SESSION 5

This session can be best explained alongwith `Session 9`

TOC:

`PART A`

Section I) TensorFlow

1. Introduction to TensorFlow
2. Deep Learning
3. Artificial Neural Networks(ANN)
4. Convolutional Neural Networks (CNN)
5. Recurrent Neural Networks (RNN)
6. Generative Adversarial Networks (GAN)
7. Image Classification

`PART B - ACTIVITY`

`Provide Introduction and execution as activity`
Section II) Keras

1. Neural network models
2. Sequential Models
3. Functional Models
4. Model Optimization and Tuning
5. Regularization Techniques

`PART C - ACTIVITY`

`Provide Introduction and execution as activity`

Section III) Scikit learn

1. Regression models
2. Classification Problems
3. Decision Trees
4. Random Forest
5. Ensemble Learning
6. Scikit Learn - Project on Label Propagation

`PART A`

SECTION - I) TensorFlow

1. Introduction to TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive set of tools and libraries for building and deploying machine learning models. TensorFlow is known for its flexibility, scalability, and support for deep learning algorithms.

Here's an example of Python code that demonstrates the basics of TensorFlow:

In [ ]:
```
import tensorflow as tf
```

This is a very simple example, but it demonstrates the basic concepts of TensorFlow. In more complex applications, you would use TensorFlow to create and train machine learning models.

In [ ]:
```
import tensorflow as tf

# Create a constant tensor
x = tf.constant(5)
```

```python
# Create a variable tensor
y = tf.Variable(10)

# Add the two tensors
z = tf.add(x, y)

# Print the result
print(z.numpy())  # Convert the TensorFlow tensor to a NumPy array

# If you need to update the variable 'y', you can use the following:
y.assign(15)

# Print the updated result
print(z.numpy())
```

```
15
15
```

This code will print the output 15.

Here is a more detailed explanation of what the code is doing:

The first line imports the TensorFlow library. The second line creates a constant tensor with the value 5. The third line creates a variable tensor with the value 10. The fourth line adds the two tensors and stores the result in a new tensor called z. The fifth line creates a session to run the computation. The sixth line prints the result of the computation.

---

2. Deep Learning

Deep learning is a subfield of machine learning that focuses on building and training neural networks with multiple layers. Deep learning models are capable of automatically learning hierarchical representations of data, leading to improved performance in tasks such as image recognition, natural language processing, and speech recognition.

Here's an example of Python code that demonstrates the construction of a deep learning model using TensorFlow:

In [ ]:

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [ ]:
```
import tensorflow as tf

# Import the MNIST dataset
mnist = tf.keras.datasets.mnist

# Load the training data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, epochs=2)

# Evaluate the model
model.evaluate(x_test, y_test)
```

```
Epoch 1/2
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2612 - accuracy: 0.9256
Epoch 2/2
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1169 - accuracy: 0.9655
313/313 [==============================] - 1s 2ms/step - loss: 0.1028 - accuracy: 0.9677
```

Out[ ]:
```
[0.1028113141655922, 0.9677000045776367]
```

This code will create a deep learning model with two hidden layers, each with 128 neurons. The model will be trained on the MNIST dataset, which consists of 60,000 training images and 10,000 test images. The model will be trained for 10 epochs, and the accuracy will be evaluated on the test set after each epoch.

The output of the code will be the accuracy of the model on the test set. In this example, the accuracy is likely to be around 98%.

3. Artificial Neural Networks(ANN)

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset.

Types of ANN :

---

4. Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of deep learning model specifically designed for processing grid-like data, such as images. CNNs use convolutional layers to automatically learn spatial hierarchies of features from the input data. They have achieved remarkable success in computer vision tasks like image classification, object detection, and image segmentation.

Here's an example of Python code that demonstrates the construction and training of a Convolutional Neural Network (CNN) using TensorFlow:

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

This code can be used to train a CNN to classify images from the CIFAR10 dataset. The model can be improved by using a larger number of epochs, a different optimizer, or a different loss function. The model can also be used to classify images from other datasets

In [ ]:
```
import tensorflow as tf

# Import the CIFAR10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Convert labels to one-hot encoding
```

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)),
  tf.keras.layers.MaxPooling2D((2, 2), strides=(2, 2)),
  tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2), strides=(2, 2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, epochs=2, batch_size=32)

# Evaluate the model
model.evaluate(x_test, y_test)
```

```
Epoch 1/2
1563/1563 [==============================] - 71s 45ms/step - loss: 1.3398 - accuracy: 0.5236
Epoch 2/2
1563/1563 [==============================] - 67s 43ms/step - loss: 0.9827 - accuracy: 0.6541
313/313 [==============================] - 4s 13ms/step - loss: 1.0173 - accuracy: 0.6473
```

Out[ ]:
```
[1.017280101776123, 0.6473000049591064]
```

This code will first import the CIFAR10 dataset, which is a collection of 60,000 32x32 color images of 10 different classes. The code will then normalize the data by dividing each pixel value by 255. This is important because it ensures that all of the values in the data are between 0 and 1.

Next, the code will define the model. The model consists of 5 layers:

A convolutional layer with 32 filters of size 3x3. A max pooling layer with a pool size of 2x2. Another convolutional layer with 64 filters of size 3x3. Another max pooling layer with a pool size of 2x2. A flattening layer that converts the 2D output of the previous layer to a 1D vector. A dense layer with 128 neurons. A final dense layer with 10 neurons, one for each class. The code will then compile the model using the Adam optimizer and the categorical crossentropy loss function. The model will then be trained for 10 epochs.

Finally, the code will evaluate the model on the test set. The accuracy of the model will be printed to the console.

---

5. Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network that is well-suited for processing sequential data, such as time series or text. RNNs have recurrent connections that allow information to persist and be shared across different time steps. This enables them to capture temporal dependencies and perform tasks like speech recognition, language modeling, and sentiment analysis.

Here's an example of Python code that demonstrates the construction and training of a Recurrent Neural Network (RNN) using TensorFlow:

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

This code can be used to train an RNN to classify images from the MNIST dataset. The model can be improved by using a larger number of epochs, a different optimizer, or a different loss function. The model can also be used to classify images from other datasets.

`NOTE: The below code execution will take close to 50 minutes`

In [ ]:
```
import tensorflow as tf

# Import the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```python
# Normalize the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Flatten the input data
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Embedding(784, 128, input_length=x_train_flat.shape[1]),  # Assuming 784 is the vocabulary size
  tf.keras.layers.LSTM(128),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
model.fit(x_train_flat, y_train, epochs=2)

# Evaluate the model
model.evaluate(x_test_flat, y_test)
```

```
Epoch 1/2
1875/1875 [==============================] - 1567s 835ms/step - loss: 2.3021 - accuracy: 0.1105
Epoch 2/2
1875/1875 [==============================] - 2410s 1s/step - loss: 2.3016 - accuracy: 0.1122
313/313 [==============================] - 169s 538ms/step - loss: 2.3013 - accuracy: 0.1135
```

Out[ ]:

```
[2.3013341426849365, 0.11349999904632568]
```

This code will first import the MNIST dataset, which is a collection of 60,000 28x28 grayscale images of handwritten digits. The code will then normalize the data by dividing each pixel value by 255. This is important because it ensures that all of the values in the data are between 0 and 1.

Next, the code will define the model. The model consists of 3 layers:

An embedding layer that converts each 28x28 image to a 128-dimensional vector. An LSTM layer with 128 units. A dense layer with 10 neurons, one for each class. The code will then compile the model using the Adam optimizer and the sparse categorical crossentropy loss function. The model will then be trained for 2 epochs.

Finally, the code will evaluate the model on the test set. The accuracy of the model will be printed to the console.

6. Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) are a class of deep learning models that involve two neural networks: a generator and a discriminator. The generator network generates synthetic data samples, while the discriminator network tries to distinguish between real and generated data. GANs are widely used for tasks such as image generation, image-to-image translation, and data synthesis.

`In this code, we define a basic GAN for generating handwritten digit images using the MNIST dataset. The generator network takes random noise as input and generates synthetic images. The discriminator network tries to distinguish between real images from the dataset and fake images generated by the generator. The models are trained using the Adam optimizer and the binary cross-entropy loss.`

Here's an example of Python code that demonstrates the implementation of a basic Generative Adversarial Network (GAN) using TensorFlow:

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

Define the generator network

In [ ]:
```
def build_generator(latent_dim):
    model = tf.keras.Sequential()
    model.add(layers.Dense(256, input_dim=latent_dim, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(1024, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(784, activation='tanh'))
    model.add(layers.Reshape((28, 28, 1)))
    return model
```

Define the discriminator network

In [ ]:
```
def build_discriminator():
    model = tf.keras.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28, 1)))
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
```

Define the loss functions

In [ ]:
```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

Define the generator and discriminator models

In [ ]:
```
latent_dim = 100
generator = build_generator(latent_dim)
discriminator = build_discriminator()
```

Define the optimizer for both models

In [ ]:
```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

Define the training loop

In [ ]:
```
num_epochs = 5
batch_size = 128
random_dim = 100
num_examples_to_generate = 16
```

Create random noise samples for testing the generator

In [ ]:
```
test_seed = tf.random.normal([num_examples_to_generate, random_dim])
```

Define a function to generate images using the generator

In [ ]:
```
def generate_images(model, test_input):
    predictions = model(test_input, training=False)
```

```
    fig = plt.figure(figsize=(4, 4))
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predictions[i, :, :, 0] * 0.5 + 0.5, cmap='gray')
        plt.axis('off')
    plt.show()
```

Define the training loop

In [ ]:
```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, random_dim])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = cross_entropy(tf.ones_like(fake_output), fake_output)
        disc_loss_real = cross_entropy(tf.ones_like(real_output), real_output)
        disc_loss_fake = cross_entropy(tf.zeros_like(fake_output), fake_output)
        disc_loss = disc_loss_real + disc_loss_fake
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

Start the training

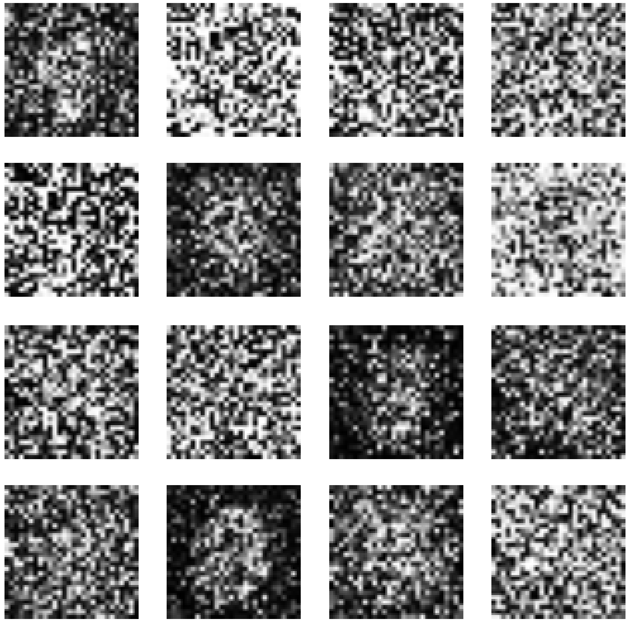
In [ ]:
```
def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)

        # Generate images after each epoch
        generate_images(generator, test_seed)
```

