Module06_DigitRecognizer

February 26, 2025

1 Module 06 Digit Recognizer

Isha Singh

Professor Irene Tsapara

MSDS 422 Practical Machine Learning

26 February 2025

1.0.1 PART 01 Introduction

The following research is based on the Kaggle Competition: Digit Recognizer. The main goal of this study is to classify handwritten numbers (0-9) using machine learning models, specifically Random Forest classifiers and K-Means. Different model factors will be evaluated by modifying hyperparameters, applying PCA for feature selection, and refining the model. The study will analyze performance metrics and validation scores to determine which model provides the most accurate and precise results. This will be done by reviewing validation accuracy, generating confusion matrices, and evaluating clustering results to better understand the model's performance.

1.0.2 PART 02 Data load / Libraries load

Libraries load

[523]:
```
import warnings
warnings.filterwarnings("ignore")
```

[524]:
```
import tensorflow as tf
print(tf.config.list_physical_devices('GPU'))
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[525]:
```
import time
```

[526]:
```
import numpy as np
```

[527]:
```
import pandas as pd
```

[528]:
```
import matplotlib.pyplot as plt
```

[529]:
```
import seaborn as sns
```

[530]:
```
from sklearn.preprocessing import MinMaxScaler
```

[531]:
```
import tensorflow.keras as keras
```

[532]:
```
from sklearn.decomposition import PCA
```

[533]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[534]:
```
from sklearn.decomposition import PCA
```

[535]:
```
from sklearn.model_selection import train_test_split
```

[536]:
```
from tensorflow.keras.utils import to_categorical
```

[537]:
```
from sklearn.ensemble import RandomForestClassifier
```

[538]:
```
from sklearn.model_selection import cross_val_score, KFold
```

[539]:
```
from sklearn.cluster import KMeans
```

[540]:
```
from sklearn.metrics import silhouette_score
```

[541]:
```
from sklearn.model_selection import GridSearchCV
```

[542]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, roc_curve, auc
```

Data load

[543]:
```
train_digit_recognizer_dataframe = pd.read_csv("/Users/isingh/Desktop/
↪digit-recognizer/train.csv")
```

[544]:
```
test_digit_recognizer_dataframe = pd.read_csv("/Users/isingh/Desktop/
↪digit-recognizer/test.csv")
```

1.0.3 PART 03 Data Presentation

The dataset used has 42,000 training images and 28,000 test images, where each specific image is 28 x 28 pixels in grayscale with values that range from 0 to 255. The value 0 represents white, and 255 represents black. The training set includes a label column that indicates the digits 0-9; however, the test dataset incorporates only test set image data. Some of the observations that were noticed during the procedure were that no missing values were found within the dataset, each of the digits from 0-9 is explicitly there, and there were various different samples of handwriting styles.

[545]:
```
print("Training Data Shape:", train_digit_recognizer_dataframe.shape)
print("Test Data Shape:", train_digit_recognizer_dataframe.shape)
```

```
Training Data Shape: (42000, 785)
Test Data Shape: (42000, 785)
```

[546]:
```
train_digit_recognizer_dataframe.head()
```

[546]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	\
0	1	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

	pixel8	…	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	\
0	0	…	0	0	0	0	0	0	
1	0	…	0	0	0	0	0	0	
2	0	…	0	0	0	0	0	0	
3	0	…	0	0	0	0	0	0	
4	0	…	0	0	0	0	0	0	

	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

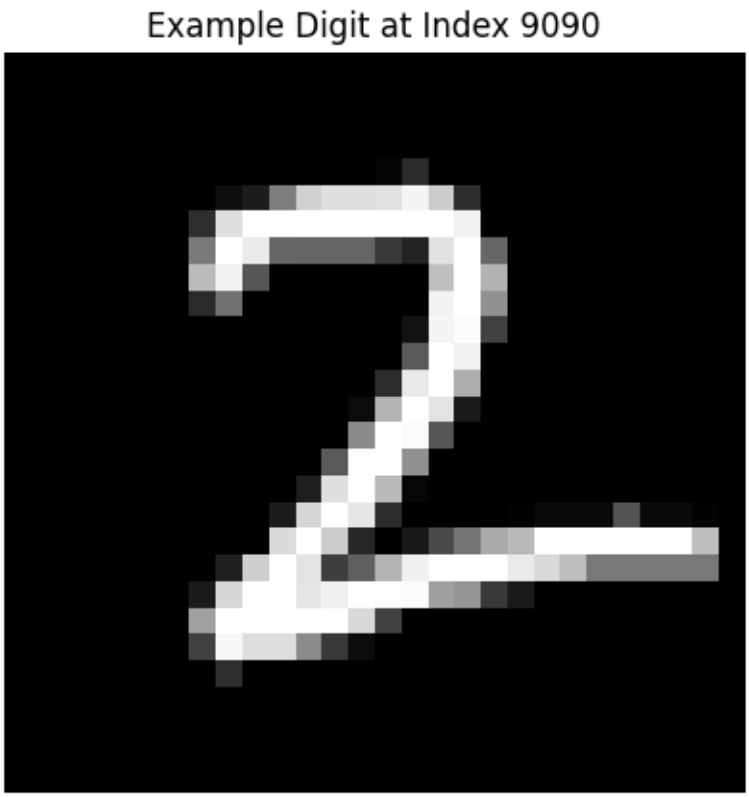
[5 rows x 785 columns]

[547]:
```
print(train_digit_recognizer_dataframe.isnull().any().sum())
print(train_digit_recognizer_dataframe.isnull().any().sum())
```

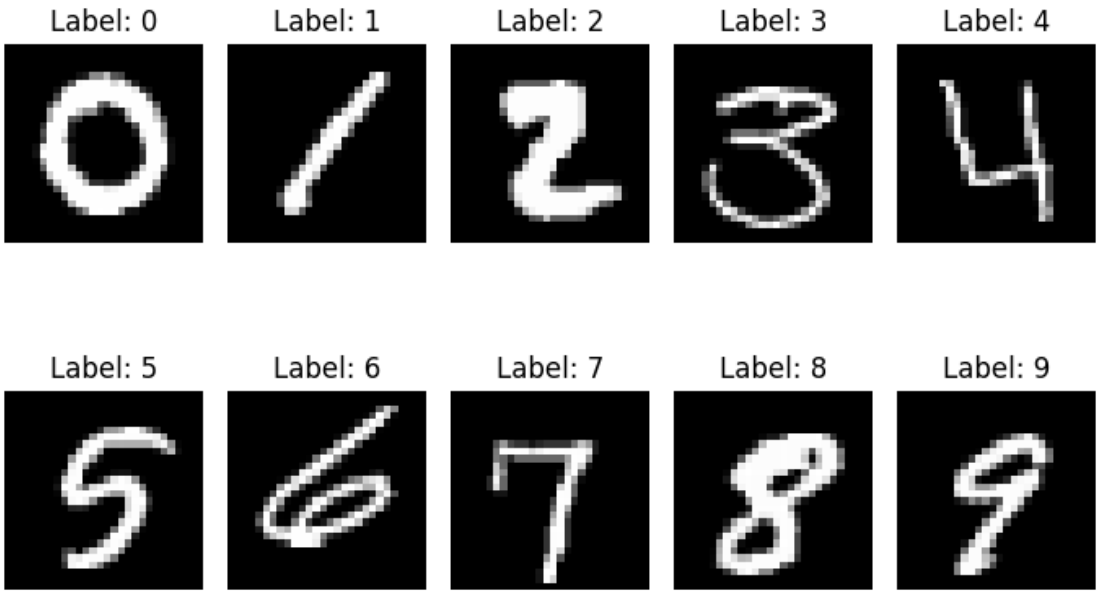
```
0
0
```

1.0.4 PART 04 EDA of DATASET

[548]:
```
random_index = 9090
image_data = train_digit_recognizer_dataframe.iloc[random_index, 1:].values.
 ↪reshape(28, 28)
plt.figure(figsize=(7, 5))
plt.imshow(image_data, cmap='gray')
plt.title(f"Example Digit at Index {random_index}")
plt.axis("off")
plt.show()
```

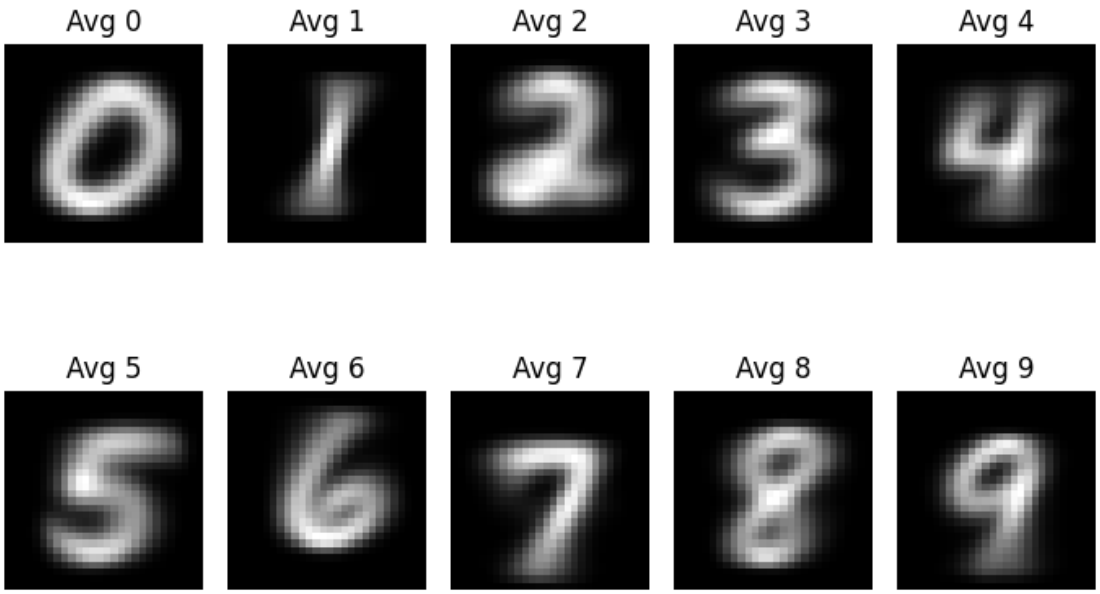


[549]:
```
fig, axes = plt.subplots(2, 5, figsize=(7, 5))
for digit in range(10):
    img =␣
 ↪train_digit_organizer_dataframe[train_digit_organizer_dataframe['label'] ==␣
 ↪digit].iloc[0, 1:].values.reshape(28, 28)
    ax = axes.flatten()[digit]
    ax.imshow(img, cmap='gray')
    ax.set_title(f"Label: {digit}")
    ax.axis("off")
plt.tight_layout()
plt.show()
```



