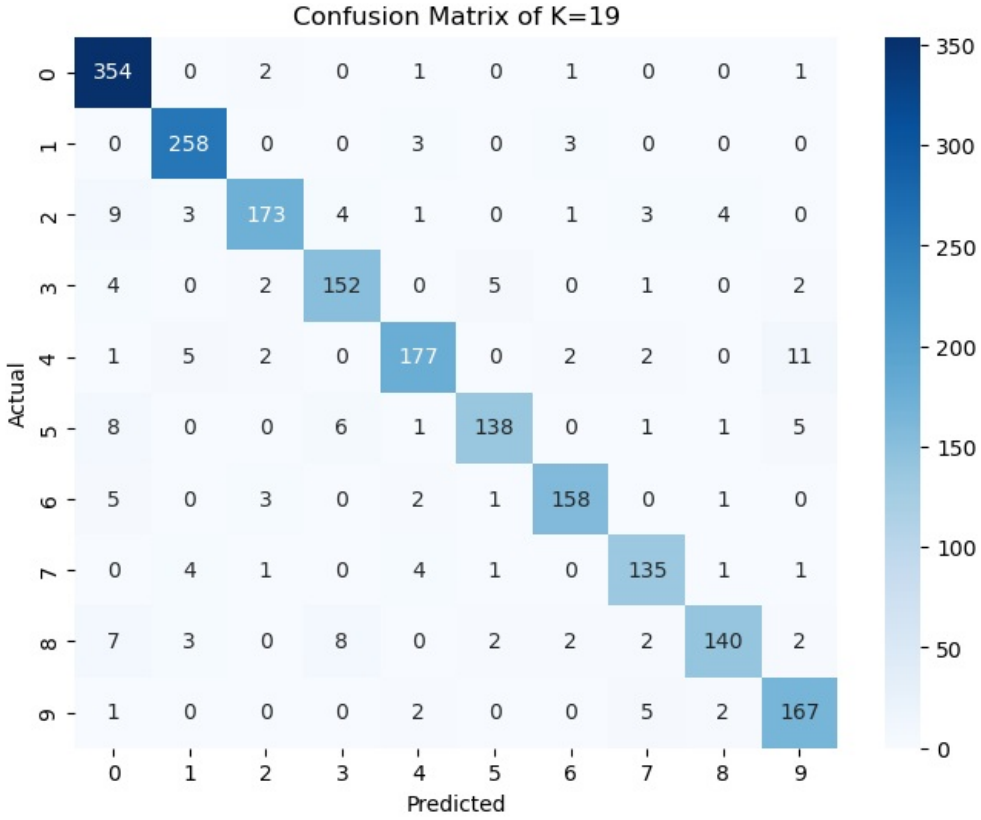

Confusion Matrix of K=14
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 179 2 1 0 1 2 3 0
3 0 2 151 0 7 0 1 0 2
0 4 2 0 179 0 2 2 0 11
7 0 1 5 0 140 0 1 1 5
5 0 2 0 2 1 159 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 1 7 0 2 2 2 141 1
1 0 0 0 1 0 0 5 2 168
350 300 250 200 150 100 50 0

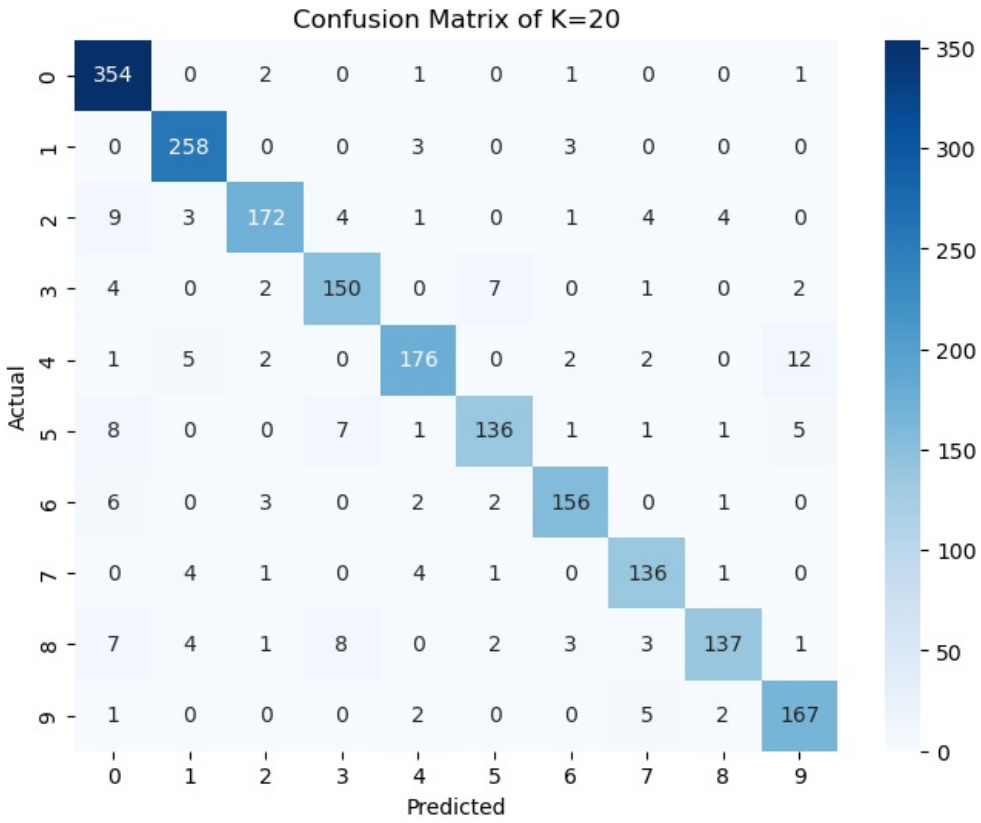

Confusion Matrix of K=15
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 178 2 1 0 1 2 4 0
3 0 2 152 0 6 0 1 0 2
0 4 2 0 179 0 2 2 0 11
7 0 1 4 0 141 0 1 1 5
5 0 2 0 2 0 160 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 1 7 0 2 2 2 141 1
1 0 0 0 1 0 0 5 2 168
350 300 250 200 150 100 50 0