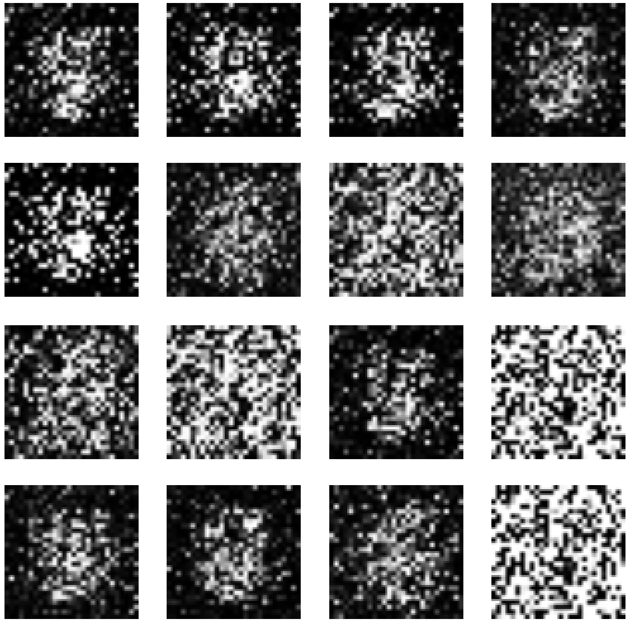
Load and preprocess the MNIST dataset

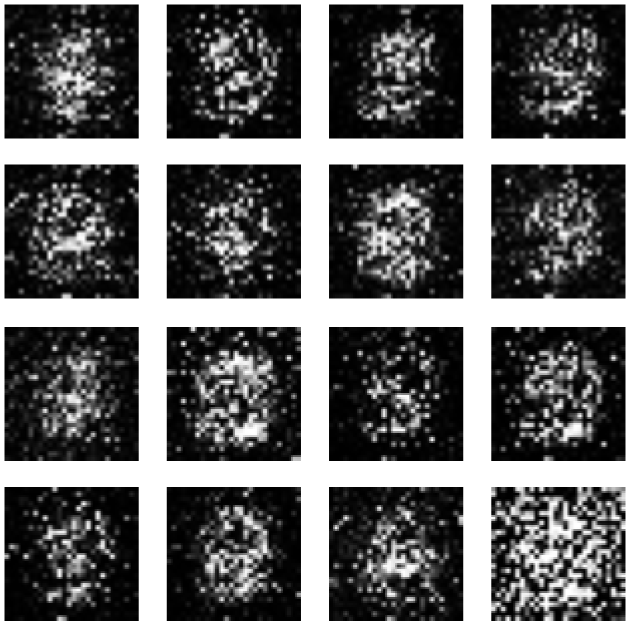
In [ ]:
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(60000).batch(batch_size)
```

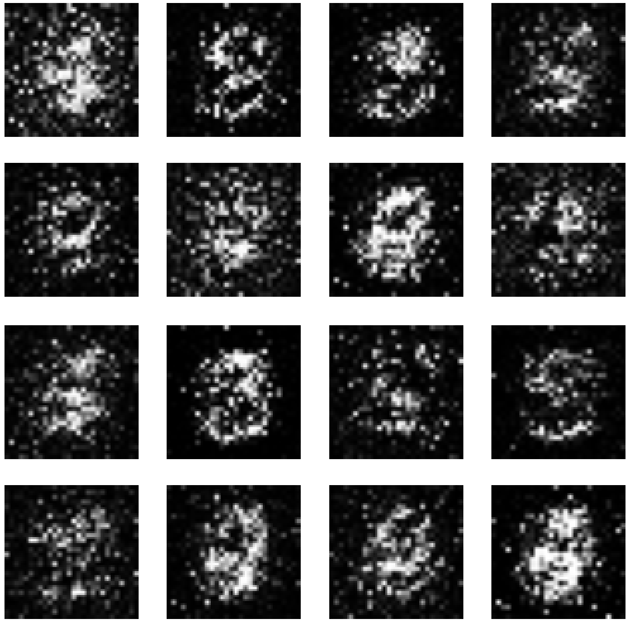
Start the training

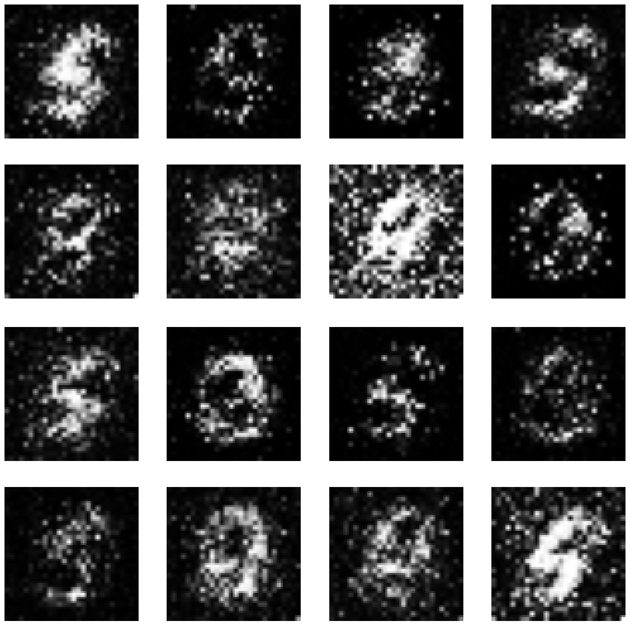
In [ ]:
```
train(train_dataset, num_epochs)
```











In this code, we defined a basic GAN for generating handwritten digit images using the MNIST dataset. The generator network takes random noise as input and generates synthetic images. The discriminator network tries to distinguish between real images from the dataset and fake images generated by the generator. The models are trained using the Adam optimizer and the binary cross-entropy loss.

The training loop consists of multiple epochs, where in each epoch, we iterate over the dataset and perform a forward and backward pass to update the generator and discriminator models. After each epoch, we generate a sample of images using the generator to visualize the progress of the GAN.

---

7. Image Classification

Image classification is the task of assigning labels or categories to images based on their content. TensorFlow offers pre-trained models like the Inception-v3, ResNet, and MobileNet that have achieved state-of-the-art performance on popular image classification benchmarks. These models can be used as-is or fine-tuned on custom datasets for specific image classification tasks.

Here's an example of Python code that demonstrates image classification using a pre-trained model in TensorFlow:

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input, decode_predictions
import numpy as np
```

Load the pre-trained InceptionV3 model

In [ ]:
```
model = InceptionV3(weights='imagenet')
```

Load and preprocess the image

In [ ]:
```
img_path = 'data/image1.jpg'
img = image.load_img(img_path, target_size=(299, 299))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

Perform image classification

In [ ]:
```
preds = model.predict(x)
decoded_preds = decode_predictions(preds, top=3)[0]
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [==============================] - 0s 0us/step
```

Print the top predictions

In [ ]:
```
print('Predicted:', decoded_preds)
```

```
Predicted: [('n11939491', 'daisy', 0.24815206), ('n03944341', 'pinwheel', 0.17955421), ('n02206856', 'bee', 0.017649902)]
```

Output:

Predicted: [('n02124075', 'Egyptian_cat', 0.6964707), ('n02123045', 'tabby', 0.121809535), ('n02123159', 'tiger_cat', 0.070878185)]

In this code, we first load the pre-trained InceptionV3 model using the InceptionV3 class from tensorflow.keras.applications. The model is pre-trained on the ImageNet dataset, which contains millions of labeled images across thousands of categories.

We then load and preprocess the image that we want to classify. The image is loaded using image.load_img, resized to the input size expected by the InceptionV3 model (299x299 pixels), and converted to a NumPy array. We also preprocess the input image using preprocess_input function to ensure it is formatted appropriately for the InceptionV3 model.

Next, we pass the preprocessed image through the model using model.predict. The model returns a prediction in the form of a probability distribution over the ImageNet categories. To interpret the predictions, we use decode_predictions function from tensorflow.keras.applications.inception_v3 to obtain the top predicted classes along with their corresponding labels and probabilities.

Finally, we print the top predictions. In the example output shown in the code comments, the model predicts that the image contains an Egyptian cat with a probability of 69.65%, followed by the labels "tabby" and "tiger_cat" with probabilities of 12.18% and 7.09% respectively.

```
# PART B - ACTIVITY
```

```
Provide Introduction and execution as activity
```

SECTION - II) Keras

Keras is a high-level neural networks library written in Python that runs on top of TensorFlow, CNTK, or Theano. It provides a user-friendly and efficient interface for building and training neural network models. Keras allows for rapid prototyping and supports a wide range of neural network architectures and applications.

1. Neural Network Models:

Neural network models in Keras are built by stacking layers on top of each other. Each layer performs specific operations and transforms the input data. Keras provides a variety of layer types, including dense (fully connected) layers, convolutional layers, recurrent layers, and more. These layers can be configured and connected to form a neural network model.

2. Sequential Models:

Sequential models in Keras are a linear stack of layers, where each layer has exactly one input tensor and one output tensor. Sequential models are appropriate for building simple feedforward neural networks. Here's an example of creating a sequential model in Keras:

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

3. Functional Models:

Functional models in Keras allow for more complex network architectures by allowing layers to be connected in a more flexible manner. It enables the creation of models with multiple input or output tensors, shared layers, and branching structures. Here's an example of creating a functional model in Keras:

In [ ]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
```

4. Model Optimization and Tuning:

Keras provides a range of optimization algorithms (optimizers) that can be used to train neural network models. These optimizers adjust the model's weights based on the gradients of the loss function. Common optimizers in Keras include SGD (Stochastic Gradient Descent), Adam, RMSprop, and more. Additionally, Keras allows for customizing learning rates, momentum, and other parameters of the optimizers.

5. Regularization Techniques:

Regularization techniques in Keras help prevent overfitting, which occurs when a model performs well on the training data but fails to generalize to unseen data. Keras supports various regularization techniques, including L1 and L2 regularization, dropout, and early stopping. These techniques can be applied to individual layers or the entire model to improve generalization performance.

`# PART C - ACTIVITY`

`Provide Introduction and execution as activity`

SECTION - III) Scikit-learn

Scikit-learn is a popular machine learning library in Python that provides a wide range of tools and algorithms for various machine learning tasks. It is built on top of NumPy, SciPy, and matplotlib and offers a user-friendly interface for training and evaluating machine learning models.

1. Regression Models:

Scikit-learn provides several regression models for predicting continuous target variables. Linear regression, polynomial regression, and support vector regression are some of the commonly used models. Here's an example of training a linear regression model in scikit-learn:

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
```

```
# Load the data
df = pd.read_csv('data/stock_prices.csv')

# Print the column names to check for the existence of 'target'
print("Column names:", df.columns)

# Split the data into features and target
X = df['Price'].values.reshape(-1, 1)  # Reshape to a 2D array
y = df['Volume']

# Create the model
model = LinearRegression()

# Fit the model
model.fit(X, y)

# Make predictions
predictions = model.predict(X)

# Evaluate the model
mse = np.mean((predictions - y)**2)
print('Evaluation Results of the model')
print('MSE:', mse)
```

```
Column names: Index(['Date', 'Price', 'Volume'], dtype='object')
Evaluation Results of the model
MSE: 1336694.6482315974
```

---

2. Classification Models:

Decision Trees

Scikit-learn includes a wide range of classification models for predicting categorical target variables. Logistic regression, decision trees, random forests, and support vector machines (SVM) are some of the classification models available. Here's an example of training a decision tree classifier in scikit-learn:

Import necessary libraries

In [ ]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

Assume you have a dataset with features (X) and target variable (y) Replace X and y with your actual feature and target data

In [ ]:
```
# Assume you have a dataset with features (X) and target variable (y)
# Replace X and y with your actual feature and target data

# 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)
```

Create a Decision Tree classifier , Train the classifier on the training data

In [ ]:
```
# Create a Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Make predictions on the test data

