Task 1

Tasks: Load the Data: Load the dataset and display the first few rows. Data Preprocessing:Handle any missing values.Split the dataset into training and testing sets. Feature Engineering:Normalize the pixel values to be between 0 and 1. Model Building:Build a classification model to recognize the digits.Train the model on the training data.Evaluate the model on the testing data using appropriate metrics (e.g., accuracy, precision, recall, F1-score) .Model Interpretation:Interpret the feature importances or model coefficients. Optional: Model ImprovementTry different algorithms and hyperparameters to improve the model's performance.

```
#Loading the data set and

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.metrics import classification_report

data= pd.read_csv(r"C:\Users\91965\Downloads\mnist_test.csv\
mnist_test.csv")
data. head()
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19
28x20  \
0      7    0    0    0    0    0    0    0    0    0  ...      0
0
1      2    0    0    0    0    0    0    0    0    0  ...      0
0
2      1    0    0    0    0    0    0    0    0    0  ...      0
0
3      0    0    0    0    0    0    0    0    0    0  ...      0
0
4      4    0    0    0    0    0    0    0    0    0  ...      0
0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

```
labels = data.iloc[:, 0]  # Assuming labels are in the first column
features = data.iloc[:, 1:]  # Assuming features start from the second column

# Print the shapes to verify
print("Labels shape:", labels.shape)
print("Features shape:", features.shape)
```

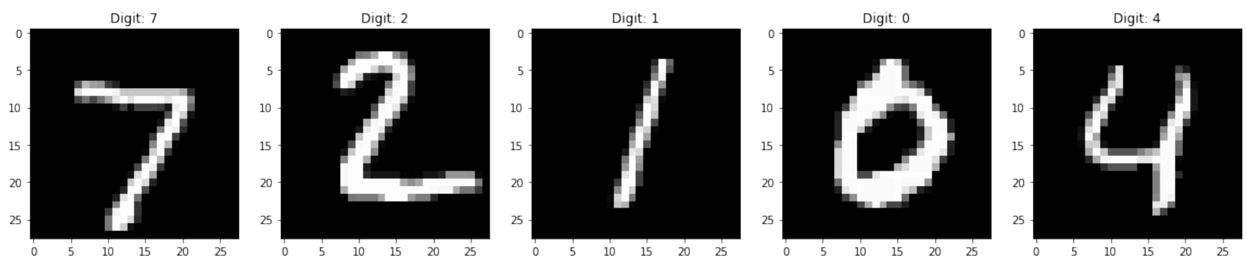
```
Labels shape: (10000,)
Features shape: (10000, 784)
```

```
#Displahing top 5 images of digits

samples = 5
fig, axes = plt.subplots(1, samples, figsize=(20, 4))

for i in range(samples):
    ax = axes[i]
    image = features.iloc[i].values.reshape(28, 28)
    ax.imshow(image, cmap='gray')
    ax.set_title(f'Digit: {labels.iloc[i]}')

plt.show()
```




```
# now let's handle any missing values that we have

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, label to 28x28
dtypes: int64(785)
memory usage: 59.9 MB
```

```
missing_values= data.isnull().sum()
missing_values.sum()

# it is seems like there are no missing value in this data. let's move to next steps.
```

```
0

from sklearn.model_selection import train_test_split

#Split the dataset into train test split

y= labels
X= features

#train test split

X_train, X_test, y_train, y_test= train_test_split(X, y,
test_size=0.2, random_state= 42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

((8000, 784), (2000, 784), (8000,), (2000,))



#Normalize the pixel values to be between 0 and 1.
#let's use MinMaxScaler to normalise

scaler = MinMaxScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)
X_train_norm

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

X_test_norm.shape

(2000, 784)

X_train_norm.shape, X_test_norm.shape

((8000, 784), (2000, 784))

X_test_norm

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```python
X_train_norm = X_train_norm.reshape(-1, 28, 28, 1)  # Adjust
dimensions as per your data
X_test_norm = X_test_norm.reshape(-1, 28, 28, 1)

# Now let's build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
C:\Users\91965\anaconda3\lib\site-packages\keras\src\layers\
convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```python
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train_norm, y_train, epochs=5, batch_size=64,
validation_data=(X_test_norm, y_test))
```

```
Epoch 1/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.6601 - loss:
1.1418 - val_accuracy: 0.9370 - val_loss: 0.2234
Epoch 2/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9552 - loss:
0.1580 - val_accuracy: 0.9585 - val_loss: 0.1431
Epoch 3/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9741 - loss:
0.0887 - val_accuracy: 0.9695 - val_loss: 0.1119
Epoch 4/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.9831 - loss:
0.0545 - val_accuracy: 0.9725 - val_loss: 0.1057
Epoch 5/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.9851 - loss:
0.0431 - val_accuracy: 0.9695 - val_loss: 0.0993

<keras.src.callbacks.history.History at 0x17aadad8f10>
```

```
# Train the model
model.fit(X_train_norm, y_train, epochs=5, batch_size=64,
validation_data=(X_test_norm, y_test))
```

```
Epoch 1/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.9900 - loss:
0.0294 - val_accuracy: 0.9705 - val_loss: 0.1154
Epoch 2/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9897 - loss:
0.0348 - val_accuracy: 0.9730 - val_loss: 0.1101
Epoch 3/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9939 - loss:
0.0183 - val_accuracy: 0.9755 - val_loss: 0.1033
Epoch 4/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9979 - loss:
0.0094 - val_accuracy: 0.9650 - val_loss: 0.1283
Epoch 5/5
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9956 - loss:
0.0165 - val_accuracy: 0.9765 - val_loss: 0.0986
```

```
<keras.src.callbacks.history.History at 0x17aadaee190>
```

```
y_pred = model.predict(X_test_norm)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_classes))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       203
           1       0.98      1.00      0.99       216
           2       0.97      0.99      0.98       213
           3       0.98      0.97      0.98       208
           4       0.98      0.97      0.97       215
           5       0.97      0.99      0.98       174
           6       0.97      0.98      0.98       200
           7       0.98      0.98      0.98       187
           8       0.98      0.94      0.96       186
           9       0.97      0.96      0.97       198

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000
```

```
#let's try ML model with hyper- parameter


svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
```

```python
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       203
           1       0.99      0.99      0.99       216
           2       0.96      0.97      0.96       213
           3       0.96      0.97      0.96       208
           4       0.94      0.95      0.95       215
           5       0.97      0.96      0.97       174
           6       0.95      0.95      0.95       200
           7       0.95      0.97      0.96       187
           8       0.99      0.96      0.97       186
           9       0.96      0.93      0.94       198

    accuracy                           0.96      2000
   macro avg       0.96      0.96      0.96      2000
weighted avg       0.96      0.96      0.96      2000
```