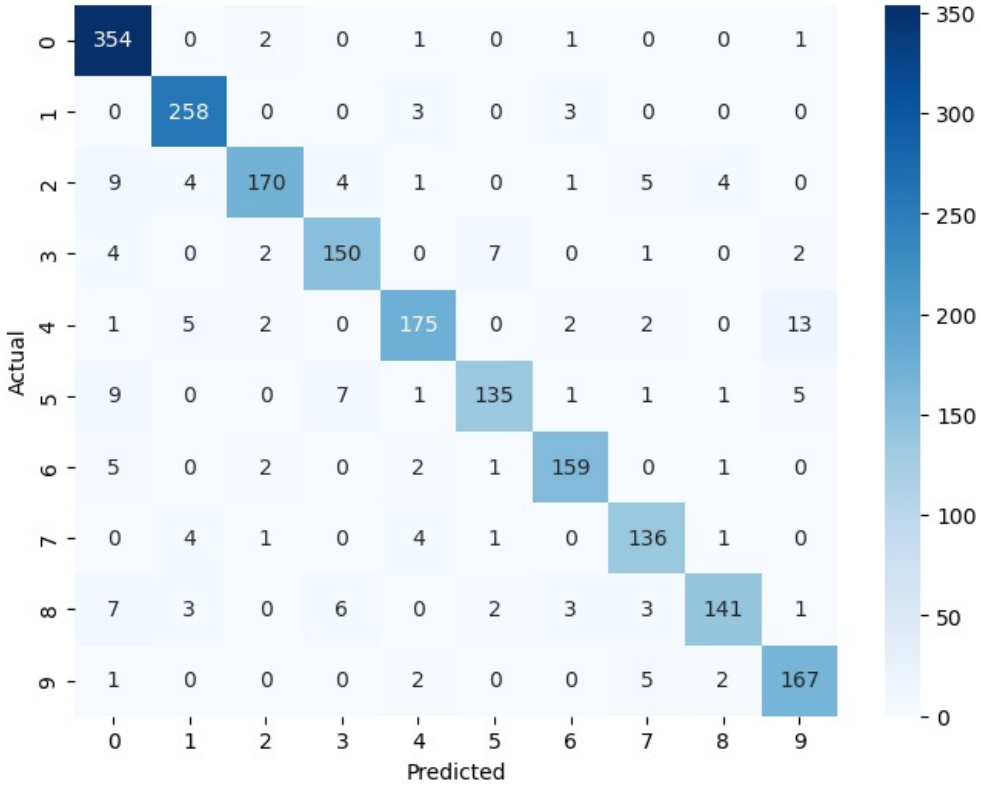

Confusion Matrix of K=16
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 1 1 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 177 3 1 0 1 3 3 0
3 0 2 151 0 7 0 1 0 2
1 4 2 0 180 0 1 2 0 10
7 0 1 5 0 140 0 1 1 5
5 0 2 0 2 1 159 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 1 7 0 2 2 2 140 2
1 0 0 0 2 0 0 6 2 166
350 300 250 200 150 100 50 0

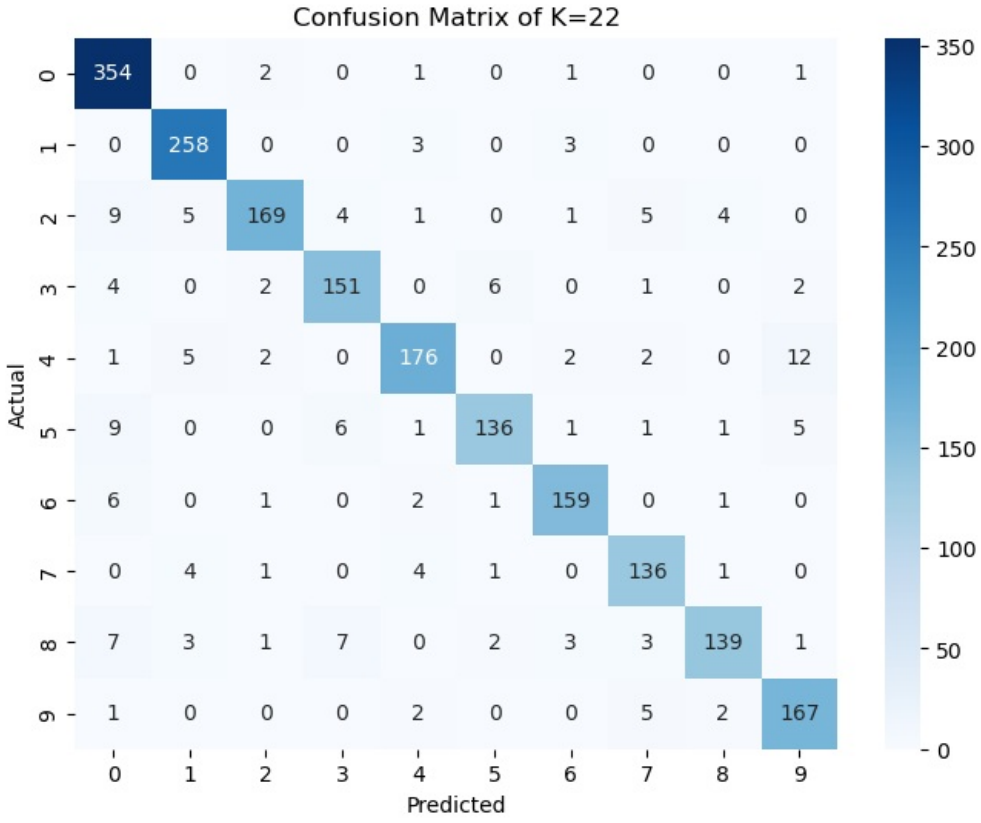

Confusion Matrix of K=17
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 1 1 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 2 176 3 2 0 1 2 4 0
3 0 2 152 0 6 0 1 0 2
1 5 2 0 177 0 2 2 0 11
8 0 1 4 0 140 0 1 1 5
5 0 2 0 2 0 160 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 1 8 0 2 2 2 138 3
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0

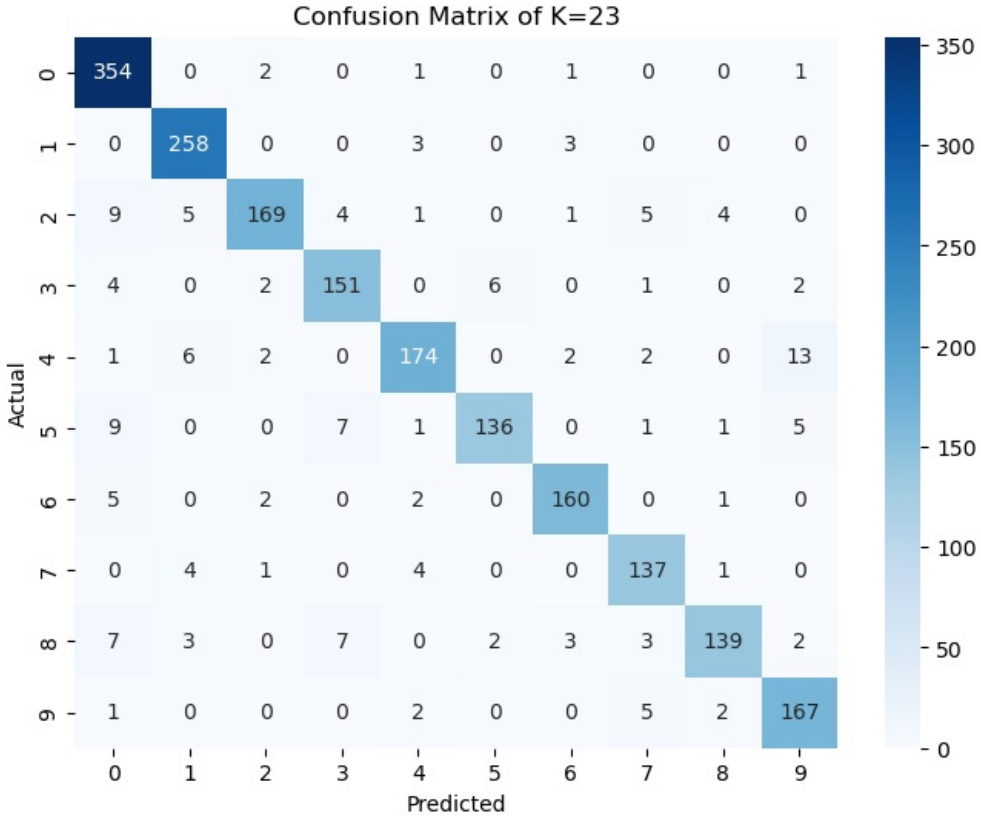

Confusion Matrix of K=18
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
8 3 175 3 1 0 1 3 4 0
4 0 2 151 0 6 0 1 0 2
1 5 2 0 178 0 2 2 0 10
9 0 0 5 1 138 0 1 1 5
6 0 3 0 2 1 157 0 1 0
0 4 1 0 4 1 0 136 1 0
6 4 1 8 0 2 3 2 137 3
1 0 0 0 2 0 0 6 2 166
350 300 250 200 150 100 50 0