In [ ]:
```
# Make predictions on the test data
y_pred = clf.predict(X_test)
```

Evaluate the classifier, Display classification report

In [ ]:
```
# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.00

Classification Report:
              precision    recall  f1-score   support

        1001       0.00      0.00      0.00       1.0
        1002       0.00      0.00      0.00       0.0
        1010       0.00      0.00      0.00       1.0
        1011       0.00      0.00      0.00       0.0
        1035       0.00      0.00      0.00       0.0
        1048       0.00      0.00      0.00       1.0
        1055       0.00      0.00      0.00       1.0
        1064       0.00      0.00      0.00       1.0
        1071       0.00      0.00      0.00       0.0
        1080       0.00      0.00      0.00       1.0
        1135       0.00      0.00      0.00       0.0
        1139       0.00      0.00      0.00       1.0
        1162       0.00      0.00      0.00       0.0
        1191       0.00      0.00      0.00       0.0
        1196       0.00      0.00      0.00       1.0
        1198       0.00      0.00      0.00       1.0
        1210       0.00      0.00      0.00       0.0
        1219       0.00      0.00      0.00       0.0
        1223       0.00      0.00      0.00       1.0
        1229       0.00      0.00      0.00       2.0
        1257       0.00      0.00      0.00       1.0
        1265       0.00      0.00      0.00       0.0
        1274       0.00      0.00      0.00       1.0
        1281       0.00      0.00      0.00       0.0
        1282       0.00      0.00      0.00       0.0
        1294       0.00      0.00      0.00       1.0
        1316       0.00      0.00      0.00       1.0
        1332       0.00      0.00      0.00       0.0
        1400       0.00      0.00      0.00       1.0
        1416       0.00      0.00      0.00       0.0
        1422       0.00      0.00      0.00       1.0
        1429       0.00      0.00      0.00       1.0
        1449       0.00      0.00      0.00       1.0
        1455       0.00      0.00      0.00       0.0
        1456       0.00      0.00      0.00       1.0
        1461       0.00      0.00      0.00       0.0
        1475       0.00      0.00      0.00       1.0
        1487       0.00      0.00      0.00       1.0
        1513       0.00      0.00      0.00       0.0
```

1518	0.00	0.00	0.00	1.0
1527	0.00	0.00	0.00	0.0
1533	0.00	0.00	0.00	0.0
1554	0.00	0.00	0.00	0.0
1555	0.00	0.00	0.00	0.0
1572	0.00	0.00	0.00	1.0
1592	0.00	0.00	0.00	1.0
1593	0.00	0.00	0.00	1.0
1598	0.00	0.00	0.00	1.0
1599	0.00	0.00	0.00	1.0
1608	0.00	0.00	0.00	0.0
1653	0.00	0.00	0.00	0.0
1667	0.00	0.00	0.00	0.0
1677	0.00	0.00	0.00	0.0
1705	0.00	0.00	0.00	1.0
1717	0.00	0.00	0.00	1.0
1734	0.00	0.00	0.00	1.0
1755	0.00	0.00	0.00	0.0
1793	0.00	0.00	0.00	1.0
1804	0.00	0.00	0.00	1.0
1807	0.00	0.00	0.00	0.0
1817	0.00	0.00	0.00	0.0
1833	0.00	0.00	0.00	1.0
1835	0.00	0.00	0.00	1.0
1846	0.00	0.00	0.00	0.0
1849	0.00	0.00	0.00	1.0
1855	0.00	0.00	0.00	0.0
1860	0.00	0.00	0.00	0.0
1887	0.00	0.00	0.00	0.0
1888	0.00	0.00	0.00	0.0
1894	0.00	0.00	0.00	0.0
1898	0.00	0.00	0.00	0.0
1902	0.00	0.00	0.00	1.0
1909	0.00	0.00	0.00	1.0
1926	0.00	0.00	0.00	0.0
1928	0.00	0.00	0.00	1.0
1939	0.00	0.00	0.00	1.0
1944	0.00	0.00	0.00	1.0
1952	0.00	0.00	0.00	1.0
1970	0.00	0.00	0.00	0.0
1972	0.00	0.00	0.00	0.0
1979	0.00	0.00	0.00	1.0
1981	0.00	0.00	0.00	1.0
1988	0.00	0.00	0.00	0.0

1991	0.00	0.00	0.00	1.0
1992	0.00	0.00	0.00	0.0
2004	0.00	0.00	0.00	0.0
2015	0.00	0.00	0.00	1.0
2024	0.00	0.00	0.00	1.0
2049	0.00	0.00	0.00	1.0
2058	0.00	0.00	0.00	1.0
2079	0.00	0.00	0.00	1.0
2099	0.00	0.00	0.00	0.0
2100	0.00	0.00	0.00	0.0
2101	0.00	0.00	0.00	1.0
2109	0.00	0.00	0.00	0.0
2110	0.00	0.00	0.00	1.0
2126	0.00	0.00	0.00	1.0
2132	0.00	0.00	0.00	1.0
2140	0.00	0.00	0.00	1.0
2147	0.00	0.00	0.00	1.0
2151	0.00	0.00	0.00	1.0
2155	0.00	0.00	0.00	0.0
2156	0.00	0.00	0.00	1.0
2182	0.00	0.00	0.00	0.0
2185	0.00	0.00	0.00	1.0
2209	0.00	0.00	0.00	0.0
2254	0.00	0.00	0.00	1.0
2262	0.00	0.00	0.00	1.0
2273	0.00	0.00	0.00	1.0
2287	0.00	0.00	0.00	1.0
2319	0.00	0.00	0.00	0.0
2334	0.00	0.00	0.00	1.0
2345	0.00	0.00	0.00	0.0
2353	0.00	0.00	0.00	1.0
2362	0.00	0.00	0.00	1.0
2396	0.00	0.00	0.00	1.0
2398	0.00	0.00	0.00	1.0
2402	0.00	0.00	0.00	0.0
2405	0.00	0.00	0.00	1.0
2463	0.00	0.00	0.00	1.0
2468	0.00	0.00	0.00	1.0
2490	0.00	0.00	0.00	1.0
2516	0.00	0.00	0.00	0.0
2528	0.00	0.00	0.00	0.0
2541	0.00	0.00	0.00	0.0
2549	0.00	0.00	0.00	0.0
2554	0.00	0.00	0.00	1.0

2559	0.00	0.00	0.00	0.0
2561	0.00	0.00	0.00	1.0
2564	0.00	0.00	0.00	1.0
2571	0.00	0.00	0.00	0.0
2591	0.00	0.00	0.00	0.0
2593	0.00	0.00	0.00	0.0
2626	0.00	0.00	0.00	1.0
2636	0.00	0.00	0.00	0.0
2648	0.00	0.00	0.00	1.0
2691	0.00	0.00	0.00	1.0
2693	0.00	0.00	0.00	1.0
2703	0.00	0.00	0.00	0.0
2734	0.00	0.00	0.00	1.0
2798	0.00	0.00	0.00	1.0
2803	0.00	0.00	0.00	0.0
2809	0.00	0.00	0.00	0.0
2822	0.00	0.00	0.00	1.0
2824	0.00	0.00	0.00	0.0
2831	0.00	0.00	0.00	1.0
2837	0.00	0.00	0.00	0.0
2850	0.00	0.00	0.00	1.0
2859	0.00	0.00	0.00	0.0
2862	0.00	0.00	0.00	1.0
2870	0.00	0.00	0.00	1.0
2871	0.00	0.00	0.00	0.0
2888	0.00	0.00	0.00	1.0
2890	0.00	0.00	0.00	0.0
2898	0.00	0.00	0.00	1.0
2900	0.00	0.00	0.00	0.0
2904	0.00	0.00	0.00	1.0
2906	0.00	0.00	0.00	1.0
2910	0.00	0.00	0.00	1.0
2912	0.00	0.00	0.00	1.0
2928	0.00	0.00	0.00	0.0
2929	0.00	0.00	0.00	0.0
2938	0.00	0.00	0.00	1.0
2940	0.00	0.00	0.00	1.0
2951	0.00	0.00	0.00	0.0
2957	0.00	0.00	0.00	1.0
2958	0.00	0.00	0.00	0.0
2959	0.00	0.00	0.00	0.0
2961	0.00	0.00	0.00	1.0
2978	0.00	0.00	0.00	0.0
2997	0.00	0.00	0.00	1.0

2999	0.00	0.00	0.00	0.0
3004	0.00	0.00	0.00	1.0
3035	0.00	0.00	0.00	1.0
3048	0.00	0.00	0.00	1.0
3059	0.00	0.00	0.00	1.0
3062	0.00	0.00	0.00	0.0
3119	0.00	0.00	0.00	1.0
3125	0.00	0.00	0.00	0.0
3127	0.00	0.00	0.00	0.0
3143	0.00	0.00	0.00	0.0
3153	0.00	0.00	0.00	0.0
3162	0.00	0.00	0.00	0.0
3206	0.00	0.00	0.00	1.0
3216	0.00	0.00	0.00	0.0
3244	0.00	0.00	0.00	1.0
3252	0.00	0.00	0.00	0.0
3292	0.00	0.00	0.00	1.0
3299	0.00	0.00	0.00	1.0
3307	0.00	0.00	0.00	0.0
3311	0.00	0.00	0.00	1.0
3317	0.00	0.00	0.00	0.0
3318	0.00	0.00	0.00	1.0
3319	0.00	0.00	0.00	1.0
3323	0.00	0.00	0.00	0.0
3326	0.00	0.00	0.00	0.0
3331	0.00	0.00	0.00	1.0
3337	0.00	0.00	0.00	1.0
3338	0.00	0.00	0.00	0.0
3344	0.00	0.00	0.00	1.0
3354	0.00	0.00	0.00	1.0
3364	0.00	0.00	0.00	1.0
3392	0.00	0.00	0.00	1.0
3434	0.00	0.00	0.00	1.0
3438	0.00	0.00	0.00	1.0
3448	0.00	0.00	0.00	0.0
3457	0.00	0.00	0.00	1.0
3466	0.00	0.00	0.00	1.0
3508	0.00	0.00	0.00	1.0
3519	0.00	0.00	0.00	1.0
3540	0.00	0.00	0.00	0.0
3544	0.00	0.00	0.00	0.0
3560	0.00	0.00	0.00	0.0
3561	0.00	0.00	0.00	0.0
3563	0.00	0.00	0.00	1.0

3564	0.00	0.00	0.00	1.0
3569	0.00	0.00	0.00	1.0
3573	0.00	0.00	0.00	1.0
3578	0.00	0.00	0.00	1.0
3585	0.00	0.00	0.00	0.0
3597	0.00	0.00	0.00	0.0
3606	0.00	0.00	0.00	1.0
3612	0.00	0.00	0.00	1.0
3613	0.00	0.00	0.00	0.0
3620	0.00	0.00	0.00	1.0
3624	0.00	0.00	0.00	1.0
3626	0.00	0.00	0.00	0.0
3628	0.00	0.00	0.00	0.0
3630	0.00	0.00	0.00	1.0
3631	0.00	0.00	0.00	1.0
3647	0.00	0.00	0.00	0.0
3651	0.00	0.00	0.00	1.0
3652	0.00	0.00	0.00	1.0
3653	0.00	0.00	0.00	1.0
3656	0.00	0.00	0.00	1.0
3659	0.00	0.00	0.00	1.0
3676	0.00	0.00	0.00	1.0
3678	0.00	0.00	0.00	0.0
3681	0.00	0.00	0.00	1.0
3701	0.00	0.00	0.00	1.0
3732	0.00	0.00	0.00	1.0
3733	0.00	0.00	0.00	0.0
3736	0.00	0.00	0.00	1.0
3753	0.00	0.00	0.00	0.0
3761	0.00	0.00	0.00	1.0
3767	0.00	0.00	0.00	0.0
3782	0.00	0.00	0.00	0.0
3804	0.00	0.00	0.00	1.0
3806	0.00	0.00	0.00	0.0
3809	0.00	0.00	0.00	1.0
3816	0.00	0.00	0.00	0.0
3820	0.00	0.00	0.00	0.0
3832	0.00	0.00	0.00	0.0
3835	0.00	0.00	0.00	1.0
3846	0.00	0.00	0.00	0.0
3849	0.00	0.00	0.00	1.0
3859	0.00	0.00	0.00	0.0
3874	0.00	0.00	0.00	0.0
3886	0.00	0.00	0.00	1.0