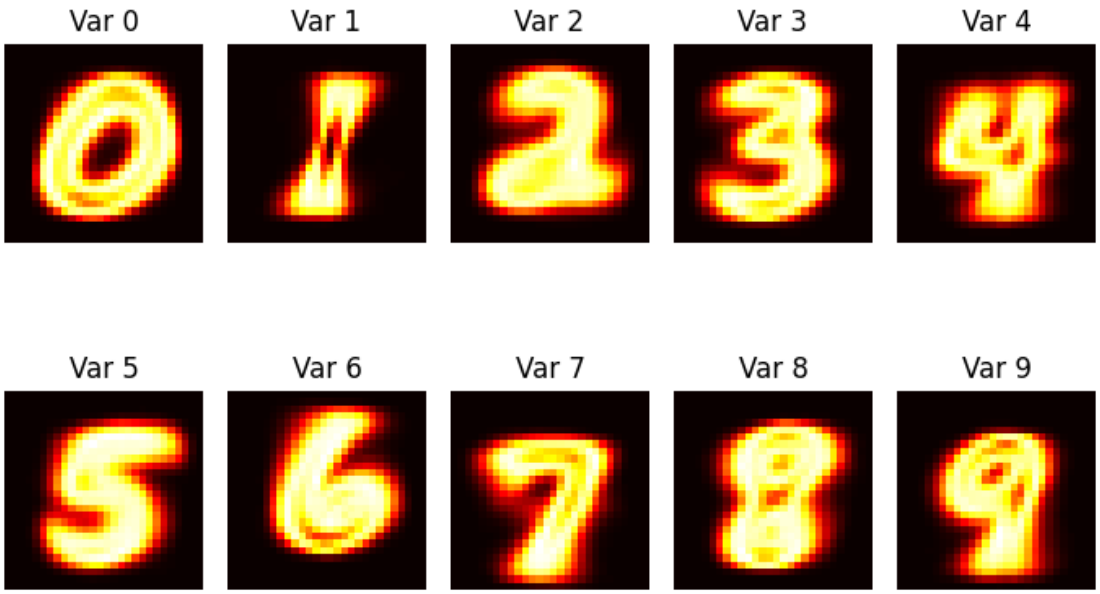
Average Image for Each Digit

[550]:
```
fig, axes = plt.subplots(2, 5, figsize=(7, 5))
for i, digit in enumerate(range(10)):
    avg_image =␣
 ↪train_digit_recognizer_dataframe[train_digit_recognizer_dataframe['label']␣
 ↪== digit].iloc[:, 1:].mean().values.reshape(28, 28)
    ax = axes.flat[i]
    ax.imshow(avg_image, cmap='gray')
    ax.set_title(f"Avg {digit}")
    ax.axis("off")
plt.tight_layout()
plt.show()
```



Pixel Variance For Digit

[551]:
```
fig, axes = plt.subplots(2, 5, figsize=(7, 5))
for digit in range(10):
    var_image =␣
 ↪train_digit_recognizer_dataframe[train_digit_recognizer_dataframe['label']␣
 ↪== digit].iloc[:, 1:].var().values.reshape(28, 28)
    ax = axes[digit // 5, digit % 5]
    ax.imshow(var_image, cmap='hot')
    ax.set_title(f"Var {digit}")
    ax.axis("off")
plt.tight_layout()
plt.show()
```



1.0.5 PART 05 Overview of findings and next steps

As of this part of the research, we can understand that the dataset has no missing values, and therefore it is ready to proceed to the next step. Before we focus on training and modeling, there are some required additional steps to take in order to make sure the model performs as expected. The pixel values will be scaled between 0 and 1 in order for the process to have a better modeling procedure and improve training. As we know, the dataset is well-structured, so it is very unlikely to have outliers and pixel values that are considered to be unusual, and we can also assume that imputation will not be required as there were no missing values indicated within the training dataset.

Since the labels are already numerical, one-hot encoding will not be applied, as it is not required for the models being used. Label encoding is also unnecessary because the labels are already in a suitable format.

Binning will not be required since the dataset contains continuous pixel intensity values rather than discrete categories. Previously, EDA was conducted to analyze pixel value patterns and variations, ensuring all digits (0–9) are properly represented. An outlier analysis will also be performed to detect any unusual pixel intensities that could impact training.

To simplify the data while keeping the most important features, Principal Component Analysis (PCA) will be used to reduce the number of features. The next stages will involve the dataset to be divided into three parts: testing, training, and validation. This will help evaluate how well the model performs. Afterwards, hyperparameter tuning will be used in order to optimize the models that were created (Random

Forest-related and KMeans-related). The adjustments in the hyperparamters will help create an improvement in the model's accuracy in being able to recongize digits that are handwritten.

1.0.6 PART 06 Cleansing and Preprocessing

[552]:
```
print("The number of missing values is", train_digit_recognizer_dataframe.
  ↪isnull().any().sum())
```

```
The number of missing values is 0
```

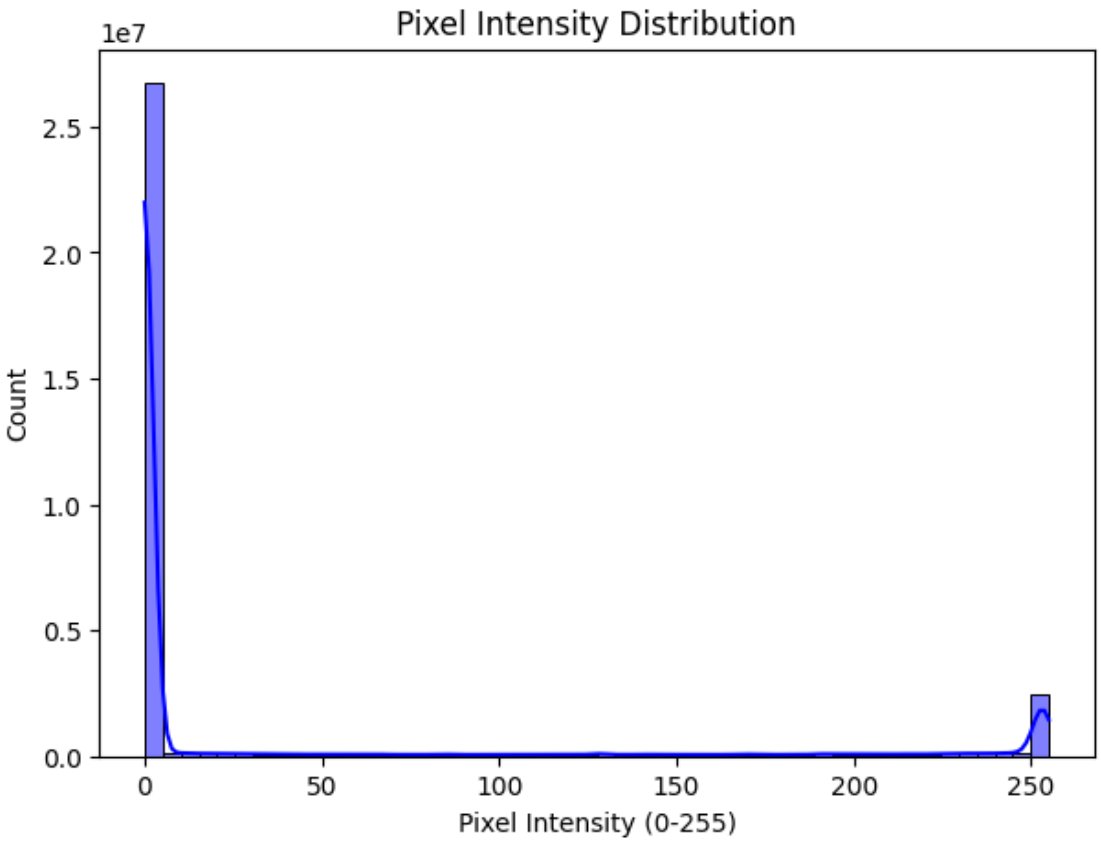
Outliers

[553]:
```
pixel_values = train_digit_recognizer_dataframe.drop(columns=['label']).values.
  ↪flatten()
```

[554]:
```
print("Pixel Value Statistics:")
print("Min:", np.min(pixel_values))
print("Max:", np.max(pixel_values))
print("Mean:", round(np.mean(pixel_values), 2))
print("Std Dev:", round(np.std(pixel_values), 2))
```

```
Pixel Value Statistics:
Min: 0
Max: 255
Mean: 33.41
Std Dev: 78.68
```

[555]:
```
plt.figure(figsize=(7, 5))
sns.histplot(pixel_values, bins=50, kde=True, color='blue')
plt.xlabel("Pixel Intensity (0-255)")
plt.ylabel("Count")
plt.title("Pixel Intensity Distribution")
plt.show()
```



1.0.7 PART 07 FEATURE ENGINEERING

Extracting features and labels

X is the Pixel values and Y is the labels

[556]:
```
X = train_digit_recognizer_dataframe.drop(columns=['label']).values
y = train_digit_recognizer_dataframe['label'].values
```

[557]:
```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

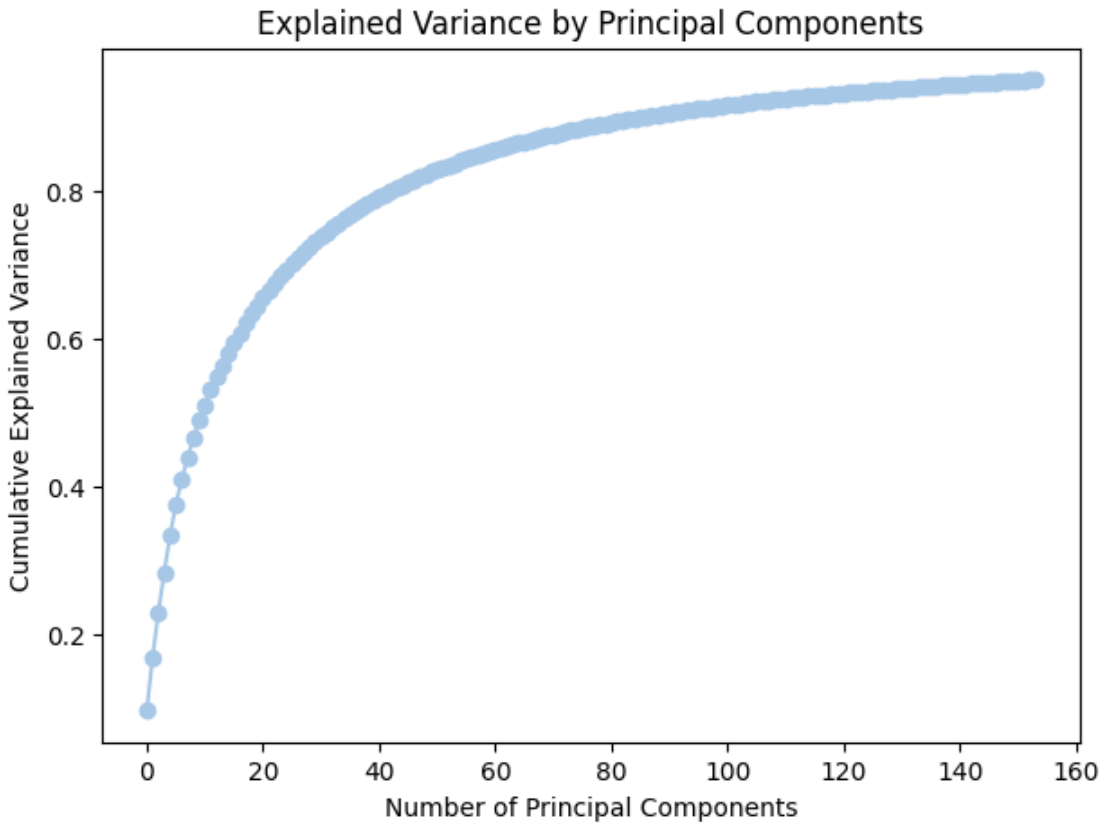
[558]:
```
start_time_pca = time.time()

pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

end_time_pca = time.time()

print(f"Time taken for the PCA to compute: {end_time_pca - start_time_pca:.2f}␣
 ↪seconds")
```

