I will be working with digits dataset!

In [55]:

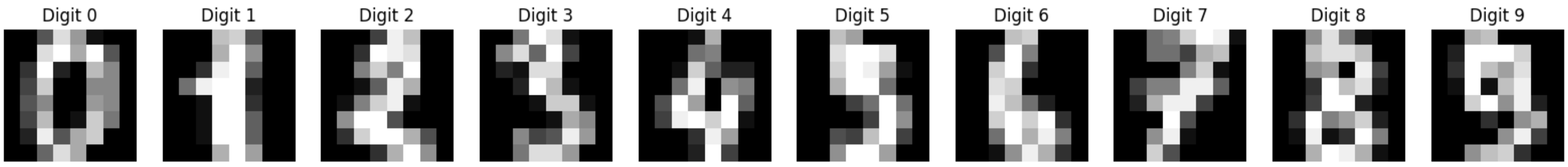
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

digits = load_digits()

num_digits_to_print = 10
fig, axes = plt.subplots(1, num_digits_to_print, figsize=(20, 4))

for i in range(num_digits_to_print):
    axes[i].imshow(digits.images[i], cmap='gray')
    axes[i].set_title(f"Digit {i}")
    axes[i].axis('off')

plt.show()
```



I will run digits data through well know K-nn algorithm to see what kind of accuracy full matrix gets.

In [45]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
X, y = load_digits(return_X_y=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the k-Nearest Neighbors classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train_scaled, y_train)

# Predict on the testing set
y_pred = knn_classifier.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.975
```

Full dataset got 97.5 % accuracy

There are just a few examples of feature selection methods that reduce feature matrix for classificaiton tasks

In [ ]:

```
X_new = SelectKBest(mutual_info_classif, k=16).fit_transform(X, y)
X_new = SelectKBest(f_classif, k=16).fit_transform(X, y)
X_new = SelectKBest(mutual_info_classif, k=16).fit_transform(X, y)
```

In [60]:

```
X_new.shape
```

Out[60]:

```
(1797, 16)
```

You can see above that matrix went from 64 dimensions to 16 dimensions

In [53]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectKBest, f_classif

# Load the digits dataset
X, y = load_digits(return_X_y=True)

# Perform feature selection using SelectKBest with F-test
```

```python
X_new = SelectKBest(mutual_info_classif, k=16).fit_transform(X, y)
print("New shape of X:", X_new.shape)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=6)

knn_classifier.fit(X_train_scaled, y_train)

y_pred = knn_classifier.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
New shape of X: (1797, 16)
Accuracy: 0.9555555555555556
```

We were able use the reduced 25% or 16/64 of features to get near original results.

Reducing dimensions helps reduce computation cost and time.