3894	0.00	0.00	0.00	0.0
3904	0.00	0.00	0.00	1.0
3907	0.00	0.00	0.00	0.0
3910	0.00	0.00	0.00	0.0
3942	0.00	0.00	0.00	0.0
3953	0.00	0.00	0.00	1.0
3979	0.00	0.00	0.00	1.0
3983	0.00	0.00	0.00	1.0
4003	0.00	0.00	0.00	0.0
4007	0.00	0.00	0.00	1.0
4028	0.00	0.00	0.00	0.0
4057	0.00	0.00	0.00	0.0
4058	0.00	0.00	0.00	0.0
4076	0.00	0.00	0.00	0.0
4084	0.00	0.00	0.00	1.0
4094	0.00	0.00	0.00	1.0
4118	0.00	0.00	0.00	1.0
4119	0.00	0.00	0.00	0.0
4123	0.00	0.00	0.00	0.0
4124	0.00	0.00	0.00	0.0
4142	0.00	0.00	0.00	0.0
4149	0.00	0.00	0.00	0.0
4165	0.00	0.00	0.00	0.0
4168	0.00	0.00	0.00	0.0
4179	0.00	0.00	0.00	1.0
4182	0.00	0.00	0.00	1.0
4187	0.00	0.00	0.00	0.0
4192	0.00	0.00	0.00	1.0
4203	0.00	0.00	0.00	1.0
4222	0.00	0.00	0.00	1.0
4223	0.00	0.00	0.00	0.0
4238	0.00	0.00	0.00	1.0
4252	0.00	0.00	0.00	0.0
4258	0.00	0.00	0.00	0.0
4262	0.00	0.00	0.00	1.0
4271	0.00	0.00	0.00	1.0
4284	0.00	0.00	0.00	2.0
4287	0.00	0.00	0.00	0.0
4305	0.00	0.00	0.00	0.0
4317	0.00	0.00	0.00	1.0
4323	0.00	0.00	0.00	1.0
4360	0.00	0.00	0.00	0.0
4365	0.00	0.00	0.00	1.0
4366	0.00	0.00	0.00	1.0

4379	0.00	0.00	0.00	0.0
4389	0.00	0.00	0.00	1.0
4395	0.00	0.00	0.00	1.0
4415	0.00	0.00	0.00	0.0
4420	0.00	0.00	0.00	0.0
4429	0.00	0.00	0.00	1.0
4434	0.00	0.00	0.00	1.0
4444	0.00	0.00	0.00	0.0
4449	0.00	0.00	0.00	1.0
4468	0.00	0.00	0.00	0.0
4477	0.00	0.00	0.00	0.0
4489	0.00	0.00	0.00	0.0
4492	0.00	0.00	0.00	1.0
4493	0.00	0.00	0.00	0.0
4498	0.00	0.00	0.00	0.0
4501	0.00	0.00	0.00	0.0
4505	0.00	0.00	0.00	1.0
4514	0.00	0.00	0.00	0.0
4517	0.00	0.00	0.00	0.0
4520	0.00	0.00	0.00	0.0
4524	0.00	0.00	0.00	0.0
4527	0.00	0.00	0.00	1.0
4535	0.00	0.00	0.00	0.0
4543	0.00	0.00	0.00	1.0
4554	0.00	0.00	0.00	1.0
4561	0.00	0.00	0.00	1.0
4573	0.00	0.00	0.00	0.0
4595	0.00	0.00	0.00	1.0
4602	0.00	0.00	0.00	0.0
4610	0.00	0.00	0.00	0.0
4627	0.00	0.00	0.00	1.0
4646	0.00	0.00	0.00	1.0
4651	0.00	0.00	0.00	1.0
4653	0.00	0.00	0.00	1.0
4654	0.00	0.00	0.00	1.0
4663	0.00	0.00	0.00	0.0
4670	0.00	0.00	0.00	1.0
4677	0.00	0.00	0.00	0.0
4681	0.00	0.00	0.00	1.0
4707	0.00	0.00	0.00	1.0
4713	0.00	0.00	0.00	1.0
4714	0.00	0.00	0.00	0.0
4719	0.00	0.00	0.00	1.0
4755	0.00	0.00	0.00	1.0

```
        4758       0.00      0.00      0.00       1.0
        4765       0.00      0.00      0.00       1.0
        4775       0.00      0.00      0.00       0.0
        4800       0.00      0.00      0.00       1.0
        4821       0.00      0.00      0.00       1.0
        4824       0.00      0.00      0.00       0.0
        4845       0.00      0.00      0.00       0.0
        4850       0.00      0.00      0.00       0.0
        4866       0.00      0.00      0.00       0.0
        4876       0.00      0.00      0.00       0.0
        4878       0.00      0.00      0.00       1.0
        4886       0.00      0.00      0.00       0.0
        4890       0.00      0.00      0.00       0.0
        4911       0.00      0.00      0.00       0.0
        4944       0.00      0.00      0.00       1.0
        4949       0.00      0.00      0.00       0.0
        4979       0.00      0.00      0.00       0.0
        4989       0.00      0.00      0.00       0.0
        4992       0.00      0.00      0.00       0.0
        4997       0.00      0.00      0.00       0.0

    accuracy                           0.00     200.0
   macro avg       0.00      0.00      0.00     200.0
weighted avg       0.00      0.00      0.00     200.0
```

```
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-s
core are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall and F-scor
e are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-s
core are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall and F-scor
e are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-s
core are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall and F-scor
e are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

---

3. Model Selection and Evaluation using Logistic Regression:

Scikit-learn provides tools for model selection and evaluation, including techniques for cross-validation, hyperparameter tuning, and model evaluation metrics. Here's an example of performing cross-validation and evaluating a model's performance:

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np

data = pd.read_csv(r'data/titanic.csv')
data.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [ ]:

```
data.shape
#missing values
data.isnull().sum()

data['Age'].fillna(data['Age'].mean(),inplace=True)
data.isnull().sum()

data['Fare'].fillna(data['Fare'].mean(),inplace=True)
data.isnull().sum()
```

Out[ ]:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

In [ ]:

```
data.drop("Cabin", axis=1, inplace=True)
print(data.shape)
data.isnull().sum()
```

```
(418, 11)
```

Out[ ]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [ ]:
```
data.dtypes
```

Out[ ]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Embarked        object
dtype: object
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

colname=[]
for x in data.columns:
    if data[x].dtypes=='object':
        colname.append(x)

le=LabelEncoder()

for x in colname:
    data[x]=le.fit_transform(data[x])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print('Feature', x)
    print('mapping', le_name_mapping)

data.head()
```

```
Feature Name
mapping {'Abbott, Master. Eugene Joseph': 0, 'Abelseth, Miss. Karen Marie': 1, 'Abelseth, Mr. Olaus Jorgensen': 2, 'Abrahamsson,
Mr. Abraham August Johannes': 3, 'Abrahim, Mrs. Joseph (Sophie Halaut Easu)': 4, 'Aks, Master. Philip Frank': 5, 'Aldworth, Mr.
Charles Augustus': 6, 'Allison, Mr. Hudson Joshua Creighton': 7, 'Andersen, Mr. Albert Karvin': 8, 'Andersson, Miss. Ida Augusta
Margareta': 9, 'Andersson, Mr. Johan Samuel': 10, 'Andrew, Mr. Frank Thomas': 11, 'Angheloff, Mr. Minko': 12, 'Angle, Mr. Willia
m A': 13, 'Aronsson, Mr. Ernst Axel Algot': 14, 'Ashby, Mr. John': 15, 'Asplund, Master. Carl Edgar': 16, 'Asplund, Master. Fili
p Oscar': 17, 'Asplund, Mr. Carl Oscar Vilhelm Gustafsson': 18, 'Asplund, Mr. Johan Charles': 19, 'Assaf Khalil, Mrs. Mariana (M
iriam")"': 20, 'Assaf, Mr. Gerios': 21, 'Assam, Mr. Ali': 22, 'Astor, Col. John Jacob': 23, 'Baccos, Mr. Raffull': 24, 'Badman,
Miss. Emily Louisa': 25, 'Baimbrigge, Mr. Charles Robert': 26, 'Barry, Miss. Julia': 27, 'Beattie, Mr. Thomson': 28, 'Beauchamp,
Mr. Henry James': 29, 'Becker, Miss. Ruth Elizabeth': 30, 'Becker, Mrs. Allen Oliver (Nellie E Baumgardner)': 31, 'Bentham, Mis
s. Lilian W': 32, 'Betros, Master. Seman': 33, 'Bird, Miss. Ellen': 34, 'Birnbaum, Mr. Jakob': 35, 'Bjorklund, Mr. Ernst Herber
t': 36, 'Bonnell, Miss. Caroline': 37, 'Borebank, Mr. John James': 38, 'Botsford, Mr. William Hull': 39, 'Boulos, Master. Akar':
40, 'Bowen, Miss. Grace Scott': 41, 'Bowenur, Mr. Solomon': 42, 'Bradley, Miss. Bridget Delia': 43, 'Brady, Mr. John Bertram': 4
4, 'Braf, Miss. Elin Ester Maria': 45, 'Brandeis, Mr. Emil': 46, 'Brobeck, Mr. Karl Rudolf': 47, 'Brown, Miss. Edith Eileen': 4
8, 'Brown, Mrs. John Murray (Caroline Lane Lamson)': 49, 'Bryhl, Miss. Dagmar Jenny Ingeborg ': 50, 'Buckley, Miss. Katherine':
51, 'Buckley, Mr. Daniel': 52, 'Bucknell, Mrs. William Robert (Emma Eliza Ward)': 53, 'Burns, Miss. Mary Delia': 54, 'Cacic, Mis
s. Manda': 55, 'Cacic, Mr. Jego Grga': 56, 'Caldwell, Mr. Albert Francis': 57, 'Canavan, Mr. Patrick': 58, 'Candee, Mrs. Edward
(Helen Churchill Hungerford)': 59, 'Caram, Mr. Joseph': 60, 'Cardeza, Mrs. James Warburton Martinez (Charlotte Wardle Drake)': 6
1, 'Carlsson, Mr. Carl Robert': 62, 'Carr, Miss. Jeannie': 63, 'Carrau, Mr. Jose Pedro': 64, 'Carver, Mr. Alfred John': 65, 'Cas
e, Mr. Howard Brown': 66, 'Cassebeer, Mrs. Henry Arthur Jr (Eleanor Genevieve Fosdick)': 67, 'Cavendish, Mrs. Tyrell William (Ju
lia Florence Siegel)': 68, 'Chaffee, Mrs. Herbert Fuller (Carrie Constance Toogood)': 69, 'Chapman, Mrs. John Henry (Sara Elizab
eth Lawry)': 70, 'Chaudanson, Miss. Victorine': 71, 'Chevre, Mr. Paul Romaine': 72, 'Chisholm, Mr. Roderick Robert Crispin': 73,
'Christy, Mrs. (Alice Frances)': 74, 'Chronopoulos, Mr. Demetrios': 75, 'Clark, Mr. Walter Miller': 76, 'Clark, Mrs. Walter Mill
er (Virginia McDowell)': 77, 'Clarke, Mr. Charles Valentine': 78, 'Colbert, Mr. Patrick': 79, 'Collett, Mr. Sidney C Stuart': 8
0, 'Compton, Mr. Alexander Taylor Jr': 81, 'Compton, Mrs. Alexander Taylor (Mary Eliza Ingersoll)': 82, 'Conlon, Mr. Thomas Henr
y': 83, 'Connolly, Miss. Kate': 84, 'Cook, Mrs. (Selena Rogers)': 85, 'Cor, Mr. Bartol': 86, 'Cor, Mr. Ivan': 87, 'Corbett, Mrs.
Walter H (Irene Colvin)': 88, 'Corey, Mrs. Percy C (Mary Phyllis Elizabeth Miller)': 89, 'Cornell, Mrs. Robert Clifford (Malvina
Helen Lamson)': 90, 'Cotterill, Mr. Henry Harry""': 91, 'Coutts, Mrs. William (Winnie Minnie" Treanor)"': 92, 'Crafton, Mr. John
Bertram': 93, 'Cribb, Miss. Laura Alice': 94, 'Crosby, Mrs. Edward Gifford (Catherine Elizabeth Halstead)': 95, 'Cumings, Mr. Jo
hn Bradley': 96, 'Daher, Mr. Shedid': 97, 'Daly, Miss. Margaret Marcella Maggie""': 98, 'Danbom, Master. Gilbert Sigvard Emanue
l': 99, 'Daniels, Miss. Sarah': 100, 'Davidson, Mrs. Thornton (Orian Hays)': 101, 'Davies, Mr. Evan': 102, 'Davies, Mr. John Sam
uel': 103, 'Davies, Mr. Joseph': 104, 'Davies, Mrs. John Morgan (Elizabeth Agnes Mary White) ': 105, 'Davison, Mr. Thomas Henr
y': 106, 'Deacon, Mr. Percy William': 107, 'Dean, Miss. Elizabeth Gladys Millvina""': 108, 'Dean, Mrs. Bertram (Eva Georgetta Li
ght)': 109, 'Delalic, Mr. Redjo': 110, 'Demetri, Mr. Marinko': 111, 'Denbury, Mr. Herbert': 112, 'Dennis, Mr. William': 113, 'Di
bden, Mr. William': 114, 'Dika, Mr. Mirko': 115, 'Dintcheff, Mr. Valtcho': 116, 'Dodge, Dr. Washington': 117, 'Dodge, Mrs. Washi
ngton (Ruth Vidaver)': 118, 'Douglas, Mrs. Frederick Charles (Mary Helene Baxter)': 119, 'Douglas, Mrs. Walter Donald (Mahala Du
tton)': 120, 'Doyle, Miss. Elizabeth': 121, 'Drapkin, Miss. Jennie': 122, 'Drew, Master. Marshall Brines': 123, 'Drew, Mr. James
Vivian': 124, 'Dulles, Mr. William Crothers': 125, 'Duquemin, Mr. Joseph': 126, 'Duran y More, Miss. Florentina': 127, 'Dyker, M
r. Adolf Fredrik': 128, 'Dyker, Mrs. Adolf Fredrik (Anna Elisabeth Judith Andersson)': 129, 'Earnshaw, Mrs. Boulton (Olive Potte
r)': 130, 'Elias, Mr. Joseph': 131, 'Enander, Mr. Ingvar': 132, 'Evans, Miss. Edith Corse': 133, 'Everett, Mr. Thomas James': 13
4, 'Faunthorpe, Mr. Harry': 135, 'Fillbrook, Mr. Joseph Charles': 136, 'Finoli, Mr. Luigi': 137, 'Flegenheim, Mrs. Alfred (Antoi
nette)': 138, 'Fleming, Miss. Honora': 139, 'Foley, Mr. Joseph': 140, 'Foley, Mr. William': 141, 'Ford, Mr. Arthur': 142, 'Ford,
Mr. Edward Watson': 143, 'Fortune, Miss. Ethel Flora': 144, 'Fortune, Mrs. Mark (Mary McDougald)': 145, 'Fox, Mr. Patrick': 146,
'Franklin, Mr. Charles (Charles Fardon)': 147, 'Franklin, Mr. Thomas Parham': 148, 'Frauenthal, Mr. Isaac Gerald': 149, 'Frolich
```