```
Time taken for the PCA to compute: 2.32 seconds
```

[559]:
```
num_components_retained = X_pca.shape[1]
print(f"Number of principal components retained: {num_components_retained}")
```

```
Number of principal components retained: 154
```

[560]:
```
plt.figure(figsize=(7, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', color='#A7C7E7')
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained Variance by Principal Components")
plt.show()
```



[561]:
```
explained_variance = pca.explained_variance_ratio_
print(" Variance Contribution of the First 10 Components")
for i in range(10):
    print(f"Component {i+1}: {explained_variance[i]:.4f}")
```

```
 Variance Contribution of the First 10 Components
Component 1: 0.0975
Component 2: 0.0716
```

```
Component 3: 0.0615
Component 4: 0.0538
Component 5: 0.0489
Component 6: 0.0430
Component 7: 0.0328
Component 8: 0.0289
Component 9: 0.0277
Component 10: 0.0235
```

[562]:
```
print("Dataset shape after PCA:", X_pca.shape)
```

```
Dataset shape after PCA: (42000, 154)
```

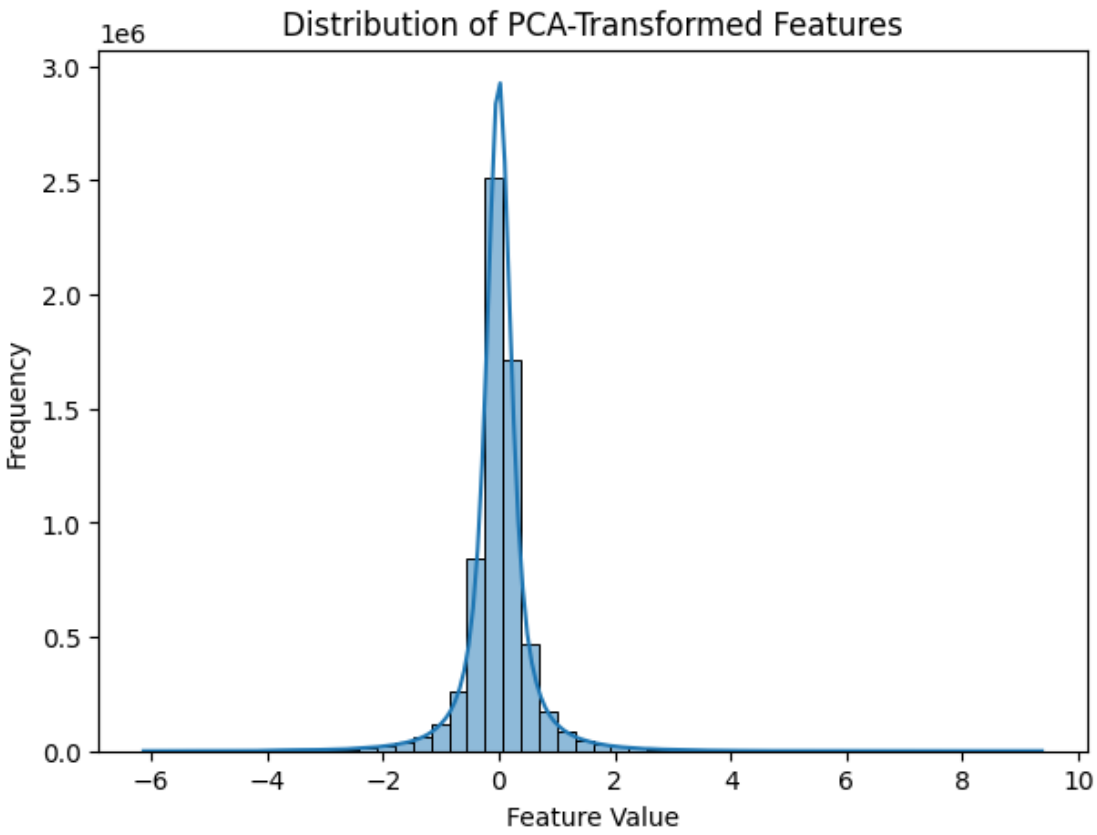
[563]:
```
augment_data = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)
```

1.0.8 PART 08 Overview presentation of the cleaned dataset

At this point in research, the data has been cleaned up and is set to proceed to the next step: training. So far, the labels have been processed correctly to be able to confirm if they are in the correct format so the models will understand. Afterwards, Principal component analysis (PCA) was implemented to reduce the number of features but however keeping the important and relevant data. With the PCA, it was decided to keep 95 percent of the important features. Data augmentation was done with techniques like rotation, shifting, and zooming, which will help by making the model more familiar with different variations of digits and improving its ability to recognize them in different forms. In the end, a confirmation was done to check if all digit classes were represented correctly and that the dataset was still organized.

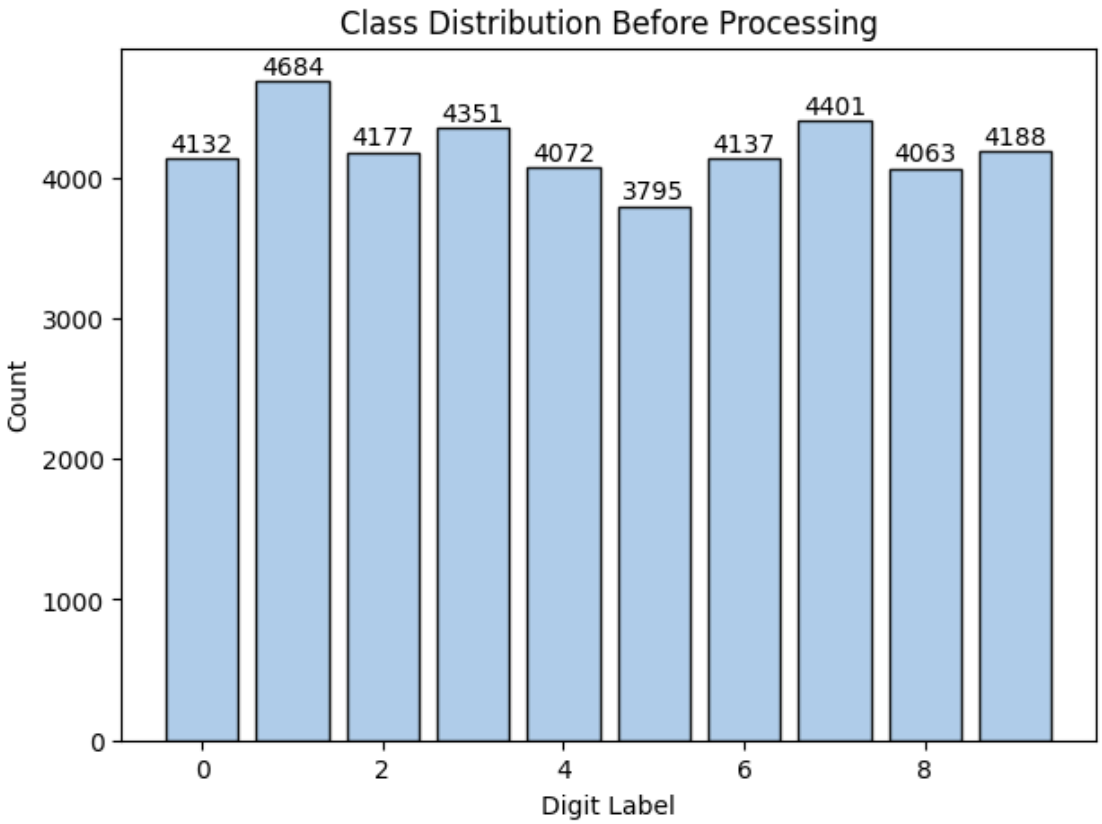
1.0.9 PART 09 Final EDA and comparisons

[564]:
```
plt.figure(figsize=(7, 5))
sns.histplot(X_pca.flatten(), bins=50, kde=True)
plt.title("Distribution of PCA-Transformed Features")
plt.xlabel("Feature Value")
plt.ylabel("Frequency")
plt.show()
```



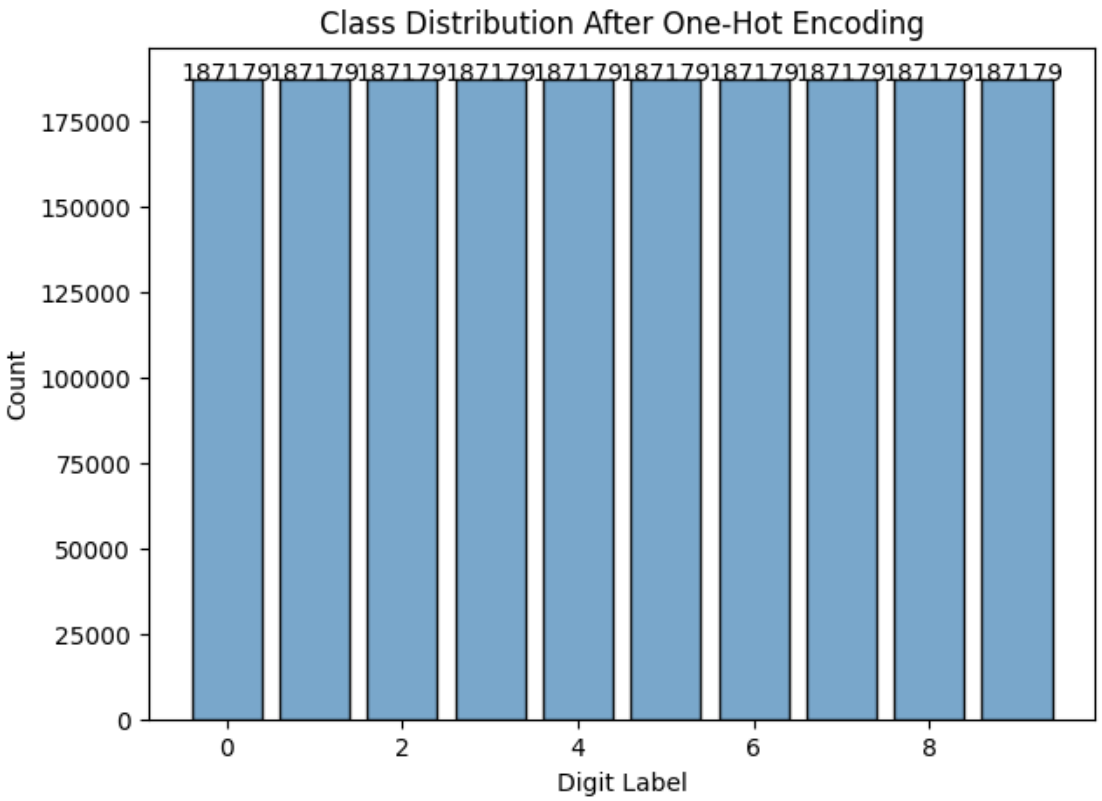
Before Processing

[565]:
```
label_counts = train_digit_organizer_dataframe['label'].value_counts().
 ↪sort_index()