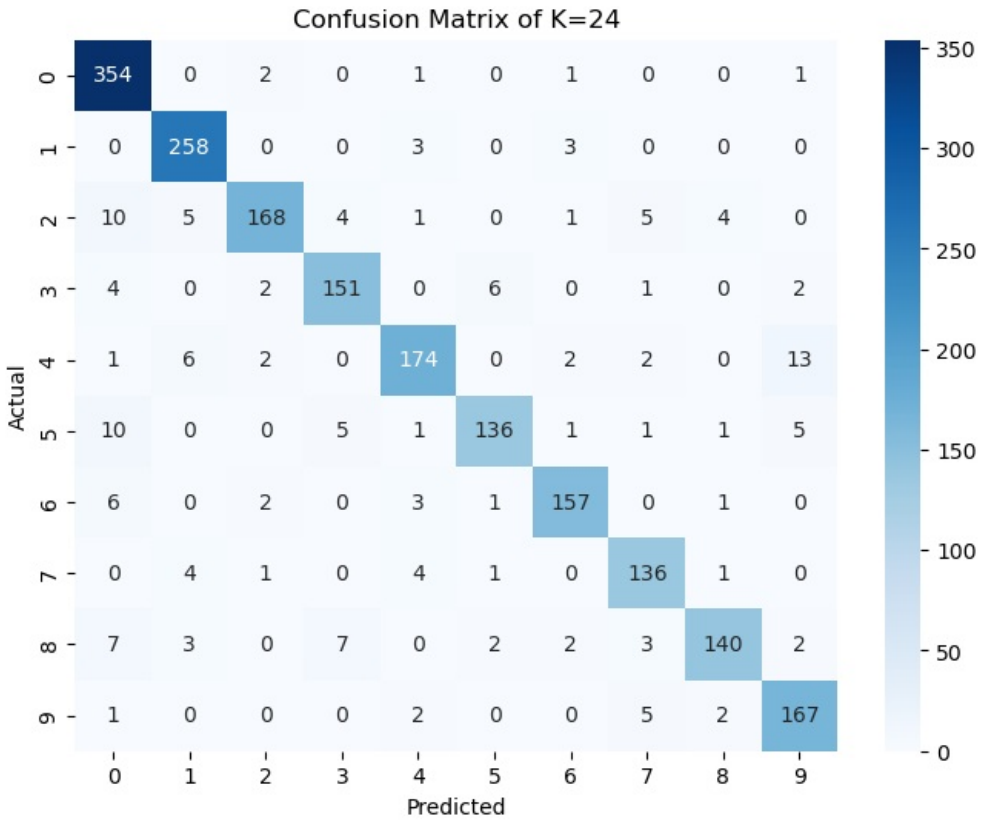

Confusion Matrix of K=19
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
9 3 173 4 1 0 1 3 4 0
4 0 2 152 0 5 0 1 0 2
1 5 2 0 177 0 2 2 0 11
8 0 0 6 1 138 0 1 1 5
5 0 3 0 2 1 158 0 1 0
0 4 1 0 4 1 0 135 1 1
7 3 0 8 0 2 2 2 140 2
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0


Confusion Matrix of K=20
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
9 3 172 4 1 0 1 4 4 0
4 0 2 150 0 7 0 1 0 2
1 5 2 0 176 0 2 2 0 12
8 0 0 7 1 136 1 1 1 5
6 0 3 0 2 2 156 0 1 0
0 4 1 0 4 1 0 136 1 0
7 4 1 8 0 2 3 3 137 1
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0


Confusion Matrix of K=21
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
9 4 170 4 1 0 1 5 4 0
4 0 2 150 0 7 0 1 0 2
1 5 2 0 175 0 2 2 0 13
9 0 0 7 1 135 1 1 1 5
5 0 2 0 2 1 159 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 0 6 0 2 3 3 141 1
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0


Confusion Matrix of K=22
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
9 5 169 4 1 0 1 5 4 0
4 0 2 151 0 6 0 1 0 2
1 5 2 0 176 0 2 2 0 12
9 0 0 6 1 136 1 1 1 5
6 0 1 0 2 1 159 0 1 0
0 4 1 0 4 1 0 136 1 0
7 3 1 7 0 2 3 3 139 1
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0


Confusion Matrix of K=23
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
354 0 2 0 1 0 1 0 0 1
0 258 0 0 3 0 3 0 0 0
9 5 169 4 1 0 1 5 4 0
4 0 2 151 0 6 0 1 0 2
1 6 2 0 174 0 2 2 0 13
9 0 0 7 1 136 0 1 1 5
5 0 2 0 2 0 160 0 1 0
0 4 1 0 4 0 0 137 1 0
7 3 0 7 0 2 3 3 139 2
1 0 0 0 2 0 0 5 2 167
350 300 250 200 150 100 50 0



Precision and Recall

In [49]:
```
for k in range(1,25):
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train,y_train) # Train the KNN Classifier on the training set
    y_pred = neigh.predict(X_test) # Predict the X_test score and store in y_pred
    precision=precision_score(y_test, y_pred=y_pred,average=None) # Calculate the precision score
    recall=recall_score(y_test, y_pred=y_pred,average=None) # Calculate the recall_score
    print(f"For K={k}, Precision={precision,}, Recall={recall,}") # Print Recall and precision
```

```
For K=1, Precision=(array([0.95687332, 0.97701149, 0.94329897, 0.92215569, 0.91919192,
       0.93548387, 0.96470588, 0.92666667, 0.94871795, 0.91351351]),), Recall=(array([0.98885794, 0.96590909, 0
.92424242, 0.92771084, 0.91      ,
       0.90625   , 0.96470588, 0.94557823, 0.89156627, 0.95480226]),)
For K=2, Precision=(array([0.92689295, 0.97003745, 0.91414141, 0.9122807 , 0.91584158,
       0.94078947, 0.97575758, 0.92666667, 0.98648649, 0.95906433]),), Recall=(array([0.98885794, 0.98106061, 0
.91414141, 0.93975904, 0.925     ,
       0.89375   , 0.94705882, 0.94557823, 0.87951807, 0.92655367]),)
For K=3, Precision=(array([0.93421053, 0.98473282, 0.93367347, 0.94444444, 0.93367347,
       0.94117647, 0.9760479 , 0.9261745 , 0.96178344, 0.90810811]),), Recall=(array([0.98885794, 0.97727273, 0
.92424242, 0.92168675, 0.915     ,
       0.9       , 0.95882353, 0.93877551, 0.90963855, 0.94915254]),)
For K=4, Precision=(array([0.93915344, 0.97358491, 0.93846154, 0.91176471, 0.93401015,
       0.95973154, 0.97005988, 0.92567568, 0.96103896, 0.91304348]),), Recall=(array([0.98885794, 0.97727273, 0
.92424242, 0.93373494, 0.92      ,
       0.89375   , 0.95294118, 0.93197279, 0.89156627, 0.94915254]),)
For K=5, Precision=(array([0.93899204, 0.97368421, 0.94791667, 0.92771084, 0.93367347,
       0.94117647, 0.95882353, 0.93243243, 0.97419355, 0.91304348]),), Recall=(array([0.98607242, 0.98106061, 0
.91919192, 0.92771084, 0.915     ,
       0.9       , 0.95882353, 0.93877551, 0.90963855, 0.94915254]),)
For K=6, Precision=(array([0.93650794, 0.96629213, 0.94300518, 0.91715976, 0.93367347,
       0.93421053, 0.95294118, 0.92      , 0.96621622, 0.9076087 ]),), Recall=(array([0.98607242, 0.97727273, 0
```