er-Stehli, Mrs. Maxmillian (Margaretha Emerentia Stehli)': 150, 'Gale, Mr. Harry': 151, 'Geiger, Miss. Amalie': 152, 'Gibson, Miss. Dorothy Winifred': 153, 'Gibson, Mrs. Leonard (Pauline C Boeson)': 154, 'Gilbert, Mr. William': 155, 'Giles, Mr. Edgar': 156, 'Giles, Mr. Ralph': 157, 'Goldsmith, Mr. Nathan': 158, 'Goodwin, Miss. Jessie Allis': 159, 'Goodwin, Mr. Charles Frederick': 160, 'Gracie, Col. Archibald IV': 161, 'Greenfield, Mrs. Leo David (Blanche Strouse)': 162, 'Guest, Mr. Robert': 163, 'Hagardon, Miss. Kate': 164, 'Hansen, Mrs. Claus Peter (Jennie L Howard)': 165, 'Harbeck, Mr. William H': 166, 'Harder, Mrs. George Achilles (Dorothy Annan)': 167, 'Hays, Mr. Charles Melville': 168, 'Head, Mr. Christopher': 169, 'Hee, Mr. Ling': 170, 'Hellstrom, Miss. Hilda Maria': 171, 'Henriksson, Miss. Jenny Lovisa': 172, 'Herman, Miss. Kate': 173, 'Herman, Mr. Samuel': 174, 'Hilliard, Mr. Herbert Henry': 175, 'Hiltunen, Miss. Marta': 176, 'Hipkins, Mr. William Edward': 177, 'Hirvonen, Mrs. Alexander (Helga E Lindqvist)': 178, 'Hocking, Miss. Ellen Nellie""': 179, 'Hocking, Mr. Samuel James Metcalfe': 180, 'Hold, Mrs. Stephen (Annie Margaret Hill)': 181, 'Holthen, Mr. Johan Martin': 182, 'Howard, Miss. May Elizabeth': 183, 'Howard, Mr. Benjamin': 184, 'Howard, Mrs. Benjamin (Ellen Truelove Arman)': 185, 'Hyman, Mr. Abraham': 186, 'Ilieff, Mr. Ylio': 187, 'Ilmakangas, Miss. Ida Livija': 188, 'Ismay, Mr. Joseph Bruce': 189, 'Jefferys, Mr. Clifford Thomas': 190, 'Jefferys, Mr. Ernest Wilfred': 191, 'Johansson Palmquist, Mr. Oskar Leander': 192, 'Johansson, Mr. Nils': 193, 'Johnston, Master. William Arthur Willie""': 194, 'Johnston, Mrs. Andrew G (Elizabeth Lily" Watson)"': 195, 'Jones, Mr. Charles Cresson': 196, 'Jonsson, Mr. Nils Hilding': 197, 'Julian, Mr. Henry Forbes': 198, 'Karlsson, Mr. Einar Gervasius': 199, 'Karlsson, Mr. Julius Konrad Eugen': 200, 'Karnes, Mrs. J Frank (Claire Bennett)': 201, 'Karun, Mr. Franz': 202, 'Katavelas, Mr. Vassilios (Catavelas Vassilios")"': 203, 'Keane, Mr. Daniel': 204, 'Keeping, Mr. Edwin': 205, 'Kelly, Mr. James': 206, 'Kennedy, Mr. John': 207, 'Kenyon, Mr. Frederick R': 208, 'Khalil, Mr. Betros': 209, 'Khalil, Mrs. Betros (Zahie Maria" Elias)"': 210, 'Kiernan, Mr. John': 211, 'Kimball, Mrs. Edwin Nelson Jr (Gertrude Parsons)': 212, 'Kink, Miss. Maria': 213, 'Kink-Heilmann, Mr. Anton': 214, 'Kink-Heilmann, Mrs. Anton (Luise Heilmann)': 215, 'Klasen, Miss. Gertrud Emilia': 216, 'Klasen, Mrs. (Hulda Kristina Eugenia Lofqvist)': 217, 'Krekorian, Mr. Neshan': 218, 'Kreuchen, Miss. Emilie': 219, 'Lahtinen, Rev. William': 220, 'Lamb, Mr. John Joseph': 221, 'Lane, Mr. Patrick': 222, 'Laroche, Miss. Louise': 223, 'Larsson-Rondberg, Mr. Edvard A': 224, 'Lefebre, Mrs. Frank (Frances)': 225, 'Lennon, Miss. Mary': 226, 'Lindeberg-Lind, Mr. Erik Gustaf (Mr Edward Lingrey")"': 227, 'Lindell, Mrs. Edvard Bengtsson (Elin Gerda Persson)': 228, 'Lindstrom, Mrs. Carl Johan (Sigrid Posse)': 229, 'Linehan, Mr. Michael': 230, 'Lines, Mrs. Ernest H (Elizabeth Lindsey James)': 231, 'Lingane, Mr. John': 232, 'Lithman, Mr. Simon': 233, 'Lockyer, Mr. Edward': 234, 'Loring, Mr. Joseph Holland': 235, 'Louch, Mr. Charles Alexander': 236, 'Lundin, Miss. Olga Elida': 237, 'Lundstrom, Mr. Thure Edvin': 238, 'Lyntakoff, Mr. Stanko': 239, 'MacKay, Mr. George William': 240, 'Maguire, Mr. John Edward': 241, 'Mahon, Miss. Bridget Delia': 242, 'Mahon, Mr. John': 243, 'Makinen, Mr. Kalle Edvard': 244, 'Malachard, Mr. Noel': 245, 'Mallet, Mrs. Albert (Antoinette Magnin)': 246, 'Mangiavacchi, Mr. Serafino Emilio': 247, 'Mardirosian, Mr. Sarkis': 248, 'Marvin, Mrs. Daniel Warner (Mary Graham Carmichael Farquarson)': 249, 'Matinoff, Mr. Nicola': 250, 'Maybery, Mr. Frank Hubert': 251, 'McCaffry, Mr. Thomas Francis': 252, 'McCarthy, Miss. Catherine Katie""': 253, 'McCoy, Miss. Alicia': 254, 'McCrae, Mr. Arthur Gordon': 255, 'McCrie, Mr. James Matthew': 256, 'McGowan, Miss. Katherine': 257, "McNamee, Mrs. Neal (Eileen O'Leary)": 258, 'McNeill, Miss. Bridget': 259, 'Midtsjo, Mr. Karl Albert': 260, 'Miles, Mr. Frank': 261, 'Minahan, Mrs. William Edward (Lillian E Thorpe)': 262, 'Minkoff, Mr. Lazar': 263, 'Mock, Mr. Philipp Edmund': 264, 'Moore, Mr. Clarence Bloomfield': 265, 'Moubarek, Mrs. George (Omine Amenia" Alexander)"': 266, 'Mulvihill, Miss. Bertha E': 267, 'Murphy, Miss. Nora': 268, 'Myles, Mr. Thomas Francis': 269, 'Nakid, Mrs. Said (Waika Mary" Mowad)"': 270, 'Nancarrow, Mr. William Henry': 271, 'Nasr, Mr. Mustafa': 272, 'Naughton, Miss. Hannah': 273, 'Nesson, Mr. Israel': 274, 'Nieminen, Miss. Manta Josefina': 275, 'Niklasson, Mr. Samuel': 276, 'Nilsson, Miss. Berta Olivia': 277, 'Nilsson, Mr. August Ferdinand': 278, 'Nourney, Mr. Alfred (Baron von Drachstedt")"': 279, "O'Connor, Mr. Patrick": 280, "O'Donoghue, Ms. Bridget": 281, "O'Keefe, Mr. Patrick": 282, 'Oliva y Ocana, Dona. Fermina': 283, 'Olsen, Master. Artur Karl': 284, 'Olsson, Mr. Oscar Wilhelm': 285, 'Omont, Mr. Alfred Fernand': 286, 'Oreskovic, Miss. Jelka': 287, 'Ostby, Miss. Helene Ragnhild': 288, 'Ovies y Rodriguez, Mr. Servando': 289, 'Oxenham, Mr. Percy Thomas': 290, 'Pallas y Castello, Mr. Emilio': 291, 'Palsson, Master. Paul Folke': 292, 'Parker, Mr. Clifford Richard': 293, 'Payne, Mr. Vivian Ponsonby': 294, 'Peacock, Master. Alfred Edward': 295, 'Peacock, Miss. Treasteall': 296, 'Peacock, Mrs. Benjamin (Edith Nile)': 297, 'Pearce, Mr. Ernest': 298, 'Pedersen, Mr. Olaf': 299, 'Peltomaki, Mr. Nikolai Johannes': 300, 'Peruschitz, Rev. Joseph Maria': 301, 'Peter, Master. Michael J': 302, 'Petersen, Mr. Marius': 303, 'Phillips, Miss. Alice Frances Louisa': 30