plt.figure(figsize=(7, 5))
bars = plt.bar(label_counts.index, label_counts.values, color='#A7C7E7',␣
 ↪edgecolor='black', alpha=0.9)  # Light blue

for bar in bars:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 50, int(bar.
 ↪get_height()), ha='center', fontsize=10)

plt.title("Class Distribution Before Processing")
plt.xlabel("Digit Label")
plt.ylabel("Count")

plt.show()
```

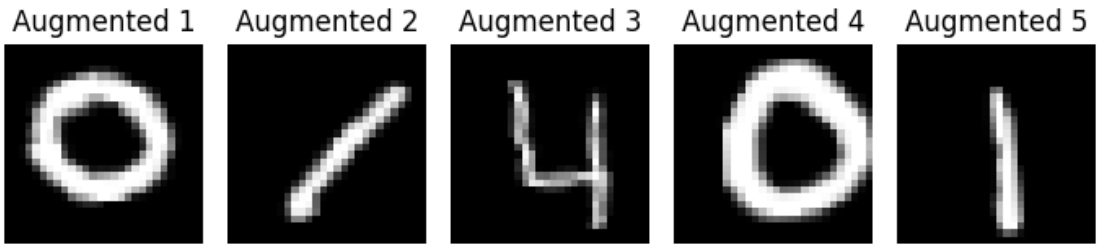


After processing

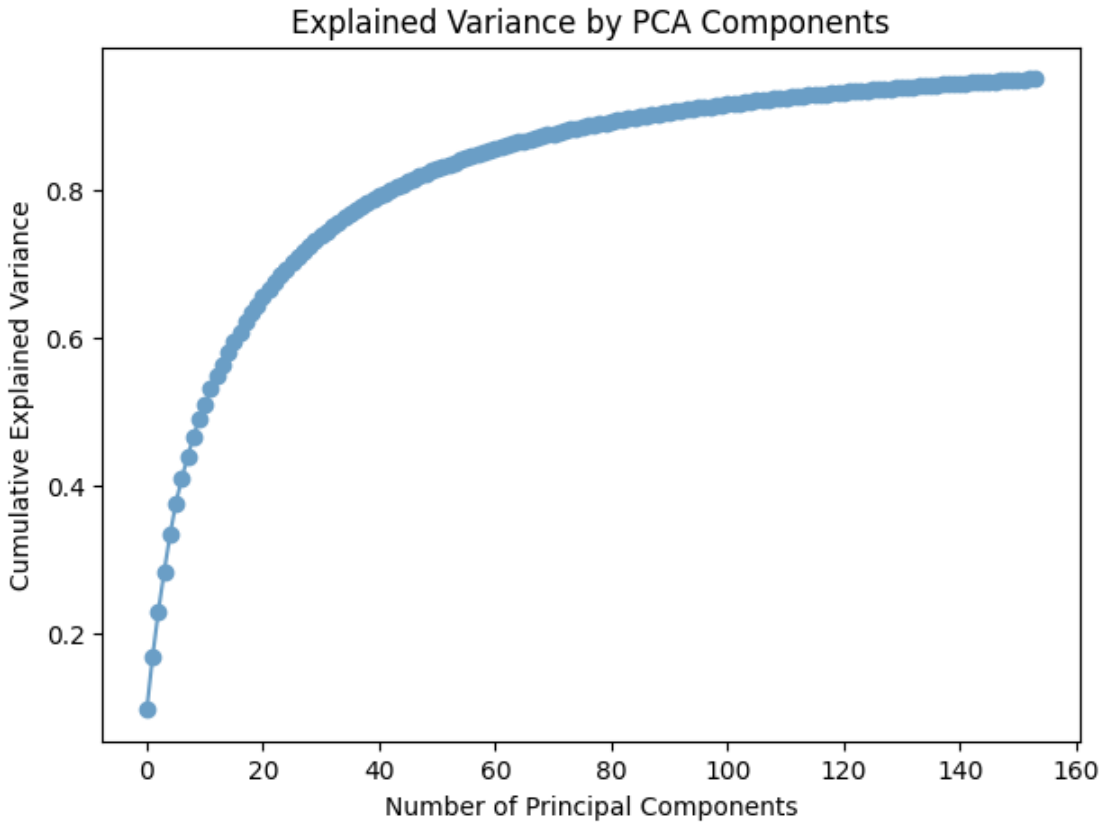
[566]:
```
one_hot_sums = np.sum(y, axis=0)

plt.figure(figsize=(7, 5))
bars = plt.bar(np.arange(10), one_hot_sums, color='#6A9EC6', edgecolor='black',␣
 ↪alpha=0.9)

for bar in bars:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 50, int(bar.
 ↪get_height()), ha='center', fontsize=10)

plt.title("Class Distribution After One-Hot Encoding")
plt.xlabel("Digit Label")
plt.ylabel("Count")

plt.show()
```



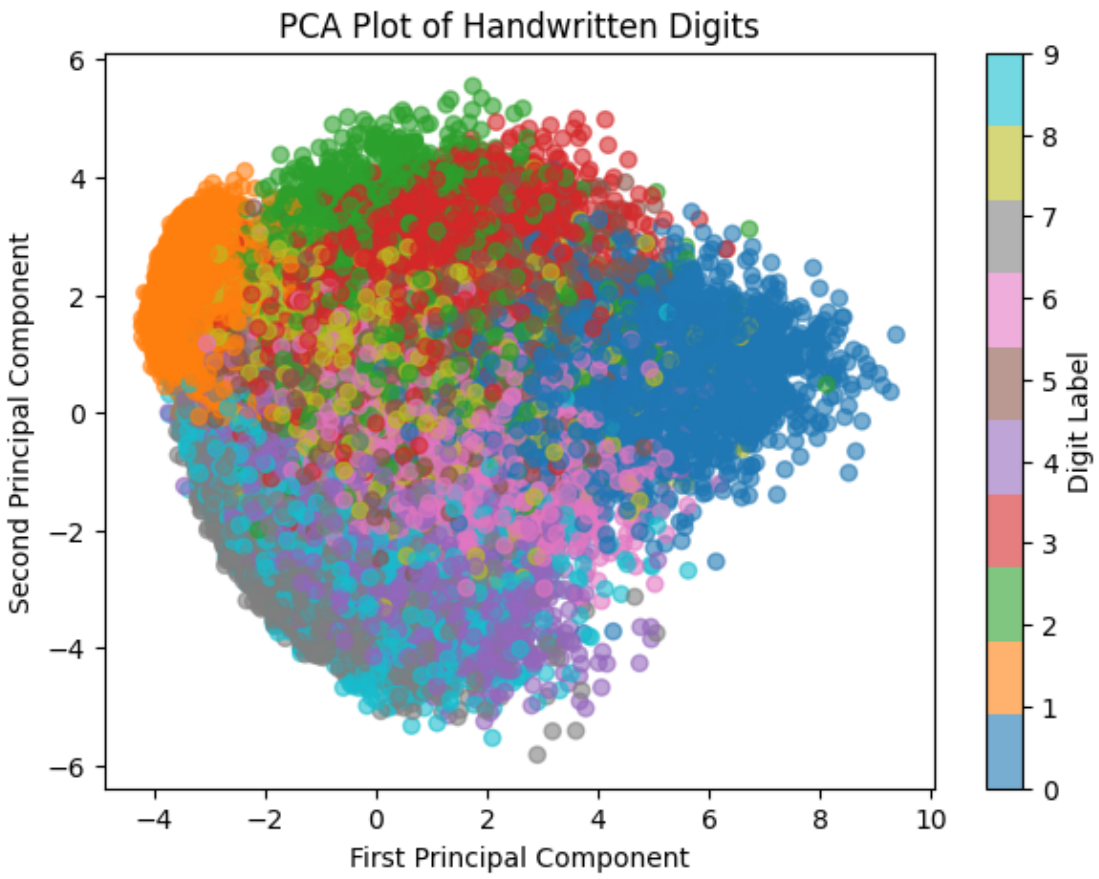
[567]:
```
sample_images = X[:5].reshape(5, 28, 28, 1)

augmented_images = next(augment_data.flow(sample_images, batch_size=5))

fig, axes = plt.subplots(1, 5, figsize=(7, 5))
for i, ax in enumerate(axes):
    ax.imshow(augmented_images[i].reshape(28, 28), cmap="gray")
    ax.set_title(f"Augmented {i+1}")
    ax.axis("off")

plt.tight_layout()
plt.show()
```



[568]:
```
plt.figure(figsize=(7, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', color="#6A9EC6")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained Variance by PCA Components")
plt.show()
```



[569]:
```
plt.figure(figsize=(7, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1],␣
 ↪c=train_digit_organizer_dataframe['label'], cmap="tab10", alpha=0.6)
plt.colorbar(label="Digit Label")
plt.xlabel("First Principal Component")
plt.ylabel("Second Principal Component")
plt.title("PCA Plot of Handwritten Digits")
plt.show()
```



1.0.10 PART 10 Data preprocessing specific to the model

[570]:
```
print("\nData preprocessing specific to the model:")
print(f"\nShape of original features: {X.shape}")

print(f"\nModel 01 Using original features for Random Forest: {X.shape}")

print(f"\nModel 02 Using PCA-transformed features for Random Forest: {X_pca.
  ↪shape}")

print(f"\nModel 03  Using standardized features for K-Means: {X_scaled.shape}")

print(f"\nFor Model 04 will be revisiting preprocessing with a proper training␣
  ↪and test setup.")
```

```
Data preprocessing specific to the model:

Shape of original features: (42000, 784)
```

Model 01 Using original features for Random Forest: (42000, 784)

Model 02 Using PCA-transformed features for Random Forest: (42000, 154)

Model 03 Using standardized features for K-Means: (42000, 784)

For Model 04 will be revisiting preprocessing with a proper training and test setup.

1.0.11 PART 11 Splitting

Random Forest Model 01

[571]:
```
X_train_rf, X_val_rf, y_train_rf, y_val_rf = train_test_split(X, y, test_size=0.
↪1, random_state=42, stratify=y)

X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_train_rf,␣
↪y_train_rf, test_size=0.11, random_state=42, stratify=y_train_rf)
```

PCA based Random Forest Model 02

[572]:
```
X_train_pca, X_val_pca, y_train_pca, y_val_pca = train_test_split(X_pca, y,␣
↪test_size=0.1, random_state=42, stratify=y)