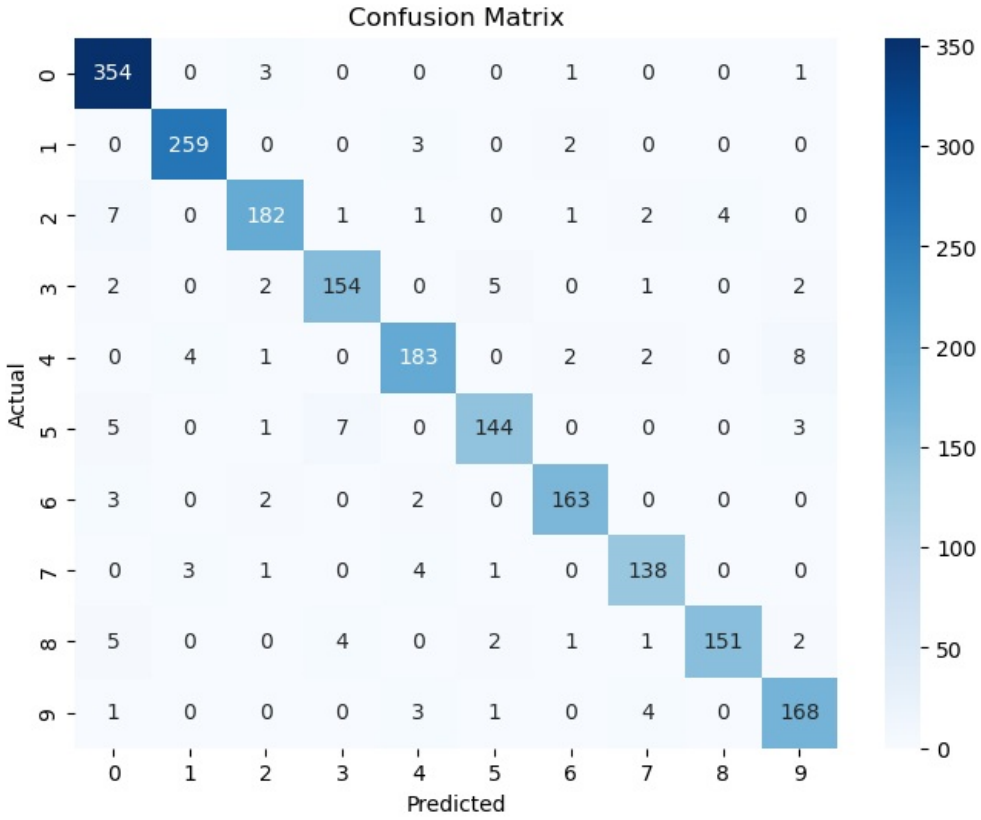
```
.91919192, 0.93373494, 0.915     ,
       0.8875    , 0.95294118, 0.93877551, 0.86144578, 0.94350282]),)
For K=7, Precision=(array([0.93899204, 0.95910781, 0.95287958, 0.93939394, 0.93333333,
       0.93589744, 0.95857988, 0.9261745 , 0.96666667, 0.90322581]),), Recall=(array([0.98607242, 0.97727273, 0
.91919192, 0.93373494, 0.91      ,
       0.9125    , 0.95294118, 0.93877551, 0.87349398, 0.94915254]),)
For K=8, Precision=(array([0.93157895, 0.96268657, 0.95287958, 0.93373494, 0.93333333,
       0.93548387, 0.95833333, 0.9261745 , 0.96666667, 0.90810811]),), Recall=(array([0.98607242, 0.97727273, 0
.91919192, 0.93373494, 0.91      ,
       0.90625   , 0.94705882, 0.93877551, 0.87349398, 0.94915254]),)
For K=9, Precision=(array([0.93157895, 0.95571956, 0.93264249, 0.94478528, 0.93717277,
       0.92948718, 0.96407186, 0.93197279, 0.96      , 0.88888889]),), Recall=(array([0.98607242, 0.98106061, 0
.90909091, 0.92771084, 0.895     ,
       0.90625   , 0.94705882, 0.93197279, 0.86746988, 0.94915254]),)
For K=10, Precision=(array([0.92913386, 0.95555556, 0.94736842, 0.92168675, 0.93782383,
       0.92857143, 0.95808383, 0.92567568, 0.95364238, 0.89839572]),), Recall=(array([0.98607242, 0.97727273, 0
.90909091, 0.92168675, 0.905     ,
       0.89375   , 0.94117647, 0.93197279, 0.86746988, 0.94915254]),)
For K=11, Precision=(array([0.92428198, 0.95202952, 0.94708995, 0.92727273, 0.93193717,
       0.93464052, 0.94642857, 0.91836735, 0.94039735, 0.88888889]),), Recall=(array([0.98607242, 0.97727273, 0
.9040404 , 0.92168675, 0.89      ,
       0.89375   , 0.93529412, 0.91836735, 0.85542169, 0.94915254]),)
For K=12, Precision=(array([0.921875  , 0.95202952, 0.94210526, 0.93209877, 0.93229167,
       0.92156863, 0.94642857, 0.91216216, 0.95364238, 0.89361702]),), Recall=(array([0.98607242, 0.97727273, 0
.9040404 , 0.90963855, 0.895     ,
       0.88125   , 0.93529412, 0.91836735, 0.86746988, 0.94915254]),)
For K=13, Precision=(array([0.921875  , 0.95202952, 0.94680851, 0.91515152, 0.93684211,
       0.92763158, 0.94674556, 0.91836735, 0.94078947, 0.88888889]),), Recall=(array([0.98607242, 0.97727273, 0
.8989899 , 0.90963855, 0.89      ,
       0.88125   , 0.94117647, 0.91836735, 0.86144578, 0.94915254]),)
For K=14, Precision=(array([0.91948052, 0.95202952, 0.94210526, 0.91515152, 0.93717277,
       0.92715232, 0.94642857, 0.91275168, 0.94630872, 0.89361702]),), Recall=(array([0.98607242, 0.97727273, 0
.9040404 , 0.90963855, 0.895     ,
       0.875     , 0.93529412, 0.92517007, 0.84939759, 0.94915254]),)
For K=15, Precision=(array([0.91948052, 0.95202952, 0.94179894, 0.92121212, 0.93717277,
       0.94      , 0.94674556, 0.91275168, 0.94      , 0.89361702]),), Recall=(array([0.98607242, 0.97727273, 0
.8989899 , 0.91566265, 0.895     ,
       0.88125   , 0.94117647, 0.92517007, 0.84939759, 0.94915254]),)
For K=16, Precision=(array([0.91709845, 0.95202952, 0.94652406, 0.90419162, 0.93264249,
       0.92715232, 0.95209581, 0.90066225, 0.94594595, 0.89247312]),), Recall=(array([0.98607242, 0.97727273, 0
.89393939, 0.90963855, 0.9       ,
       0.875     , 0.93529412, 0.92517007, 0.84337349, 0.93785311]),)
For K=17, Precision=(array([0.91472868, 0.94852941, 0.94623656, 0.9047619 , 0.92670157,
       0.93959732, 0.94674556, 0.91275168, 0.93877551, 0.88359788]),), Recall=(array([0.98607242, 0.97727273, 0
.88888889, 0.91566265, 0.885     ,
       0.875     , 0.94117647, 0.92517007, 0.8313253 , 0.94350282]),)
For K=18, Precision=(array([0.91002571, 0.94160584, 0.94086022, 0.90419162, 0.92708333,
       0.93243243, 0.94011976, 0.90066225, 0.93835616, 0.88770053]),), Recall=(array([0.98607242, 0.97727273, 0
.88383838, 0.90963855, 0.89      ,
       0.8625    , 0.92352941, 0.92517007, 0.8253012 , 0.93785311]),)
For K=19, Precision=(array([0.91002571, 0.94505495, 0.94535519, 0.89411765, 0.92670157,
       0.93877551, 0.94610778, 0.90604027, 0.93959732, 0.88359788]),), Recall=(array([0.98607242, 0.97727273, 0
.87373737, 0.91566265, 0.885     ,
       0.8625    , 0.92941176, 0.91836735, 0.84337349, 0.94350282]),)
For K=20, Precision=(array([0.90769231, 0.94160584, 0.93989071, 0.88757396, 0.92631579,
       0.91891892, 0.93413174, 0.89473684, 0.93835616, 0.88829787]),), Recall=(array([0.98607242, 0.97727273, 0
.86868687, 0.90361446, 0.88      ,
       0.85      , 0.91764706, 0.92517007, 0.8253012 , 0.94350282]),)
For K=21, Precision=(array([0.90769231, 0.94160584, 0.94972067, 0.89820359, 0.92592593,
       0.92465753, 0.93529412, 0.88888889, 0.94      , 0.88359788]),), Recall=(array([0.98607242, 0.97727273, 0
.85858586, 0.90361446, 0.875     ,
       0.84375   , 0.93529412, 0.92517007, 0.84939759, 0.94350282]),)
For K=22, Precision=(array([0.90537084, 0.93818182, 0.9494382 , 0.89880952, 0.92631579,
       0.93150685, 0.93529412, 0.88888889, 0.93918919, 0.88829787]),), Recall=(array([0.98607242, 0.97727273, 0
.85353535, 0.90963855, 0.88      ,
       0.85      , 0.93529412, 0.92517007, 0.8373494 , 0.94350282]),)
For K=23, Precision=(array([0.90769231, 0.93478261, 0.9494382 , 0.89349112, 0.92553191,
       0.94444444, 0.94117647, 0.88961039, 0.93918919, 0.87894737]),), Recall=(array([0.98607242, 0.97727273, 0
.85353535, 0.90963855, 0.87      ,
       0.85      , 0.94117647, 0.93197279, 0.8373494 , 0.94350282]),)
For K=24, Precision=(array([0.90076336, 0.93478261, 0.94915254, 0.90419162, 0.92063492,
       0.93150685, 0.94011976, 0.88888889, 0.93959732, 0.87894737]),), Recall=(array([0.98607242, 0.97727273, 0
.84848485, 0.90963855, 0.87      ,
       0.85      , 0.92352941, 0.92517007, 0.84337349, 0.94350282]),)
```