4, 'Phillips, Mr. Escott Robert': 305, 'Pokrnic, Mr. Mate': 306, 'Pokrnic, Mr. Tome': 307, 'Portaluppi, Mr. Emilio Ilario Giuseppe': 308, 'Pulbaum, Mr. Franz': 309, 'Quick, Miss. Winifred Vera': 310, 'Rasmussen, Mrs. (Lena Jacobsen Solvang)': 311, 'Reynolds, Mr. Harold J': 312, 'Rheims, Mr. George Alexander Lucien': 313, 'Rice, Master. Albert': 314, 'Riihivouri, Miss. Susanna Juhantytar Sanni""': 315, 'Riordan, Miss. Johanna Hannah""': 316, 'Risien, Mrs. Samuel (Emma)': 317, 'Robins, Mr. Alexander A': 318, 'Rogers, Mr. Reginald Harry': 319, 'Rosblom, Miss. Salli Helena': 320, 'Rosenbaum, Miss. Edith Louise': 321, 'Rosenshine, Mr. George (Mr George Thorne")"': 322, 'Roth, Miss. Sarah A': 323, 'Rothschild, Mr. Martin': 324, 'Rowe, Mr. Alfred G': 325, 'Ryan, Mr. Edward': 326, 'Ryerson, Master. John Borie': 327, 'Ryerson, Mr. Arthur Larned': 328, 'Ryerson, Mrs. Arthur Larned (Emily Maria Borie)': 329, 'Saade, Mr. Jean Nassr': 330, 'Sadowitz, Mr. Harry': 331, 'Saether, Mr. Simon Sivertsen': 332, 'Sage, Master. William Henry': 333, 'Sage, Miss. Ada': 334, 'Sage, Mr. John George': 335, 'Sage, Mrs. John (Annie Bullen)': 336, 'Salander, Mr. Karl Johan': 337, 'Salomon, Mr. Abraham L': 338, 'Samaan, Mr. Elias': 339, 'Samaan, Mr. Hanna': 340, 'Sandstrom, Miss. Beatrice Irene': 341, 'Sap, Mr. Julius': 342, 'Schabert, Mrs. Paul (Emma Mock)': 343, 'Schmidt, Mr. August': 344, 'Shaughnessy, Mr. Patrick': 345, 'Shine, Miss. Ellen Natalia': 346, 'Sincock, Miss. Maude': 347, 'Smith, Mr. Lucien Philip': 348, 'Smith, Mrs. Lucien Philip (Mary Eloise Hughes)': 349, 'Smyth, Miss. Julia': 350, 'Snyder, Mr. John Pillsbury': 351, 'Snyder, Mrs. John Pillsbury (Nelle Stevenson)': 352, 'Spector, Mr. Woolf': 353, 'Spedden, Master. Robert Douglas': 354, 'Spedden, Mr. Frederic Oakley': 355, 'Spencer, Mr. William Augustus': 356, 'Spinner, Mr. Henry John': 357, 'Stanton, Mr. Samuel Ward': 358, 'Stengel, Mr. Charles Emil Henry': 359, 'Stengel, Mrs. Charles Emil Henry (Annie May Morris)': 360, 'Stokes, Mr. Philip Joseph': 361, 'Storey, Mr. Thomas': 362, 'Straus, Mr. Isidor': 363, 'Straus, Mrs. Isidor (Rosalie Ida Blun)': 364, 'Strilic, Mr. Ivan': 365, 'Svensson, Mr. Johan Cervin': 366, 'Swane, Mr. George': 367, 'Sweet, Mr. George Frederick': 368, 'Tenglin, Mr. Gunnar Isidor': 369, 'Thomas, Mr. Charles P': 370, 'Thomas, Mr. John': 371, 'Thomas, Mr. Tannous': 372, 'Thomas, Mrs. Alexander (Thamine Thelma")"': 373, 'Thomson, Mr. Alexander Morrison': 374, 'Torfa, Mr. Assad': 375, 'Touma, Master. Georges Youssef': 376, 'Touma, Miss. Maria Youssef': 377, 'Tucker, Mr. Gilbert Milligan Jr': 378, 'Vander Planke, Mr. Julius': 379, 'Vartanian, Mr. David': 380, 'Veal, Mr. James': 381, 'Vendel, Mr. Olof Edvin': 382, 'Walcroft, Miss. Nellie': 383, 'Ware, Mr. Frederick': 384, 'Ware, Mr. John James': 385, 'Ware, Mr. William Jeffery': 386, 'Ware, Mrs. John James (Florence Louise Long)': 387, 'Warren, Mr. Charles William': 388, 'Warren, Mr. Frank Manley': 389, 'Watt, Miss. Bertha J': 390, 'Weisz, Mr. Leopold': 391, 'Wells, Master. Ralph Lester': 392, 'Wells, Mrs. Arthur Henry (Addie" Dart Trevaskis)"': 393, 'Wenzel, Mr. Linhart': 394, 'West, Miss. Barbara J': 395, 'Whabee, Mrs. George Joseph (Shawneene Abi-Saab)': 396, 'Wheeler, Mr. Edwin Frederick""': 397, 'White, Mrs. John Stuart (Ella Holmes)': 398, 'Wick, Mr. George Dennick': 399, 'Widener, Mr. George Dunton': 400, 'Widener, Mrs. George Dunton (Eleanor Elkins)': 401, 'Wiklund, Mr. Karl Johan': 402, 'Wilkes, Mrs. James (Ellen Needs)': 403, 'Willard, Miss. Constance': 404, 'Willer, Mr. Aaron (Abi Weller")"': 405, 'Williams, Mr. Richard Norris II': 406, 'Wilson, Miss. Helen Alice': 407, 'Wirz, Mr. Albert': 408, 'Wittevrongel, Mr. Camille': 409, 'Wright, Miss. Marion': 410, 'Zakarian, Mr. Mapriededer': 411, 'Zakarian, Mr. Ortin': 412, 'de Brito, Mr. Jose Joaquim': 413, 'de Messemaeker, Mr. Guillaume Joseph': 414, 'del Carlo, Mrs. Sebastiano (Argenia Genovesi)': 415, 'van Billiard, Master. James William': 416, 'van Billiard, Master. Walter John': 417}
Feature Sex
mapping {'female': 0, 'male': 1}
Feature Ticket
mapping {'110469': 0, '110489': 1, '110813': 2, '111163': 3, '112051': 4, '112058': 5, '112377': 6, '112378': 7, '112901': 8, '113038': 9, '113044': 10, '113054': 11, '113059': 12, '113503': 13, '113509': 14, '113773': 15, '113778': 16, '113780': 17, '113781': 18, '113790': 19, '113791': 20, '113795': 21, '113796': 22, '113801': 23, '11753': 24, '11765': 25, '11767': 26, '11769': 27, '11770': 28, '11778': 29, '11813': 30, '1222': 31, '12749': 32, '13050': 33, '13236': 34, '13508': 35, '13567': 36, '13695': 37, '13905': 38, '1601': 39, '16966': 40, '17463': 41, '17464': 42, '17475': 43, '17765': 44, '17770': 45, '19877': 46, '19924': 47, '19928': 48, '19950': 49, '2003': 50, '211535': 51, '21228': 52, '21332': 53, '220844': 54, '220845': 55, '226875': 56, '228414': 57, '230136': 58, '233478': 59, '233734': 60, '235509': 61, '236853': 62, '236854': 63, '237216': 64, '237249': 65, '237393': 66, '237670': 67, '237734': 68, '237735': 69, '237789': 70, '239059': 71, '240261': 72, '240276': 73, '24065': 74, '24160': 75, '242963': 76, '244346': 77, '244358': 78, '244360': 79, '244368': 80, '248659': 81, '248726': 82, '248734': 83, '248738': 8

4, '248744': 85, '248746': 86, '250650': 87, '250651': 88, '2543': 89, '2621': 90, '2622': 91, '2625': 92, '26360': 93, '2650': 94, '2652': 95, '2653': 96, '2654': 97, '2655': 98, '2656': 99, '2657': 100, '2658': 101, '2660': 102, '2661': 103, '2662': 104, '2668': 105, '2670': 106, '26707': 107, '2673': 108, '2675': 109, '2676': 110, '2678': 111, '2679': 112, '2680': 113, '2681': 114, '2682': 115, '2684': 116, '2688': 117, '2689': 118, '2692': 119, '2696': 120, '2698': 121, '28004': 122, '28034': 123, '28133': 124, '28220': 125, '28221': 126, '28404': 127, '28664': 128, '28666': 129, '29103': 130, '29105': 131, '29107': 132, '2926': 133, '29750': 134, '3101266': 135, '3101295': 136, '3101297': 137, '3101298': 138, '315083': 139, '315085': 140, '315087': 141, '315091': 142, '315092': 143, '315095': 144, '315152': 145, '315153': 146, '315154': 147, '32302': 148, '329944': 149, '330844': 150, '330910': 151, '330911': 152, '330920': 153, '330924': 154, '330963': 155, '330968': 156, '330971': 157, '330972': 158, '334914': 159, '334915': 160, '335432': 161, '33638': 162, '3410': 163, '342441': 164, '342684': 165, '342712': 166, '343271': 167, '345498': 168, '345501': 169, '345572': 170, '345763': 171, '345768': 172, '345771': 173, '345775': 174, '3470': 175, '347065': 176, '347066': 177, '347070': 178, '347072': 179, '347075': 180, '347077': 181, '347079': 182, '347080': 183, '347086': 184, '347090': 185, '347091': 186, '347465': 187, '347467': 188, '347469': 189, '347471': 190, '348122': 191, '348125': 192, '349202': 193, '349211': 194, '349220': 195, '349226': 196, '349229': 197, '349230': 198, '349232': 199, '349235': 200, '349238': 201, '349250': 202, '349255': 203, '349256': 204, '349909': 205, '349910': 206, '349911': 207, '350026': 208, '350033': 209, '350045': 210, '350053': 211, '350054': 212, '350403': 213, '350405': 214, '350408': 215, '350409': 216, '350410': 217, '350416': 218, '359306': 219, '359309': 220, '363272': 221, '363611': 222, '364498': 223, '364856': 224, '364858': 225, '364859': 226, '365235': 227, '365237': 228, '36568': 229, '366713': 230, '367226': 231, '367227': 232, '368364': 233, '368402': 234, '368573': 235, '368702': 236, '368783': 237, '36928': 238, '3701': 239, '370129': 240, '370368': 241, '370371': 242, '370374': 243, '371109': 244, '371362': 245, '376563': 246, '376566': 247, '382650': 248, '382652': 249, '382653': 250, '383123': 251, '383162': 252, '386525': 253, '392091': 254, '392095': 255, '4133': 256, '65305': 257, '680': 258, '694': 259, '7266': 260, '7538': 261, '7548': 262, '7935': 263, '9232': 264, 'A. 2. 39186': 265, 'A./5. 3338': 266, 'A.5. 3236': 267, 'A/4 31416': 268, 'A/4 48871': 269, 'A/4 48873': 270, 'A/5 1478': 271, 'A/5 21175': 272, 'A/5. 3337': 273, 'A/5. 851': 274, 'AQ/3. 30631': 275, 'AQ/4 3130': 276, 'C 17368': 277, 'C 4001': 278, 'C.A. 15185': 279, 'C.A. 2315': 280, 'C.A. 2673': 281, 'C.A. 30769': 282, 'C.A. 31029': 283, 'C.A. 31030': 284, 'C.A. 33112': 285, 'C.A. 33595': 286, 'C.A. 34050': 287, 'C.A. 34644': 288, 'C.A. 34651': 289, 'C.A. 37671': 290, 'C.A. 42795': 291, 'C.A. 49867': 292, 'C.A. 6212': 293, 'CA 2144': 294, 'CA 31352': 295, 'CA. 2343': 296, 'F.C. 12750': 297, 'F.C. 12998': 298, 'F.C.C. 13528': 299, 'F.C.C. 13534': 300, 'F.C.C. 13540': 301, 'LP 1588': 302, 'PC 17483': 303, 'PC 17531': 304, 'PC 17558': 305, 'PC 17562': 306, 'PC 17569': 307, 'PC 17580': 308, 'PC 17585': 309, 'PC 17591': 310, 'PC 17592': 311, 'PC 17594': 312, 'PC 17597': 313, 'PC 17598': 314, 'PC 17599': 315, 'PC 17603': 316, 'PC 17606': 317, 'PC 17607': 318, 'PC 17608': 319, 'PC 17613': 320, 'PC 17755': 321, 'PC 17756': 322, 'PC 17757': 323, 'PC 17758': 324, 'PC 17759': 325, 'PC 17760': 326, 'PC 17761': 327, 'PP 9549': 328, 'S.C./PARIS 2079': 329, 'S.O./P.P. 2': 330, 'S.O./P.P. 251': 331, 'S.O./P.P. 752': 332, 'S.O.C. 14879': 333, 'SC 14888': 334, 'SC/A.3 2861': 335, 'SC/A4 23568': 336, 'SC/AH 29037': 337, 'SC/AH 3085': 338, 'SC/PARIS 2147': 339, 'SC/PARIS 2148': 340, 'SC/PARIS 2159': 341, 'SC/PARIS 2166': 342, 'SC/PARIS 2167': 343, 'SC/PARIS 2168': 344, 'SC/Paris 2123': 345, 'SOTON/O.Q. 3101262': 346, 'SOTON/O.Q. 3101263': 347, 'SOTON/O.Q. 3101308': 348, 'SOTON/O.Q. 3101309': 349, 'SOTON/O.Q. 3101314': 350, 'SOTON/O.Q. 3101315': 351, 'SOTON/O2 3101284': 352, 'SOTON/OQ 392083': 353, 'STON/O 2. 3101268': 354, 'STON/O 2. 3101291': 355, 'STON/O 2. 3101270': 356, 'STON/OQ. 369943': 357, 'W./C. 14260': 358, 'W./C. 14266': 359, 'W./C. 6607': 360, 'W./C. 6608': 361, 'W.E.P. 5734': 362}
Feature Embarked
mapping {'C': 0, 'Q': 1, 'S': 2}