X_train_pca, X_test_pca, y_train_pca, y_test_pca =␣
↪train_test_split(X_train_pca, y_train_pca, test_size=0.11, random_state=42,␣
↪stratify=y_train_pca)
```

KMeans Model 03 - no need to split as KMeans is unsupervised

[573]:
```
X_kmeans = X_scaled.copy()
```

Fixed Experiment Model 04

[574]:
```
X_train_fixed, X_test_fixed, y_train_fixed, y_test_fixed = train_test_split(X,␣
↪y, test_size=0.2, random_state=42, stratify=y)
```

Summary of All Splitting

[575]:
```
print("\nData Splitting Summary of Shapes:")
print(f"\nRandom Forest - Train: {X_train_rf.shape}, Val: {X_val_rf.shape},␣
↪Test: {X_test_rf.shape}")
print(f"\nPCA Random Forest - Train: {X_train_pca.shape}, Val: {X_val_pca.
↪shape}, Test: {X_test_pca.shape}")
print(f"\nK-Means ( no splitting required as unsupervised learning): {X_kmeans.
↪shape}")
print(f"\nFixed Model - Train: {X_train_fixed.shape} Test: {X_test_fixed.
↪shape}")
```

```
Data Splitting Summary of Shapes:

Random Forest - Train: (33642, 784), Val: (4200, 784), Test: (4158, 784)

PCA Random Forest - Train: (33642, 154), Val: (4200, 154), Test: (4158, 154)

K-Means ( no splitting required as unsupervised learning): (42000, 784)

Fixed Model - Train: (33600, 784) Test: (8400, 784)
```

1.0.12 PART 12 Overview of the steps to be completed and the rationale

The dataset has been split into training, testing, and validation sets, the features have been scaled, and PCA has been used to reduce dimensions. Now the three important parts are done, the models will be trained using K-Means, Random Forest with PCA, and Random Forest with the full dataset. After that, the Random Forest model will be fine-tuned by adjusting the hyperparameters, increasing the number of trees and depth, and finding the best K-value for K-Means. Once the training is complete, we will evaluate the models using important factors such as accuracy, confusion matrices, and clustering scores. With all the information obtained, we will compare the results to understand whether PCA improves efficiency and how well K-Means groups the digits. Based on these findings, we will then correct Model 4 to ensure proper implementation.

1.0.13 PART 13 Model training cross-validation.

[576]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

Model 01

[577]:
```
model_01 = RandomForestClassifier(n_estimators=100, # Number of trees in the␣
 ↪forest
 max_depth=None, # Maximum depth of the tree
 min_samples_split=2, # Minimum number of samples required to split an internal␣
 ↪node
 min_samples_leaf=1, # Minimum number of samples required to be at a leaf node
 random_state=42) # Random state for reproducibility
```

[578]:
```
accuracy_model_01 = cross_val_score(model_01, X_train_rf, y_train_rf, cv=kf,␣
 ↪scoring='accuracy')
```

[579]:
```
print(f"The Average Accuracy for Model 01: {np.mean(accuracy_model_01):.5f}")
```

```
The Average Accuracy for Model 01: 0.96168
```

Model 02

[580]:
```
model_02 = RandomForestClassifier(
    n_estimators=100,       # Number of trees in the forest
    max_depth=None,         # Maximum depth of the tree (None means unlimited␣
 ↪depth)
    min_samples_split=2,    # Minimum number of samples required to split an␣
 ↪internal node
    min_samples_leaf=1,     # Minimum number of samples required to be at a leaf␣
 ↪node
    random_state=42         # Random state for reproducibility
)
```

[581]:
```
accuracy_model_02 = cross_val_score(model_02, X_train_pca, y_train_pca, cv=kf,␣
 ↪scoring='accuracy')
```

[582]:
```
print(f"The Average Accuracy for Model 02: {np.mean(accuracy_model_02):.5f}")
```

```
The Average Accuracy for Model 02: 0.93832
```

Model 03 is included in the Hyperparameter Tuning section, while Model 04 will be implemented towards the end

1.0.14 PART 14 Model Hypertuning

Model 01

[584]:
```
param_grid_rf = {
    'n_estimators': [50, 100, 150],  # Number of trees in the forest
    'max_depth': [5, 15, None],  # Maximum depth of the tree,  None means␣
 ↪unlimited depth
    'min_samples_split': [2, 4, 8],# Minimum number of samples required to␣
 ↪split an internal node
    'min_samples_leaf': [1, 3, 5]  # Minimum number of samples required to be␣
 ↪at a leaf node
}
```

[585]:
```
rf_model = RandomForestClassifier(random_state=42)
```

[586]:
```
start_time_grid_rf = time.time()

grid_rf = GridSearchCV(rf_model, param_grid_rf, cv=3, scoring='accuracy',␣
 ↪n_jobs=-1)
grid_rf.fit(X_train_rf, y_train_rf)

end_time_grid_rf = time.time()
```

[319]:
```
print("Best parameters for Model 01:", grid_rf.best_params_)
print(f"Time taken for Model 01 tuning: {end_time_grid_rf - start_time_grid_rf:.
 ↪2f} seconds")
```

```
Best parameters for Model 01: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 150}
Time taken for Model 01 tuning: 241.28 seconds
```

Model 02

[587]:
```
param_grid_rf_pca = {
    'n_estimators': [50, 100, 150],  # Number of trees in the forest
    'max_depth': [5, 15, None],  # Maximum depth of the tree, none means␣
 ↪unlimited depth
    'min_samples_split': [2, 4, 8],  #  Minimum number of samples required to␣
 ↪split an internal node
    'min_samples_leaf': [1, 3, 5]  # Minimum number of samples required to be␣
 ↪at a leaf node
}
```

[588]:
```
rf_model_pca = RandomForestClassifier(random_state=42)
```

[589]:
```
start_time_grid_pca = time.time()

grid_rf_pca = GridSearchCV(rf_model_pca, param_grid_rf_pca, cv=3,␣
 ↪scoring='accuracy', n_jobs=-1)
grid_rf_pca.fit(X_train_pca, y_train_pca)

end_time_grid_pca = time.time()
```

[590]:
```
print("Best parameters for Model 02:", grid_rf_pca.best_params_)
print(f"Time taken for Grid Search (Model 02): {end_time_grid_pca -␣
 ↪start_time_grid_pca:.2f} seconds")
```

```
Best parameters for Model 02: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 4, 'n_estimators': 150}
Time taken for Grid Search (Model 02): 581.89 seconds
```

Model 03

[591]:
```
np.random.seed(42)
k_values = np.random.choice(range(3, 12), size=5, replace=False)
```

[592]:
```
print(f"Best K for Model 03: {best_k_silhouette_score}, Silhouette Score:␣
 ↪{max(silhouette_scores):.5f}")
print(f"Time taken for K-Means clustering: {end_time_kmeans - start_time_kmeans:
 ↪.2f} seconds")
```

```
Best K for Model 03: 4, Silhouette Score: 0.09107
Time taken for K-Means clustering: 144.28 seconds
```

Model 4 will be implemented after identifying the best hyperparameters, serving as the final version ready for submission.

1.0.15 PART 15 Model Testing and PART 16 Model evaluation

Model 01

[594]:
```
y_pred_rf = grid_rf.best_estimator_.predict(X_val_rf)
```

Accuracy Score of Model 01

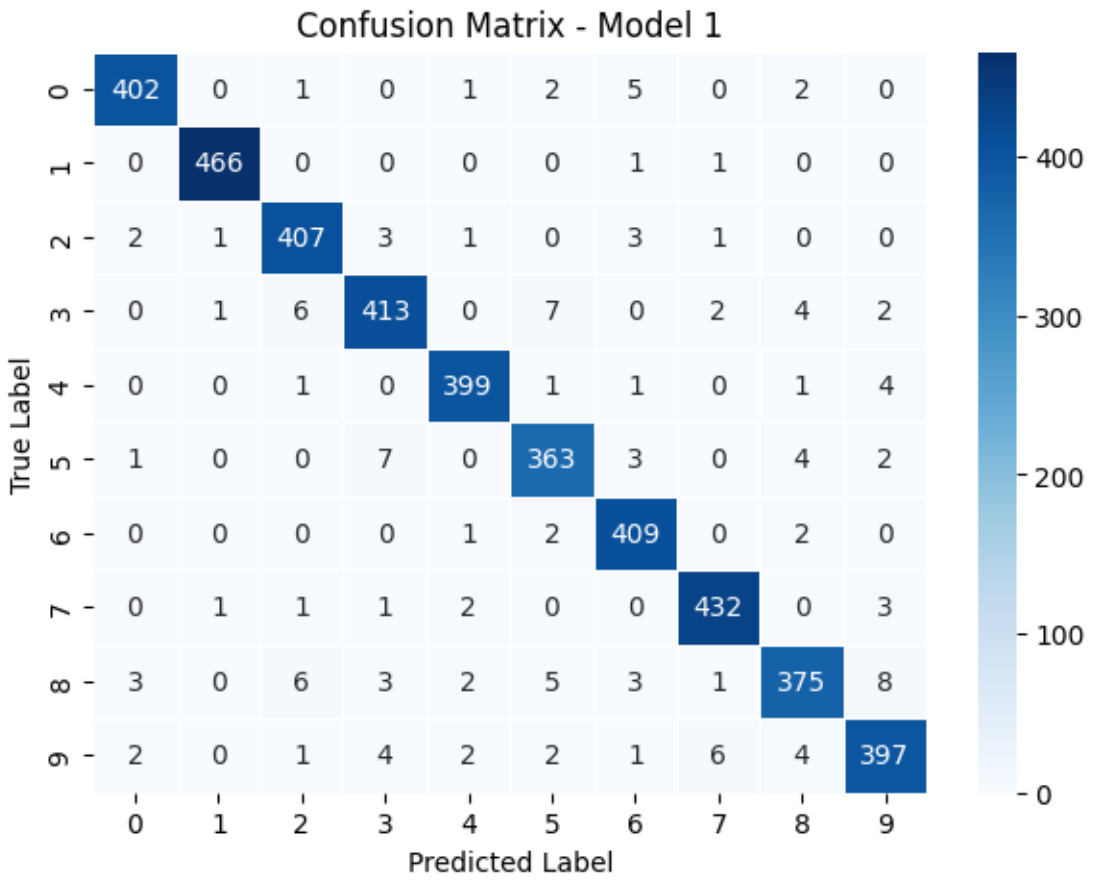
[595]:
```
accuracy_score_rf = accuracy_score(y_val_rf, y_pred_rf)
print(f"The Accuracy Score for random forest which used the full dataset is:␣
↪{accuracy_score_rf:.5f}")
```

```
The Accuracy Score for random forest which used the full dataset is: 0.96738
```

Confusion Matrix of Model 01

[596]:
```
confusion_matrix_rf = confusion_matrix(y_val_rf, y_pred_rf)
```

[597]:
```
plt.figure(figsize=(7, 5))
sns.heatmap(confusion_matrix_rf, annot=True, fmt="d", cmap="Blues",␣
↪linewidths=0.5)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Model 1")
plt.show()
```



Classification Report of Model 01

[598]:
```
print("                Classification Report for Model 1:")
print(classification_report(y_val_rf, y_pred_rf))
```