In [50]:
```
# Choose K = 5 because it has best cross validation score and does not lead to overfitting
best_k = 5
neigh = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1) # Use KNeighborsClassifier for K=5
neigh.fit(X_train,y_train) # fit the model
```

Out[50]:
```
KNeighborsClassifier(n_jobs=-1)
```

In [51]:
```
y_pred = neigh.predict(X_test) # Predict the X_test score and store in y_pred


cm = confusion_matrix(y_test, y_pred) # Calculate the confusion_matrix
```

```
class_labels = np.unique(np.concatenate((y_test, y_pred))) #Find the class_labels

# Plot the confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [52]:
```
precision_score(y_test, y_pred=y_pred,average=None) # Print the precision score
```

Out[52]:
```
array([0.93899204, 0.97368421, 0.94791667, 0.92771084, 0.93367347,
       0.94117647, 0.95882353, 0.93243243, 0.97419355, 0.91304348])
```

In [53]:
```
recall_score(y_test, y_pred=y_pred,average=None) # Print the recall_score
```

Out[53]:
```
array([0.98607242, 0.98106061, 0.91919192, 0.92771084, 0.915     ,
       0.9       , 0.95882353, 0.93877551, 0.90963855, 0.94915254])
```

Hog Image feature Extraction and KNN Classifier

In [54]:
```
# Import the required library
import numpy as np
from skimage.feature import hog
from skimage import exposure
hog_feat = []

orientations = 8  #Declaring Number of orientation bins
pixels_per_cell = (2, 2)  #Setting Cell size in pixels
cells_per_block = (2, 2)  #Setting Number of cells per block

# Define hog features in X_train
for image in X_train:
    features = hog(
        image.reshape((16, 16)),
        orientations=orientations,
        pixels_per_cell=pixels_per_cell,
        cells_per_block=cells_per_block)
    hog_feat.append(features) # Append HOG features to hog_feat

hog_feat = np.array(hog_feat) # Assign hog_feat array to hog_feat

hog_te = [] # Empty List
```

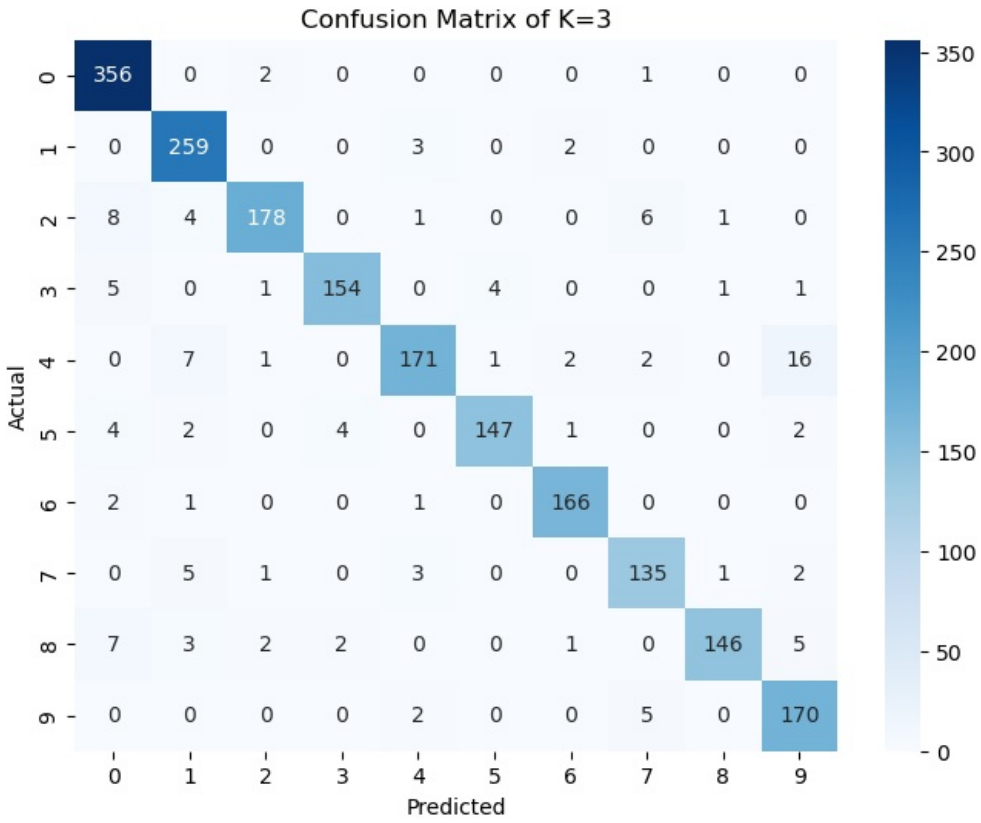
In [55]:
```
# Define hog features in X_test
for image in X_test:
    features = hog(
        image.reshape((16, 16)),
```