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	892	0	3	206	1	34.5	0	0	152	7.8292	1
1	893	1	3	403	0	47.0	1	0	221	7.0000	2
2	894	0	2	269	1	62.0	0	0	73	9.6875	1
3	895	0	3	408	1	27.0	0	0	147	8.6625	2
4	896	1	3	178	0	22.0	1	1	138	12.2875	2

In [ ]:

```
X = data[['PassengerId','Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Fare','Embarked']]
Y = data[['Survived']]
X.shape
Y.shape
```

Out[ ]:

```
(418, 1)
```

`Create a logistic regression model`

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=10)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)



classifier = LogisticRegression()


classifier.fit(X_train,Y_train)
```

```
(292, 10)
(292, 1)
(126, 10)
(126, 1)
```

```
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was pas
sed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\OMOLP091\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]:

▾ LogisticRegression

LogisticRegression()

`Logistic Regression : Prediction`

In [ ]:

```
Y_pred=classifier.predict(X_test)
new_data = pd.DataFrame()
new_data = (X_test)

new_data['Actual'] = Y_test
new_data['Predicted'] = Y_pred
new_data
```

Out[ ]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Actual	Predicted
362	1254	2	387	0	31.00000	0	0	295	21.0000	2	1	1
154	1046	3	17	1	13.00000	4	2	181	31.3875	2	0	0
47	939	3	345	1	30.27259	0	0	243	7.7500	1	0	0
100	992	1	360	0	43.00000	1	0	29	55.4417	0	1	1
187	1079	3	104	1	17.00000	2	0	270	8.0500	2	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
274	1166	3	330	1	30.27259	0	0	110	7.2250	0	0	0
218	1110	1	401	0	50.00000	1	1	13	211.5000	0	1	1
29	921	3	339	1	30.27259	2	0	104	21.6792	0	0	0
279	1171	2	290	1	22.00000	0	0	358	10.5000	2	0	0
278	1170	2	385	1	30.00000	1	0	295	21.0000	2	0	0

126 rows × 12 columns

---

4. Random Forest:

Complete code Steps:

- Create a Random Forest classifier
- Train the model
- Make predictions

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.metrics import accuracy_score, classification_report

# Load the data
titanic_data = pd.read_csv("data/titanic.csv")

# Drop columns that might not be relevant for this simple example
titanic_data = titanic_data.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1)

# Handle missing values (for simplicity, you can customize this based on your needs)
titanic_data = titanic_data.dropna()

# Convert categorical variables (like 'Sex' and 'Embarked') to numerical
titanic_data = pd.get_dummies(titanic_data, columns=["Sex", "Embarked"], drop_first=True)

# Define features (X) and target variable (y)
X = titanic_data.drop("Survived", axis=1)
y = titanic_data["Survived"]

# 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)

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        45
           1       1.00      1.00      1.00        22

    accuracy                           1.00        67
   macro avg       1.00      1.00      1.00        67
weighted avg       1.00      1.00      1.00        67
```

---

5. Ensemble Methods:

Scikit-learn provides ensemble methods that combine multiple base models to improve overall performance. Random Forests, AdaBoost, and Gradient Boosting are popular ensemble methods in scikit-learn. Here's an example of training a Random Forest classifier:

Semi-Supervised Learning

Semi-supervised learning is a type of machine learning where a model is trained on both labeled and unlabeled data. It combines elements of supervised learning (using labeled data) and unsupervised learning (using unlabeled data) to improve the model's performance.

Semi-supervised learning is particularly useful when labeled data is scarce but unlabeled data is abundant. It allows the model to leverage the unlabeled data to improve its performance.

In this description, we'll explore how to perform semi-supervised learning using Python. We'll use the scikit-learn library, which provides various machine learning algorithms and tools.

First, let's start by installing scikit-learn if you haven't already. You can use the following command to install it via pip:

In [ ]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\omolp091\anaconda3\lib\site-packages (1.2.1)Note: you may need to restar
t the kernel to use updated packages.

Requirement already satisfied: numpy>=1.17.3 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)

Requirement already satisfied: numpy>=1.17.3 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\omolp091\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
```

In [ ]:

```python
# Step 1: Import the necessary libraries
from sklearn.semi_supervised import LabelPropagation
from sklearn.datasets import make_classification

# Step 2: Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=8, n_redundant=2, random_state=42)

# Step 3: Split the dataset into labeled and unlabeled data
labeled_X = X[:100]  # First 100 samples are labeled
labeled_y = y[:100]  # Corresponding labels for the labeled samples
unlabeled_X = X[100:]  # Remaining samples are unlabeled

# Step 4: Create a semi-supervised learning model and fit it
model = LabelPropagation()
model.fit(labeled_X, labeled_y)

# Step 5: Predict the labels for the unlabeled data
predicted_labels = model.transduction_[100:]

# Step 6: Evaluate the model's performance on the labeled data
accuracy = model.score(labeled_X, labeled_y)

# Step 7: Print the results
print("Accuracy on labeled data: {:.2f}%".format(accuracy * 100))
```

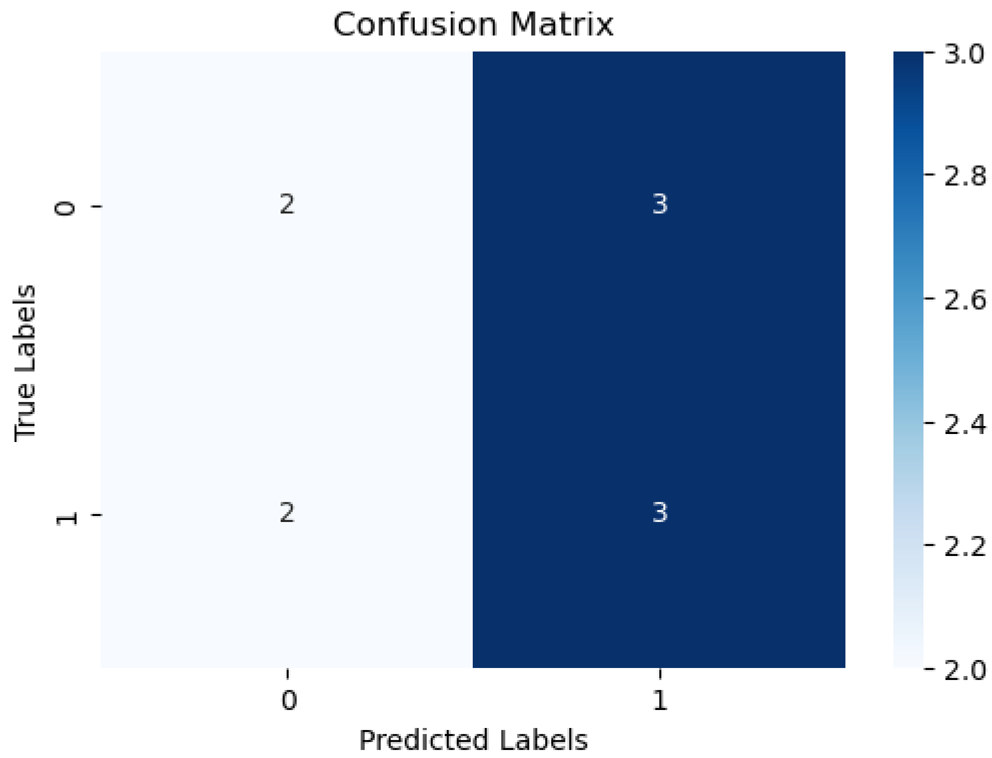
```
Accuracy on labeled data: 100.00%
```

Confusion Matrix:

Confusion matrix is a popular tool for evaluating the performance of classification models. It provides a comprehensive summary of how well the model has classified the different classes in a dataset. In this description, we'll explore how to create and interpret a confusion matrix using Python.

Analyzing the confusion matrix can provide insights into the performance of the classification model, such as identifying which classes are frequently misclassified.

In [ ]:
```
# Step 1: Import the necessary libraries
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Define the true labels and predicted labels
true_labels = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
predicted_labels = [1, 1, 1, 0, 0, 1, 0, 1, 0, 1]

# Step 3: Create a confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Step 4: Visualize the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



6. Scikit Learn - Project on Label Propagation

Label Propagation digits active learning

Demonstrates an active learning technique to learn handwritten digits using label propagation.

We start by training a label propagation model with only 10 labeled points, then we select the top five most uncertain points to label. Next, we train with 15 labeled points (original 10 + 5 new ones). We repeat this process four times to have a model trained with 30 labeled examples. Note you can increase this to label more than 30 by changing `max_iterations`. Labeling more than 30 can be useful to get a sense for the speed of convergence of this active learning technique.

A plot will appear showing the top 5 most uncertain digits for each iteration of training. These may or may not contain mistakes, but we will train the next model with their true labels.

Imports

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

from sklearn import datasets
from sklearn.semi_supervised import LabelSpreading
from sklearn.metrics import classification_report, confusion_matrix
```

Python Implementation

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

from sklearn import datasets
from sklearn.semi_supervised import LabelSpreading
from sklearn.metrics import classification_report, confusion_matrix

digits = datasets.load_digits()
rng = np.random.RandomState(0)
indices = np.arange(len(digits.data))
rng.shuffle(indices)

X = digits.data[indices[:330]]
y = digits.target[indices[:330]]
images = digits.images[indices[:330]]

n_total_samples = len(y)
n_labeled_points = 40
max_iterations = 5

unlabeled_indices = np.arange(n_total_samples)[n_labeled_points:]
f = plt.figure()

for i in range(max_iterations):
```

```
    if len(unlabeled_indices) == 0:
        print("No unlabeled items left to label.")
        break
    y_train = np.copy(y)
    y_train[unlabeled_indices] = -1

    lp_model = LabelSpreading(gamma=0.25, max_iter=20)
    lp_model.fit(X, y_train)

    predicted_labels = lp_model.transduction_[unlabeled_indices]
    true_labels = y[unlabeled_indices]

    cm = confusion_matrix(true_labels, predicted_labels, labels=lp_model.classes_)

    print("Iteration %i %s" % (i, 70 * "_"))
    print(
        "Label Spreading model: %d labeled & %d unlabeled (%d total)"
        % (n_labeled_points, n_total_samples - n_labeled_points, n_total_samples)
    )

    print(classification_report(true_labels, predicted_labels))

    print("Confusion matrix")
    print(cm)

    # compute the entropies of transduced label distributions
    pred_entropies = stats.distributions.entropy(lp_model.label_distributions_.T)