```
                Classification Report for Model 1:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       413
           1       0.99      1.00      0.99       468
           2       0.96      0.97      0.97       418
           3       0.96      0.95      0.95       435
           4       0.98      0.98      0.98       407
           5       0.95      0.96      0.95       380
           6       0.96      0.99      0.97       414
           7       0.98      0.98      0.98       440
           8       0.96      0.92      0.94       406
           9       0.95      0.95      0.95       419

    accuracy                           0.97      4200
```

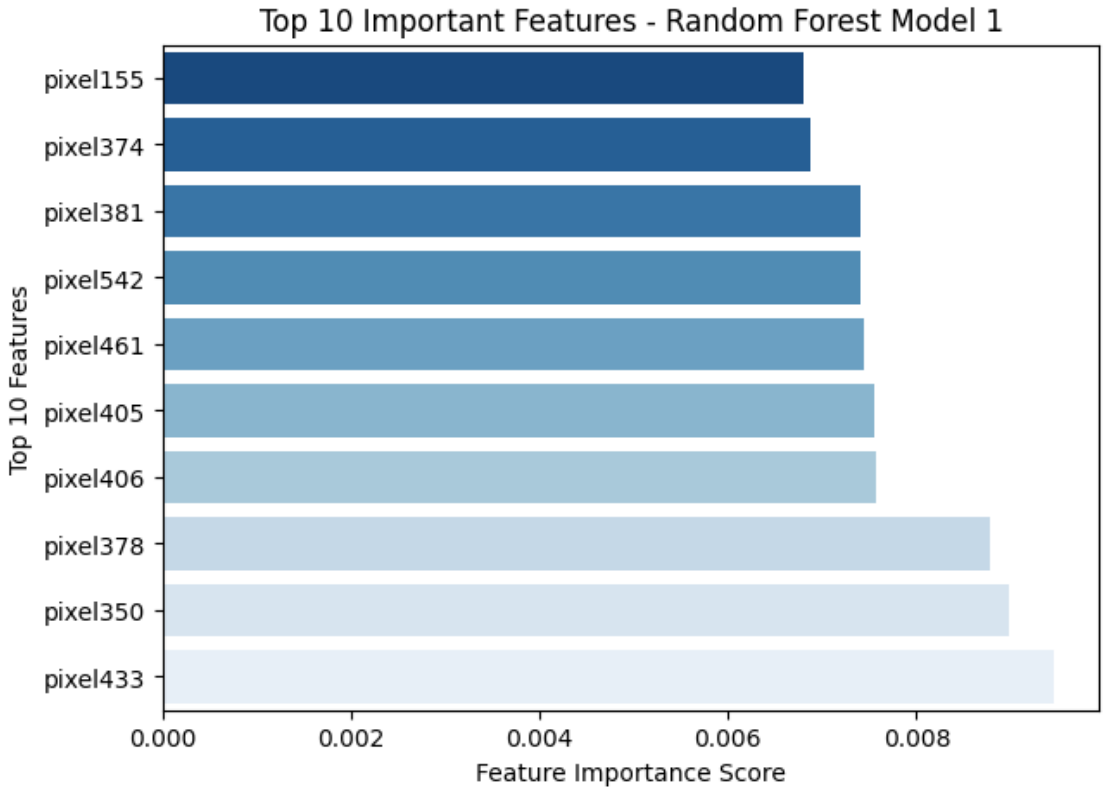
```
   macro avg       0.97      0.97      0.97      4200
weighted avg       0.97      0.97      0.97      4200
```

Top 10 Features of Model 01

[599]:
```
feature_importances = grid_rf.best_estimator_.feature_importances_
feature_names = train_digit_recognizer_dataframe.drop(columns=['label']).columns
```

[600]:
```
top_n = 10
top_indices = np.argsort(feature_importances)[-top_n:]
top_features, top_importance_values = feature_names[top_indices],␣
 ↪feature_importances[top_indices]
```

[601]:
```
plt.figure(figsize=(7, 5))
sns.barplot(x=top_importance_values, y=top_features, palette="Blues_r")
plt.xlabel("Feature Importance Score")
plt.ylabel("Top 10 Features")
plt.title("Top 10 Important Features - Random Forest Model 1")
plt.show()
```



Model 02

[602]:
```
y_pred_rf_pca = grid_rf_pca.best_estimator_.predict(X_val_pca)
```

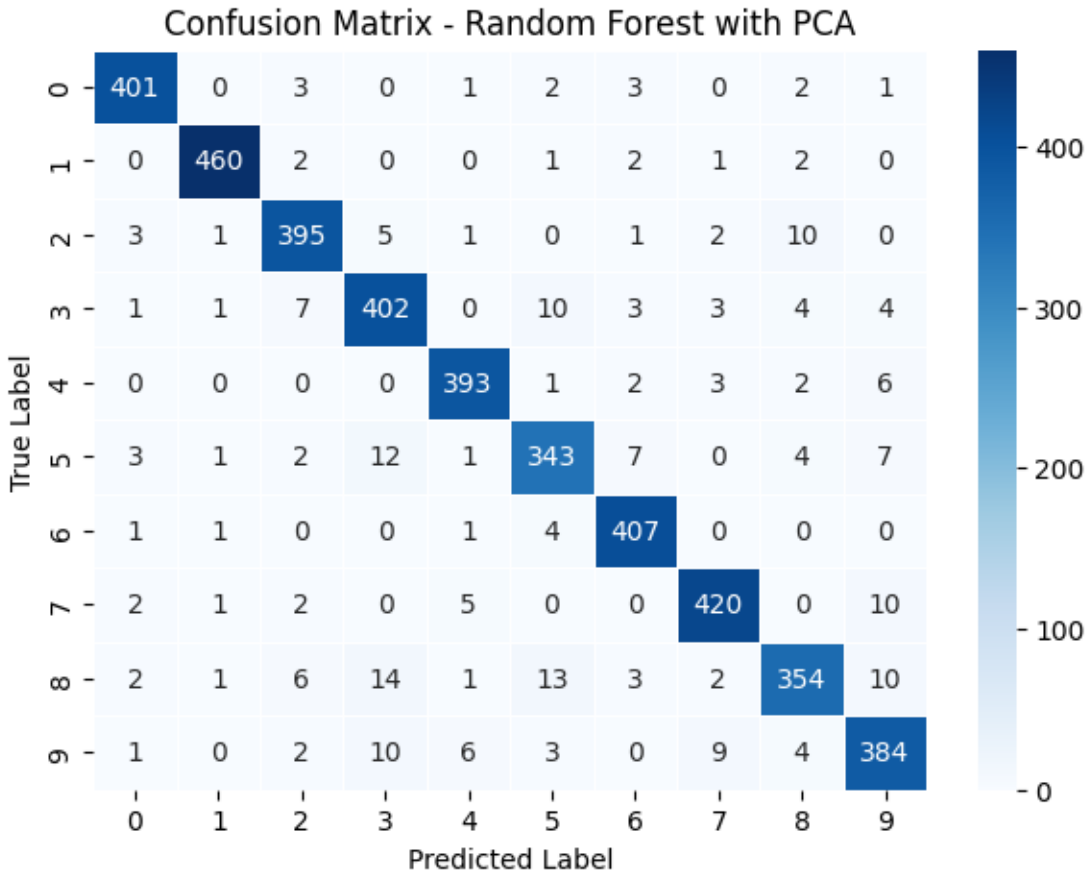
[603]:
```
accuracy_score_rf_pca = accuracy_score(y_val_pca, y_pred_rf_pca)
print(f"The Accuracy Score for Random Forest with PCA is:␣
 ↪{accuracy_score_rf_pca:.5f}")
```

```
The Accuracy Score for Random Forest with PCA is: 0.94262
```

Confusion Matrix of Model 02

[604]:
```
conf_matrix_rf_pca = confusion_matrix(y_val_pca, y_pred_rf_pca)
```

[605]:
```
plt.figure(figsize=(7, 5))
sns.heatmap(conf_matrix_rf_pca, annot=True, fmt='d', cmap="Blues", linewidths=0.
 ↪5)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Random Forest with PCA")
plt.show()
```



Classification Report of Model 02

[606]:
```
print("             Classification Report - Random Forest with PCA:\n")
print(classification_report(y_val_pca, y_pred_rf_pca))
```

```
             Classification Report - Random Forest with PCA:

              precision    recall  f1-score   support

           0       0.97      0.97      0.97       413
           1       0.99      0.98      0.99       468
           2       0.94      0.94      0.94       418
           3       0.91      0.92      0.92       435
           4       0.96      0.97      0.96       407
           5       0.91      0.90      0.91       380
           6       0.95      0.98      0.97       414
           7       0.95      0.95      0.95       440
           8       0.93      0.87      0.90       406
           9       0.91      0.92      0.91       419

    accuracy                           0.94      4200
   macro avg       0.94      0.94      0.94      4200
weighted avg       0.94      0.94      0.94      4200
```

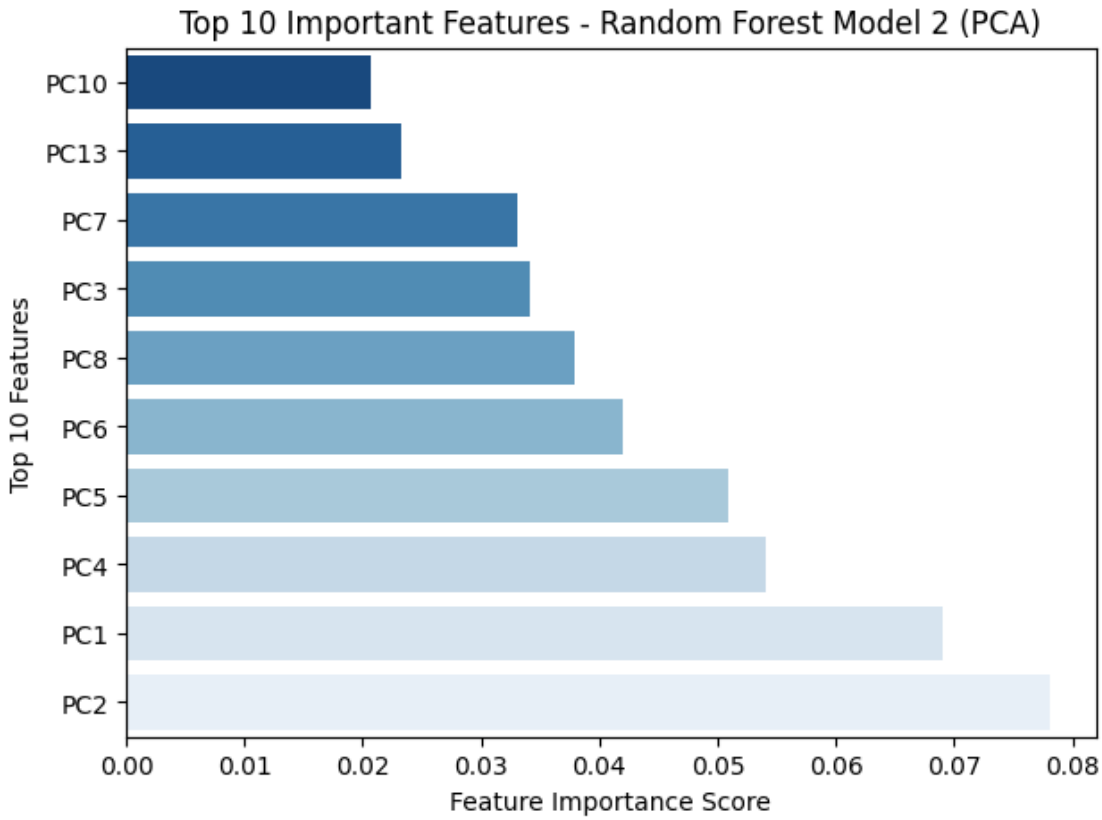
Top 10 Features of Model 02

[607]:
```
feature_importances_pca = grid_rf_pca.best_estimator_.feature_importances_
```

[608]:
```
feature_names_pca = [f'PC{i+1}' for i in range(len(feature_importances_pca))] ␣
↪# Naming as PC1, PC2, etc.
```

[609]:
```
top_n = 10
top_indices = np.argsort(feature_importances_pca)[-top_n:]
top_features_pca = np.array(feature_names_pca)[top_indices]
top_importance_values_pca = feature_importances_pca[top_indices]
```

[610]:
```
plt.figure(figsize=(7, 5))
sns.barplot(x=top_importance_values_pca, y=top_features_pca, palette="Blues_r")
plt.xlabel("Feature Importance Score")
plt.ylabel("Top 10 Features")
plt.title("Top 10 Important Features - Random Forest Model 2 (PCA)")
plt.show()
```



Model 03

[611]:
```
y_pred_kmeans = model_03.predict(X_kmeans)
```

[612]:
```
silhouette_avg = silhouette_score(X_kmeans, y_pred_kmeans)
print(f"The Silhouette Score for K-Means Model: {silhouette_avg:.5f}")
```