```
        orientations=orientations,
        pixels_per_cell=pixels_per_cell,
        cells_per_block=cells_per_block)

    hog_te.append(features) # Append HOG features
```

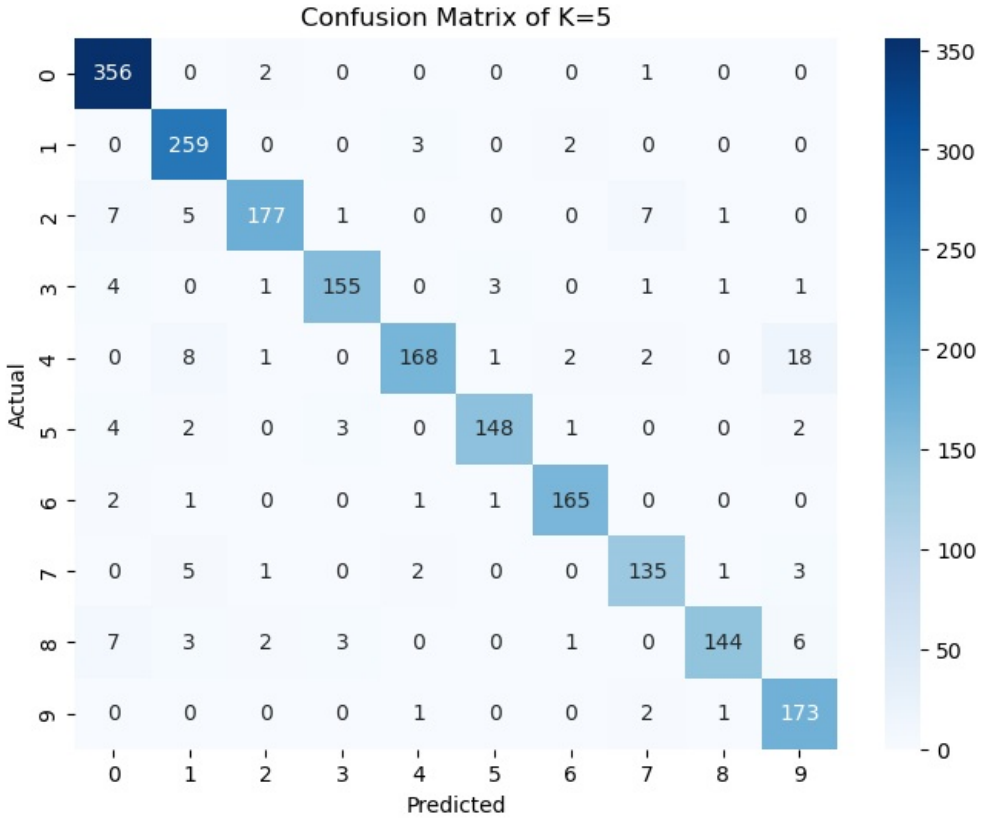
In [56]:

```
scores = [] # Empty list
l = []      # Empty list 'l'
# Define different k values
knn = [3, 5, 7, 9, 11, 13, 15, 17]

for k in knn:
    l.append(k)
    neigh = KNeighborsClassifier(n_neighbors=k) # KNN Classifier
    neigh.fit(hog_feat,y_train) # Train the KNN Classifier
    y_pred = neigh.predict(hog_te) # Predict the hog_te score
    scores.append(neigh.score(hog_te,y_test))# Accuracy
    accuracy = accuracy_score(y_test, y_pred=y_pred)
    print(f"Accuracy of K={k} is {accuracy}")
    cm = confusion_matrix(y_test, y_pred) # confusion_matrix
    class_labels = np.unique(np.concatenate((y_test, y_pred))) # class_labels
    # Plot confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
    plt.xlabel("Predicted") # Xlabel
    plt.ylabel("Actual")    # Ylabel
    plt.title(f"Confusion Matrix of K={k}") # Title
    plt.show()
    precision=precision_score(y_test, y_pred=y_pred,average=None) # Calculate Precision
    recall=recall_score(y_test, y_pred=y_pred,average=None)       # Calculate Recall
    print(f"For K={k}, Precision={precision,}, Recall={recall,}") # Print Precision and Recall
```

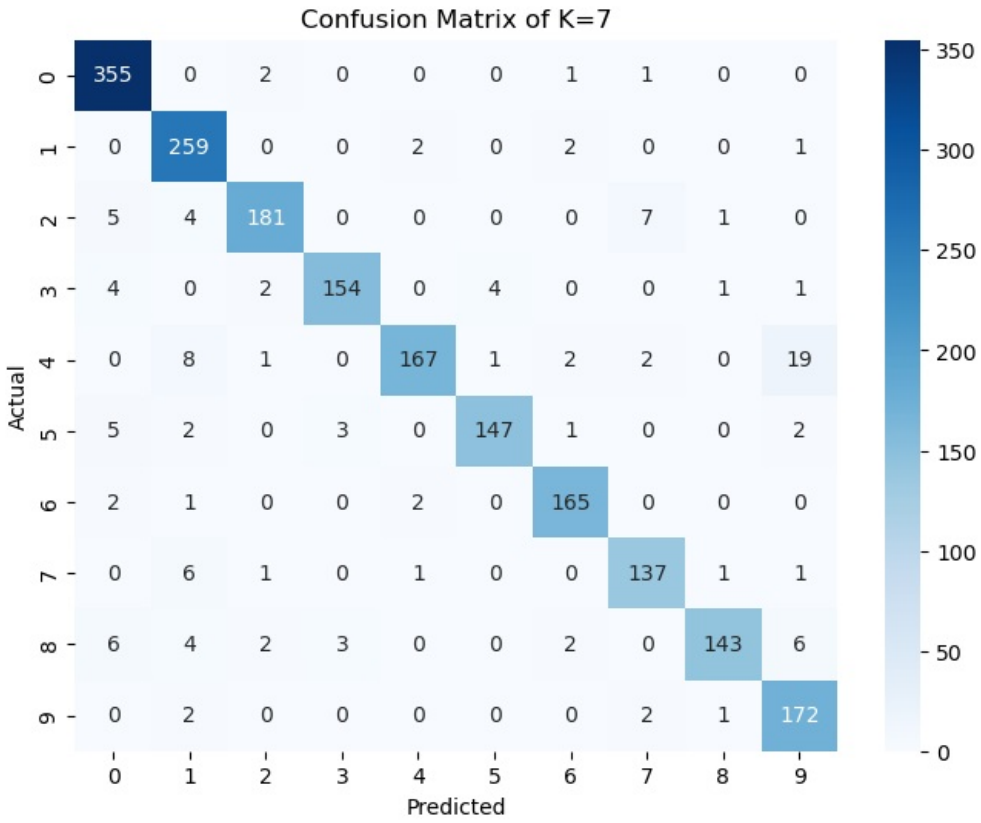
```
Accuracy of K=3 is 0.9377179870453413
```



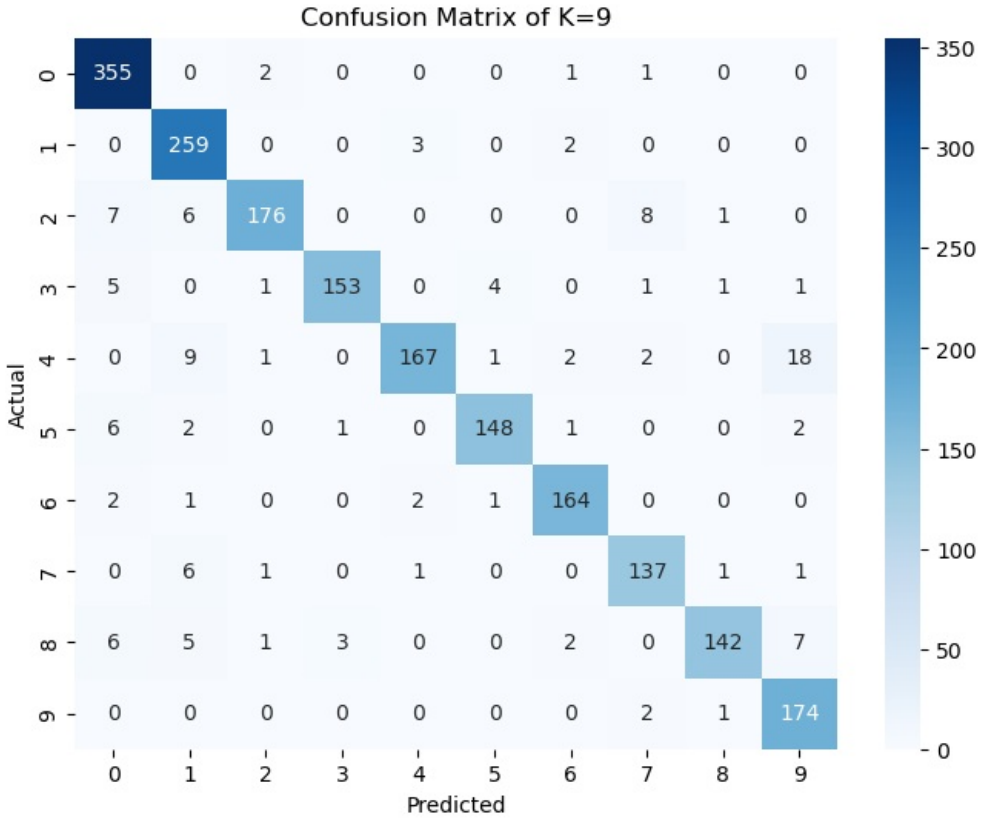