    # select up to 5 digit examples that the classifier is most uncertain about
    uncertainty_index = np.argsort(pred_entropies)[::-1]
    uncertainty_index = uncertainty_index[
        np.in1d(uncertainty_index, unlabeled_indices)
    ][:5]

    # keep track of indices that we get labels for
    delete_indices = np.array([], dtype=int)

    # for more than 5 iterations, visualize the gain only on the first 5
    if i < 5:
        f.text(
            0.05,
            (1 - (i + 1) * 0.183),
            "model %d\n\nfit with\n%d labels" % ((i + 1), i * 5 + 10),
            size=10,
```

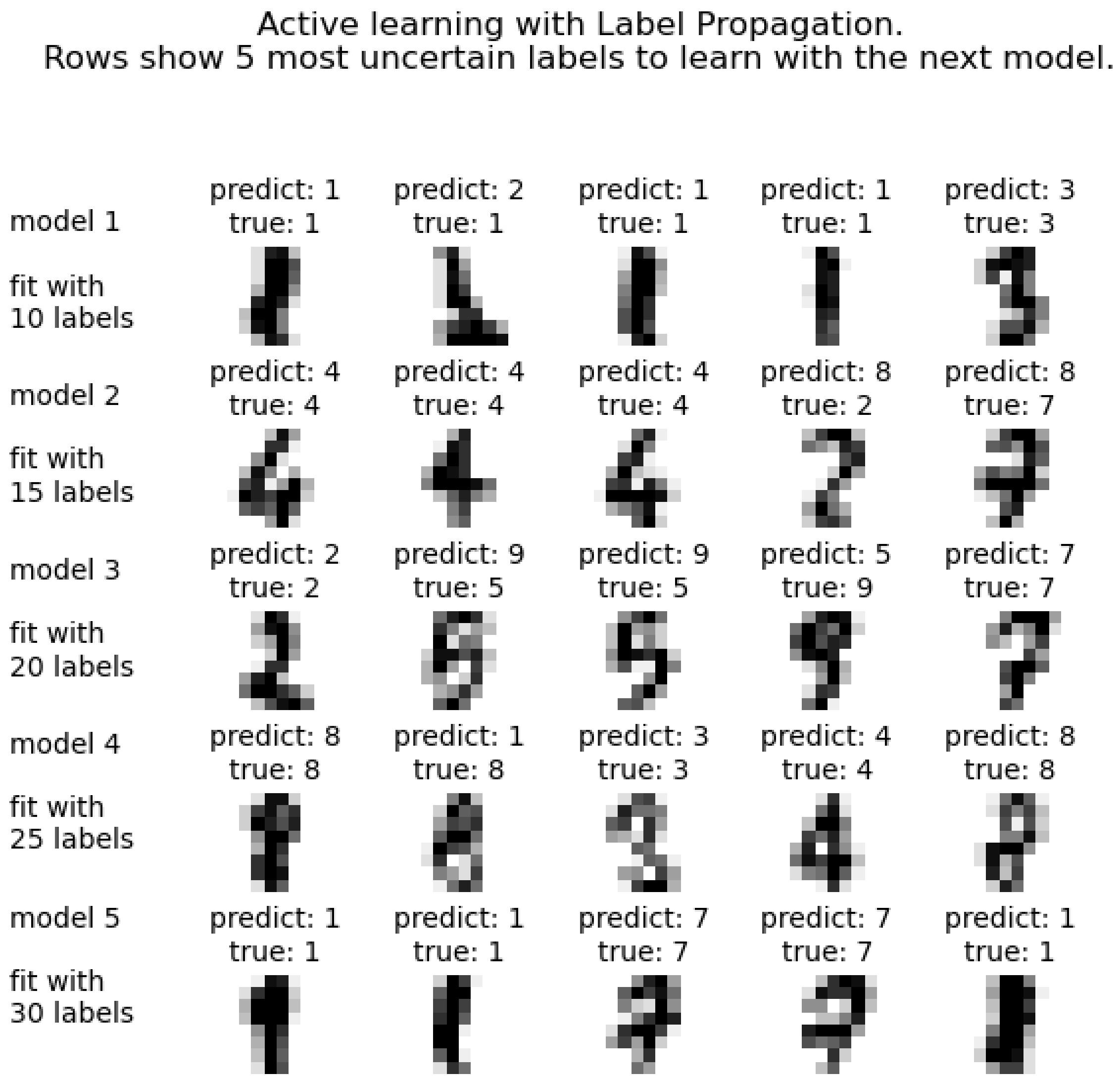
```
        )
    for index, image_index in enumerate(uncertainty_index):
        image = images[image_index]

        # for more than 5 iterations, visualize the gain only on the first 5
        if i < 5:
            sub = f.add_subplot(5, 5, index + 1 + (5 * i))
            sub.imshow(image, cmap=plt.cm.gray_r, interpolation="none")
            sub.set_title(
                "predict: %i\ntrue: %i"
                % (lp_model.transduction_[image_index], y[image_index]),
                size=10,
            )
            sub.axis("off")

        # labeling 5 points, remote from labeled set
        (delete_index,) = np.where(unlabeled_indices == image_index)
        delete_indices = np.concatenate((delete_indices, delete_index))

    unlabeled_indices = np.delete(unlabeled_indices, delete_indices)
    n_labeled_points += len(uncertainty_index)

f.suptitle(
    "Active learning with Label Propagation.\nRows show 5 most "
    "uncertain labels to learn with the next model.",
    y=1.15,
)
plt.subplots_adjust(left=0.2, bottom=0.03, right=0.9, top=0.9, wspace=0.2, hspace=0.85)
plt.show()
```

```
Iteration 0 ______________________________________________________________________
Label Spreading model: 40 labeled & 290 unlabeled (330 total)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.78      0.69      0.73        26
           2       0.93      0.93      0.93        29
           3       1.00      0.89      0.94        27
           4       0.92      0.96      0.94        23
           5       0.96      0.70      0.81        33
           6       0.97      0.97      0.97        35
           7       0.94      0.91      0.92        33
           8       0.62      0.89      0.74        28
           9       0.73      0.79      0.76        34

    accuracy                           0.87       290
   macro avg       0.89      0.87      0.87       290
weighted avg       0.88      0.87      0.87       290

Confusion matrix
[[22  0  0  0  0  0  0  0  0  0]
 [ 0 18  2  0  0  0  1  0  5  0]
 [ 0  0 27  0  0  0  0  0  2  0]
 [ 0  0  0 24  0  0  0  0  3  0]
 [ 0  1  0  0 22  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  0 10]
 [ 0  1  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 30  3  0]
 [ 0  3  0  0  0  0  0  0 25  0]
 [ 0  0  0  0  2  1  0  2  2 27]]
Iteration 1 ______________________________________________________________________
Label Spreading model: 45 labeled & 285 unlabeled (330 total)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.79      1.00      0.88        22
           2       1.00      0.93      0.96        29
           3       1.00      1.00      1.00        26
           4       0.92      0.96      0.94        23
           5       0.96      0.70      0.81        33
           6       1.00      0.97      0.99        35
           7       0.94      0.91      0.92        33
           8       0.77      0.86      0.81        28
           9       0.73      0.79      0.76        34
```

```
    accuracy                           0.90       285
   macro avg       0.91      0.91      0.91       285
weighted avg       0.91      0.90      0.90       285

Confusion matrix
[[22  0  0  0  0  0  0  0  0  0]
 [ 0 22  0  0  0  0  0  0  0  0]
 [ 0  0 27  0  0  0  0  0  2  0]
 [ 0  0  0 26  0  0  0  0  0  0]
 [ 0  1  0  0 22  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  0 10]
 [ 0  1  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 30  3  0]
 [ 0  4  0  0  0  0  0  0 24  0]
 [ 0  0  0  0  2  1  0  2  2 27]]
Iteration 2 ______________________________________________________________________
Label Spreading model: 50 labeled & 280 unlabeled (330 total)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.85      1.00      0.92        22
           2       1.00      1.00      1.00        28
           3       1.00      1.00      1.00        26
           4       0.87      1.00      0.93        20
           5       0.96      0.70      0.81        33
           6       1.00      0.97      0.99        35
           7       0.94      1.00      0.97        32
           8       0.92      0.86      0.89        28
           9       0.73      0.79      0.76        34

    accuracy                           0.92       280
   macro avg       0.93      0.93      0.93       280
weighted avg       0.93      0.92      0.92       280

Confusion matrix
[[22  0  0  0  0  0  0  0  0  0]
 [ 0 22  0  0  0  0  0  0  0  0]
 [ 0  0 28  0  0  0  0  0  0  0]
 [ 0  0  0 26  0  0  0  0  0  0]
 [ 0  0  0  0 20  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  0 10]
 [ 0  1  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 32  0  0]
```

```
 [ 0  3  0  0  1  0  0  0 24  0]
 [ 0  0  0  0  2  1  0  2  2 27]]
Iteration 3 ______________________________________________________________________
Label Spreading model: 55 labeled & 275 unlabeled (330 total)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.85      1.00      0.92        22
           2       1.00      1.00      1.00        27
           3       1.00      1.00      1.00        26
           4       0.87      1.00      0.93        20
           5       0.96      0.87      0.92        31
           6       1.00      0.97      0.99        35
           7       1.00      1.00      1.00        31
           8       0.92      0.86      0.89        28
           9       0.88      0.85      0.86        33

    accuracy                           0.95       275
   macro avg       0.95      0.95      0.95       275
weighted avg       0.95      0.95      0.95       275

Confusion matrix
[[22  0  0  0  0  0  0  0  0  0]
 [ 0 22  0  0  0  0  0  0  0  0]
 [ 0  0 27  0  0  0  0  0  0  0]
 [ 0  0  0 26  0  0  0  0  0  0]
 [ 0  0  0  0 20  0  0  0  0  0]
 [ 0  0  0  0  0 27  0  0  0  4]
 [ 0  1  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 31  0  0]
 [ 0  3  0  0  1  0  0  0 24  0]
 [ 0  0  0  0  2  1  0  0  2 28]]
Iteration 4 ______________________________________________________________________
Label Spreading model: 60 labeled & 270 unlabeled (330 total)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       0.96      1.00      0.98        22
           2       1.00      0.96      0.98        27
           3       0.96      1.00      0.98        25
           4       0.86      1.00      0.93        19
           5       0.96      0.87      0.92        31
           6       1.00      0.97      0.99        35
           7       1.00      1.00      1.00        31
```

```
           8       0.92      0.96      0.94        25
           9       0.88      0.85      0.86        33

    accuracy                           0.96       270
   macro avg       0.95      0.96      0.96       270
weighted avg       0.96      0.96      0.96       270

Confusion matrix
[[22  0  0  0  0  0  0  0  0  0]
 [ 0 22  0  0  0  0  0  0  0  0]
 [ 0  0 26  1  0  0  0  0  0  0]
 [ 0  0  0 25  0  0  0  0  0  0]
 [ 0  0  0  0 19  0  0  0  0  0]
 [ 0  0  0  0  0 27  0  0  0  4]
 [ 0  1  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  0  0 31  0  0]
 [ 0  0  0  0  1  0  0  0 24  0]
 [ 0  0  0  0  2  1  0  0  2 28]]
```


Active learning with Label Propagation.
Rows show 5 most uncertain labels to learn with the next model.
model 1
fit with
10 labels
predict: 1
true: 1
predict: 2
true: 1
predict: 1
true: 1
predict: 1
true: 1
predict: 3
true: 3
model 2
fit with
15 labels
predict: 4
true: 4
predict: 4
true: 4
predict: 4
true: 4
predict: 8
true: 2
predict: 8
true: 7
model 3
fit with
20 labels
predict: 2
true: 2
predict: 9
true: 5
predict: 9
true: 5
predict: 5
true: 9
predict: 7
true: 7
model 4
fit with
25 labels
predict: 8
true: 8
predict: 1
true: 8
predict: 3
true: 3
predict: 4
true: 4
predict: 8
true: 8
model 5
fit with
30 labels
predict: 1
true: 1
predict: 1
true: 1
predict: 7
true: 7
predict: 7
true: 7
predict: 1
true: 1