```
The Silhouette Score for K-Means Model: 0.06030
```

[613]:
```
inertia = model_03.inertia_
print(f"Inertia for Model 03 (K-Means): {inertia:.5f}")
```

```
Inertia for Model 03 (K-Means): 1895833.46431
```

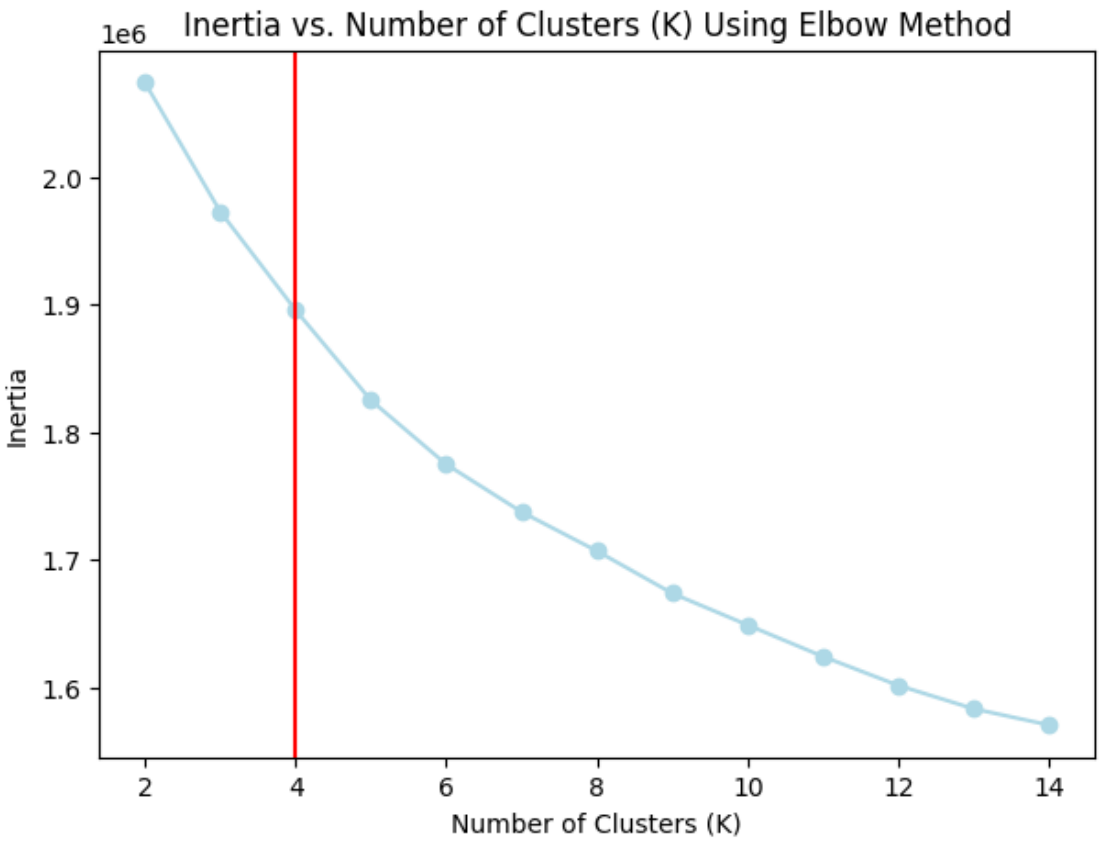
Elbow Method Graph of Model 03

[614]:
```
k_values = range(2, 15)
inertia_values = []

time_start_kmeans = time.time()


for k in k_values:
```

```
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_kmeans)
    inertia_values.append(kmeans.inertia_)

time_end_kmeans = time.time()
print(f"Time taken for K-Means clustering: {time_end_kmeans - time_start_kmeans:
  ↪.2f} seconds")
```

```
Time taken for K-Means clustering: 13.95 seconds
```

[615]:
```
plt.figure(figsize=(7, 5))
plt.plot(k_values, inertia_values, marker='o', color="lightblue",␣
  ↪label="Inertia")

plt.axvline(x=best_k_silhouette_score, color='red', label=f"Best K =␣
  ↪{best_k_silhouette_score}")
plt.title("Inertia vs. Number of Clusters (K) Using Elbow Method")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.show()
```



KAGGLE SUBMISSIONS FOR MODEL 01, 02, AND 03

Model 01 submission

[616]:
```
X_test_rf = test_digit_recognizer_dataframe.drop(columns=['label'],␣
↪errors='ignore')
```

[617]:
```
y_pred_kaggle_rf = grid_rf.best_estimator_.predict(X_test_rf)
```

[618]:
```
kaggle_submission_df_rf = pd.DataFrame({
    "ImageId": range(1, len(y_pred_kaggle_rf) + 1),
    "Label": y_pred_kaggle_rf
})
```

[619]:
```
kaggle_submission_file_rf = "/Users/isingh/Desktop/digit-recognizer/
↪random_forest_submission.csv"
```

[620]:
```
kaggle_submission_df_rf.to_csv(kaggle_submission_file_rf, index=False)
```

[621]:
```
print(f"Model 01 submission file saved as: {kaggle_submission_file_rf}")
```

```
Model 01 submission file saved as: /Users/isingh/Desktop/digit-
recognizer/random_forest_submission.csv
```

Model 02 submission

[622]:
```
X_test_pca = pca.transform(test_digit_recognizer_dataframe.
↪drop(columns=['label'], errors='ignore'))
```

[623]:
```
y_pred_kaggle_rf_pca = grid_rf_pca.best_estimator_.predict(X_test_pca)
```

[624]:
```
kaggle_submission_df_rf_pca = pd.DataFrame({
    "ImageId": range(1, len(y_pred_kaggle_rf_pca) + 1),
    "Label": y_pred_kaggle_rf_pca
})
```

[625]:
```
kaggle_submission_file_rf_pca = "/Users/isingh/Desktop/digit-recognizer/
↪random_forest_pca_submission.csv"
```

[626]:
```
kaggle_submission_df_rf_pca.to_csv(kaggle_submission_file_rf_pca, index=False)
```

[627]:
```
print(f"Model 02 submission file saved as: {kaggle_submission_file_rf_pca}")
```

```
Model 02 submission file saved as: /Users/isingh/Desktop/digit-
recognizer/random_forest_pca_submission.csv
```

Model 03 submission

[628]:
```
X_test_kmeans = scaler.transform(test_digit_recognizer_dataframe.
↪drop(columns=['label'], errors='ignore'))
```

[629]:
```
y_pred_kmeans = model_03.predict(X_test_kmeans)
```

[630]:
```
kaggle_submission_df_kmeans = pd.DataFrame({
    "ImageId": range(1, len(y_pred_kmeans) + 1),
    "Label": y_pred_kmeans
})
```

[631]:
```
kaggle_submission_file_kmeans = "/Users/isingh/Desktop/digit-recognizer/
↪kmeans_submission.csv"
```

[632]:
```
kaggle_submission_df_kmeans.to_csv(kaggle_submission_file_kmeans, index=False)
```

[633]:
```
print(f"Model 03 submission file saved as: {kaggle_submission_file_kmeans}")
```

```
Model 03 submission file saved as: /Users/isingh/Desktop/digit-
recognizer/kmeans_submission.csv
```

1.0.16 Analysis of Model Performance

Overall, from the Kaggle submission, Model 1 that was performed with Random Forest performed the best with a Kaggle score of 0.96460, which meant that it was able to classify the digits in a proper manner. The next highest was Model 2 that performed using Random Forest but with PCA and had a Kaggle score of 0.62010. The reason for that must have been that as we kept 95 percent of the important features but removed 5 percent of them to reduce the dimensions, it gave us a low score. Model 3 was about K-Means Clustering, which had the lowest score overall at 0.23646, which made sense as K-Means is an unsupervised model—overall, it does not use labeled data and hence it uses patterns and distance to form clusters rather than actual digit labels. As we know, Kaggle evaluated the submissions based on correct digit labels, so K-Means did not work efficiently. Model 3 grouped similar-looking digits into clusters, which created a problem as the clusters were formed based on feature similarities rather than actual labels, and however, cluster numbers do not match actual labels, and that is the reason why the accuracy score was so low.

For Model 4, some of the ways we can fix this include matching clusters to digits that appear in each cluster, using a semi-supervised approach by training a small classifier to match clusters to digits, trying PCA before K-Means, tuning parameters, and exploring a different unsupervised method.

1.0.17 Model 04

[634]:
```
pca_model_04 = PCA(n_components=0.95, random_state=42)
X_pca_kmeans_model04 = pca_model_04.fit_transform(X_scaled)
```

[635]:
```
k_values = range(2, 15)
silhouette_scores = []
start_time = time.time()
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_pca_kmeans_model04)
```

```
    score = silhouette_score(X_pca_kmeans_model04, labels)
    silhouette_scores.append(score)
end_time = time.time()

best_k = k_values[np.argmax(silhouette_scores)]
print(f"Best K for Model 04 is: {best_k}")
print(f"Time taken for K-Means clustering in Model 04: {end_time - start_time:.
 ↪2f} seconds")
```

```
Best K for Model 04 is: 2
Time taken for K-Means clustering in Model 04: 190.23 seconds
```

[636]:
```
model_04 = KMeans(n_clusters=best_k, random_state=42, n_init=10)
y_clusters = model_04.fit_predict(X_pca_kmeans_model04)
```

[637]:
```
cluster_labels = {}
for i in range(best_k):
    cluster_members = (y_clusters == i)
    if np.any(cluster_members):
        cluster_labels[i] = np.bincount(y[cluster_members]).argmax()

y_train_kmeans = np.array([cluster_labels[label] for label in y_clusters])
```

[638]:
```
start_time_model04 = time.time()
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(y_clusters.reshape(-1, 1), y_train_kmeans)
end_time_model04 = time.time()
```

[639]:
```
print(f"Training time for Model 04: {end_time - start_time:.2f} seconds")
```