```
For K=3, Precision=(array([0.93193717, 0.92170819, 0.96216216, 0.9625    , 0.94475138,
       0.96710526, 0.96511628, 0.90604027, 0.97986577, 0.86734694]),), Recall=(array([0.99164345, 0.98106061, 0
.8989899 , 0.92771084, 0.855     ,
       0.91875   , 0.97647059, 0.91836735, 0.87951807, 0.96045198]),)
Accuracy of K=5 is 0.9367214748380668
```



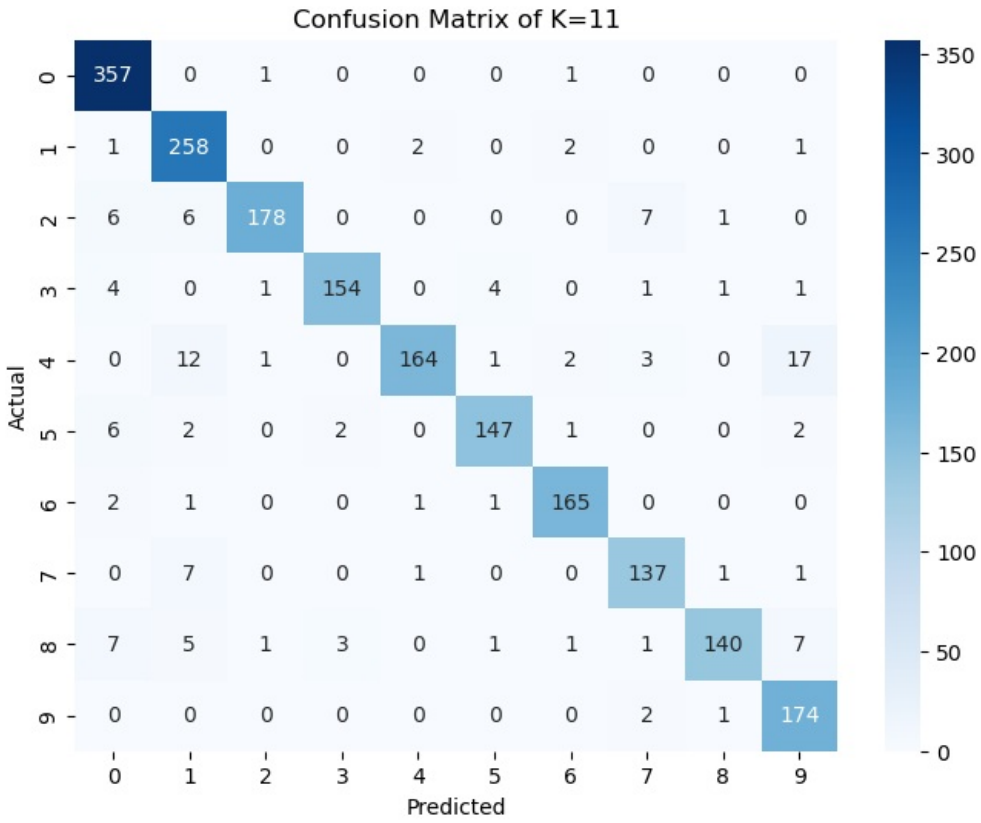

```
For K=5, Precision=(array([0.93684211, 0.91519435, 0.96195652, 0.95679012, 0.96      ,
       0.96732026, 0.96491228, 0.91216216, 0.97297297, 0.85221675]),), Recall=(array([0.99164345, 0.98106061, 0
.89393939, 0.93373494, 0.84      ,
       0.925     , 0.97058824, 0.91836735, 0.86746988, 0.97740113]),)
Accuracy of K=7 is 0.9367214748380668
```



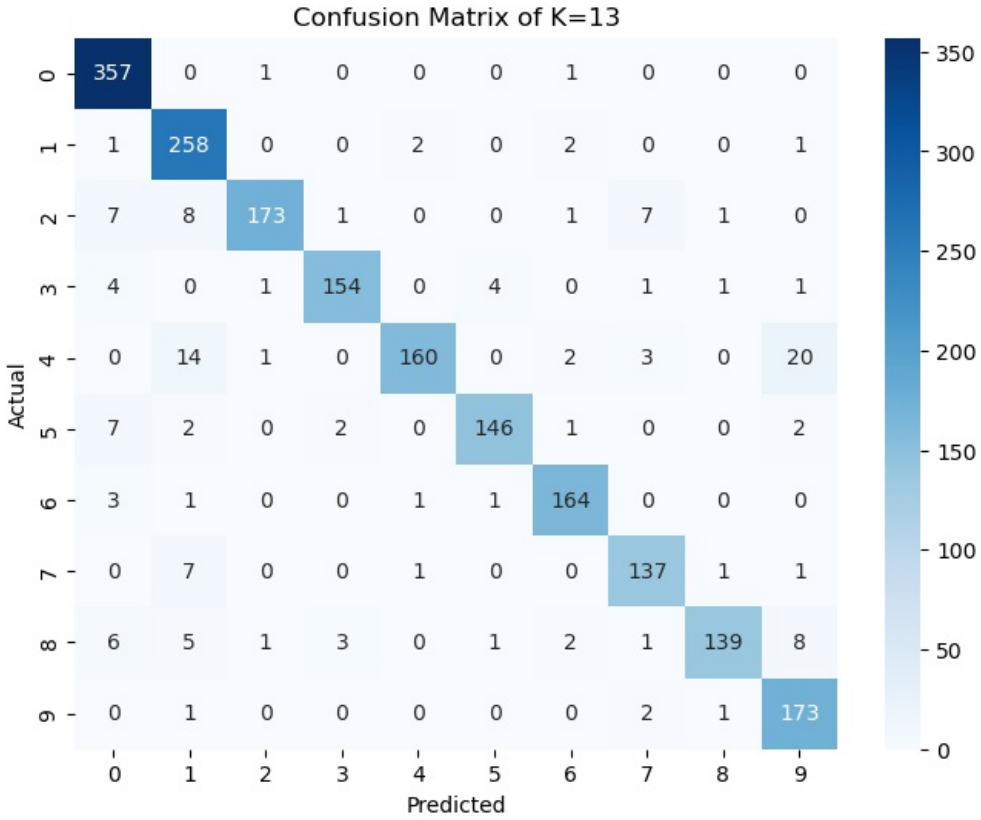