```
Training time for Model 04: 190.23 seconds
```

[640]:
```
y_pred_rf_model4 = rf_classifier.predict(y_clusters.reshape(-1, 1))
```

[641]:
```
accuracy_model4 = accuracy_score(y_train_kmeans, y_pred_rf_model4)
print(f"Model 04 Accuracy: {accuracy_model4:.5f}")
```

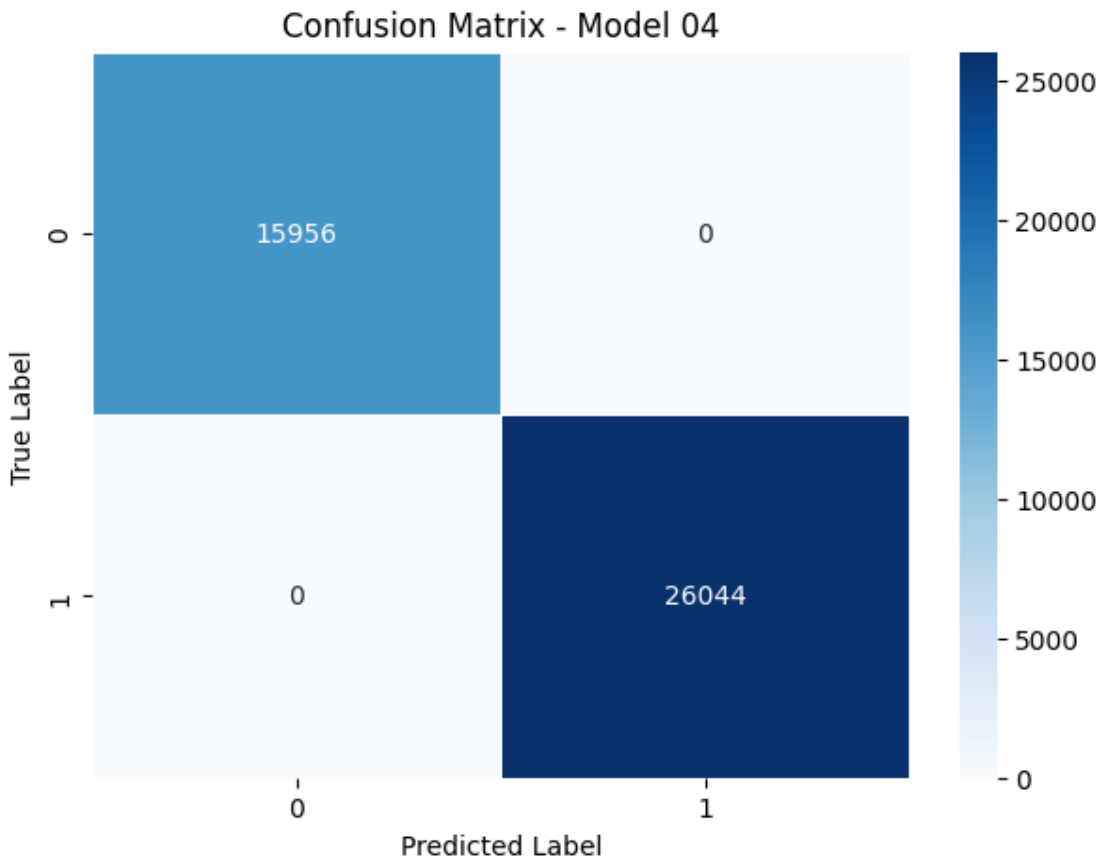
```
Model 04 Accuracy: 1.00000
```

[642]:
```
print("Classification Report for Model 04:")
print(classification_report(y_train_kmeans, y_pred_rf_model4))
```

```
Classification Report for Model 04:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     15956
           1       1.00      1.00      1.00     26044

    accuracy                           1.00     42000
```

```
   macro avg       1.00      1.00      1.00     42000
weighted avg       1.00      1.00      1.00     42000
```

[643]:
```
conf_matrix_model4 = confusion_matrix(y_train_kmeans, y_pred_rf_model4)
plt.figure(figsize=(7, 5))
sns.heatmap(conf_matrix_model4, annot=True, fmt='d', cmap="Blues", linewidths=0.
↪5)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Model 04")
plt.show()
```



Kaggle submission for Model 04

[644]:
```
X_test_pca_kmeans = pca_model_04.transform(X_test_kmeans)
```

[645]:
```
y_test_clusters = model_04.predict(X_test_pca_kmeans)
```

[646]:
```
y_test_kmeans = np.array([cluster_labels[label] for label in y_test_clusters])
```

[647]:
```
kaggle_submission_df_kmeans_model04 = pd.DataFrame({
    "ImageId": range(1, len(y_test_kmeans) + 1),
    "Label": y_test_kmeans
})
```

[648]:
```
kaggle_submission_file_kmeans = "/Users/isingh/Desktop/digit-recognizer/
↪model_04_kmeans_submission.csv"
```

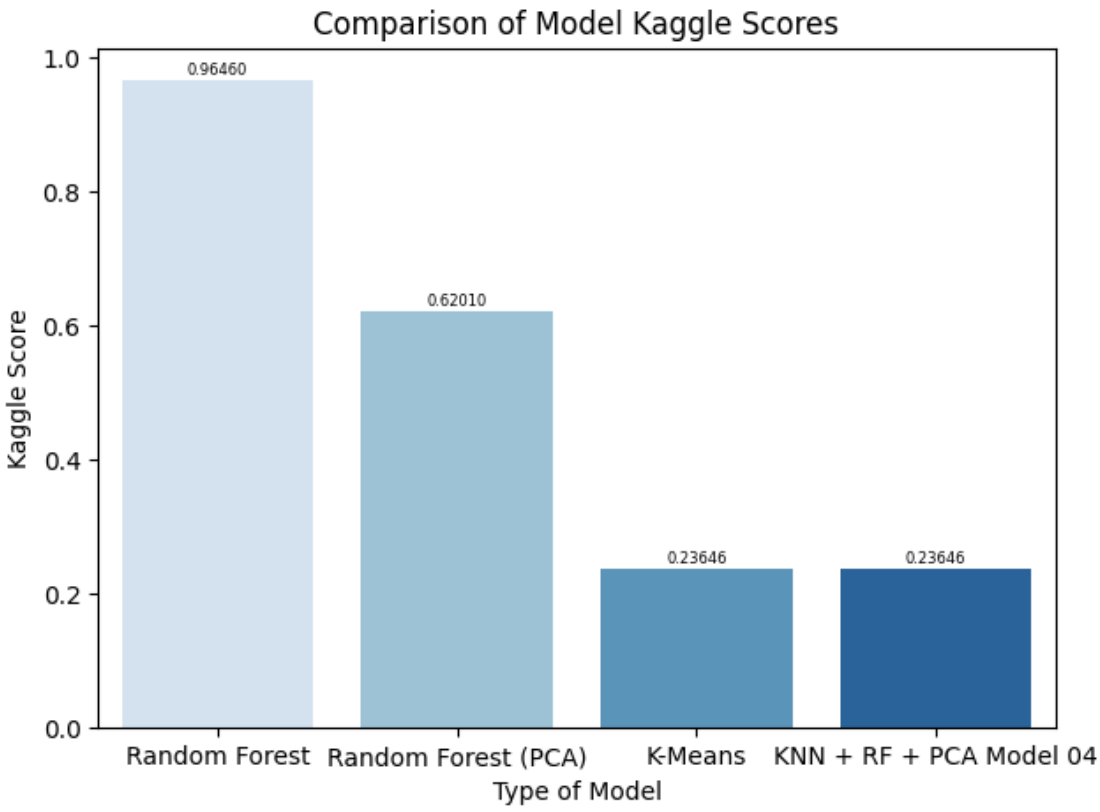
[649]:
```
kaggle_submission_df_kmeans.to_csv(kaggle_submission_file_kmeans, index=False)
```

[650]:
```
print(f"Model 04 submission file saved as: {kaggle_submission_file_kmeans}")
```

```
Model 04 submission file saved as: /Users/isingh/Desktop/digit-
recognizer/model_04_kmeans_submission.csv
```

Kaggle Scores Comparison

[651]:
```
models = ["Random Forest", "Random Forest (PCA)", "K-Means", "KNN + RF + PCA␣
↪Model 04"]
kaggle_scores = [0.96460, 0.62010, 0.23646, 0.23646]

plt.figure(figsize=(7, 5))
ax = sns.barplot(x=models, y=kaggle_scores, palette="Blues")

for i, height in enumerate(kaggle_scores):
    ax.text(i, height + 0.01, f"{height:.5f}", ha='center', fontsize=6)


plt.xlabel("Type of Model")
plt.ylabel("Kaggle Score")
plt.title("Comparison of Model Kaggle Scores")
plt.show()
```



1.0.18 PART 17 Conclusion

Overall, this research was a great way to learn how each model works with handwriting recognition. The Random Forest model, which was the first model, gave the best accuracy overall with a score of 0.96460, meaning the hyperparameters worked well. Secondly, the Random Forest with PCA had a lower score of 0.62010 as we had to reduce dimensions, which possibly removed important information. K-means clustering was the third model with a score of 0.23646 since it did not use labeled data and instead clustered similar data, making it difficult to match the clusters to the digits.The last model was improved through a K Means approach and then afterward using the random forest approach once the clusters were matched to digit labels. However, sadly, after testing this experiment out, the model's score was the same. That overall meant that the model was still not able to capture the classification of the digits accurately. Therefore, in the future, focusing on hyperparameters and using alternative methods will be better.

1.0.19 PART 18 Management Question

The goal of this machine learning research was to be able to develop an efficient model that will recognize different and unique handwritten digits. This is useful in the outside world to be able to understand when in work scenario one is not able to understand a specific numeric. Improving accuracy with the help of feature selection

and clustering will help create models that will be able to assist onto the next step: generalizing handwritten digits, which is crucial for real world and practical settings.

1.0.20 PART 20 References

- ChatGPT. *Accessed February 26, 2025.* https://chatgpt.com/
- "Confusion_matrix." *scikit-learn.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- "Digit Recognizer." *Kaggle.* Accessed February 26, 2025. https://www.kaggle.com/competitions/digit-recognizer
- GeeksforGeeks. "Python Most_common() Function." *GeeksforGeeks*, March 20, 2024. https://www.geeksforgeeks.org/python-most_common-function/
- GeeksforGeeks. "Recovering Feature Names of Explained_variance_ratio_ in PCA with Sklearn." *GeeksforGeeks*, June 27, 2024. https://www.geeksforgeeks.org/recovering-feature-names-of-explainedvarianceratio-in-pca-with-sklearn/
- "KMeans." *scikit-learn.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- "MinMaxScaler." *scikit-learn.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- "PCA." *scikit-learn.* Accessed February 22, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- "RandomForestClassifier." *scikit-learn.* Accessed February 25, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html