```
For K=7, Precision=(array([0.94164456, 0.90559441, 0.95767196, 0.9625    , 0.97093023,
       0.96710526, 0.95375723, 0.91946309, 0.97278912, 0.85148515]),), Recall=(array([0.98885794, 0.98106061, 0
.91414141, 0.92771084, 0.835     ,
       0.91875   , 0.97058824, 0.93197279, 0.86144578, 0.97175141]),)
Accuracy of K=9 is 0.9342301943198804
```



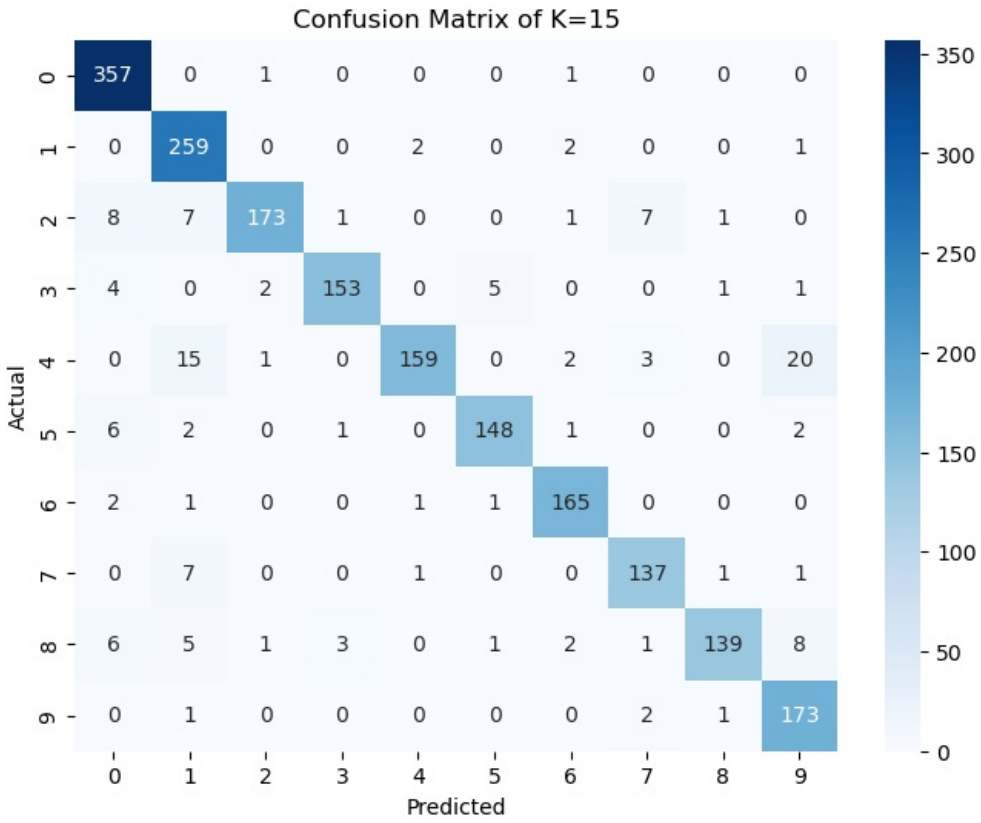

```
For K=9, Precision=(array([0.93175853, 0.89930556, 0.96703297, 0.97452229, 0.96531792,
       0.96103896, 0.95348837, 0.90728477, 0.97260274, 0.85714286]),), Recall=(array([0.98885794, 0.98106061, 0
.88888889, 0.92168675, 0.835     ,
       0.925     , 0.96470588, 0.93197279, 0.85542169, 0.98305085]),)
Accuracy of K=11 is 0.9337319382162431
```



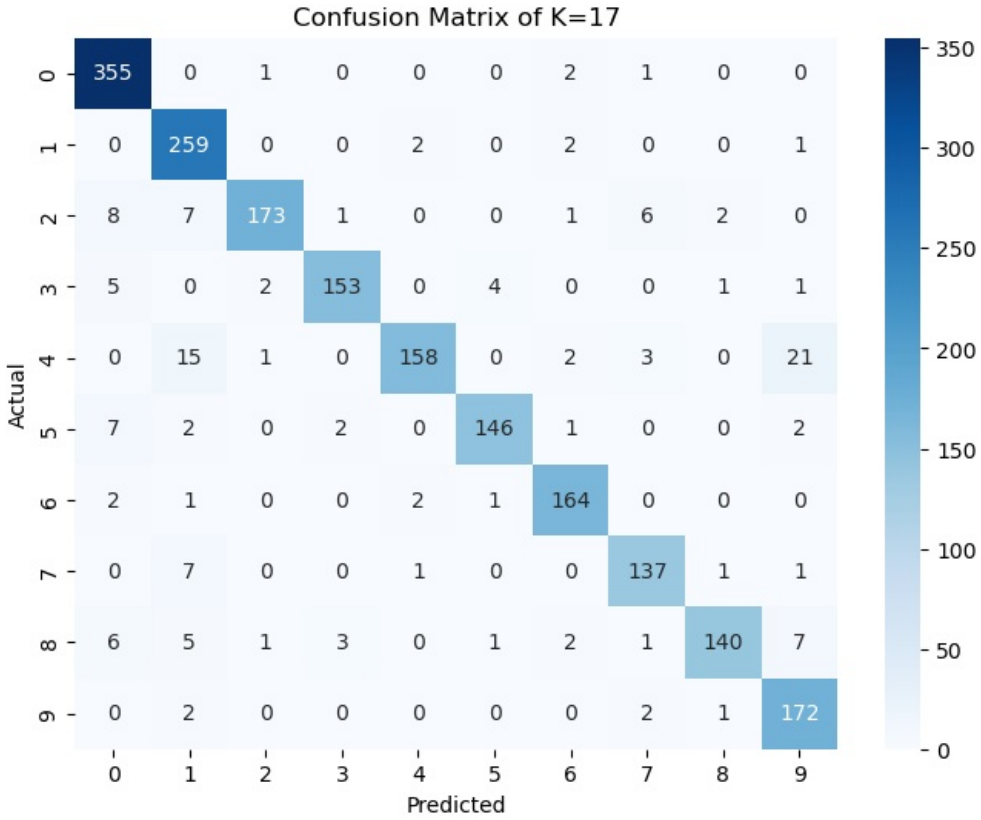

```
For K=11, Precision=(array([0.93211488, 0.88659794, 0.97802198, 0.96855346, 0.97619048,
       0.95454545, 0.95930233, 0.90728477, 0.97222222, 0.85714286]),), Recall=(array([0.99442897, 0.97727273, 0
.8989899 , 0.92771084, 0.82      ,
       0.91875   , 0.97058824, 0.93197279, 0.84337349, 0.98305085]),)
Accuracy of K=13 is 0.9272546088689586
```




```
For K=13, Precision=(array([0.92727273, 0.87162162, 0.97740113, 0.9625    , 0.97560976,
       0.96052632, 0.94797688, 0.90728477, 0.97202797, 0.83980583]),), Recall=(array([0.99442897, 0.97727273, 0
.87373737, 0.92771084, 0.8       ,
       0.9125    , 0.96470588, 0.93197279, 0.8373494 , 0.97740113]),)
Accuracy of K=15 is 0.9282511210762332
```




```
For K=15, Precision=(array([0.93211488, 0.87205387, 0.97191011, 0.96835443, 0.97546012,
       0.95483871, 0.94827586, 0.91333333, 0.97202797, 0.83980583]),), Recall=(array([0.99442897, 0.98106061, 0
.87373737, 0.92168675, 0.795     ,
       0.925     , 0.97058824, 0.93197279, 0.8373494 , 0.97740113]),)
Accuracy of K=17 is 0.9252615844544095
```




```
For K=17, Precision=(array([0.92689295, 0.86912752, 0.97191011, 0.96226415, 0.96932515,
       0.96052632, 0.94252874, 0.91333333, 0.96551724, 0.83902439]),), Recall=(array([0.98885794, 0.98106061, 0
.87373737, 0.92168675, 0.79       ,
       0.9125    , 0.96470588, 0.93197279, 0.84337349, 0.97175141]),)
```

In [ ]: