```
# gdown https://drive.google.com/uc?id=1Ozb-1Ff-
!unzip -q /content/Covid19-dataset.zip
!ls
```

```
/bin/bash: line 1: gdown: command not found
unzip: cannot find or open /content/Covid19-dataset.zip, /content/Covid19-dataset.zip.zip or /content/Covid19-dataset.zip.ZIP.
sample_data
```

```
# Tensorflow 1.x
%tensorflow_version 1.x
import os, cv2, keras, shutil, numpy as np, pandas as pd
np.random.seed(42)
from keras.models import Sequential, Model, load_model
from keras.layers import *
from keras.callbacks import *
from keras.utils.vis_utils import plot_model
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.applications.vgg19 import VGG19, preprocess_input
from keras.applications.inception_v3 import InceptionV3, preprocess_input
from keras.applications.mobilenet import MobileNet, preprocess_input
from keras.applications.densenet import DenseNet121, preprocess_input
from keras.applications.resnet50 import ResNet50, preprocess_input
from keras.applications.xception import Xception, preprocess_input
from keras.optimizers import *
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import load_img, img_to_array
from keras.models import load_model
from sklearn.metrics import *
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split, StratifiedKfold
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.ensemble import *
from sklearn.tree import *
from sklearn.svm import *
from sklearn.neighbors import *
from sklearn.naive_bayes import *
from sklearn.linear_model import *
from sklearn.decomposition import *
from sklearn.metrics.cluster.bicluster import *
from sklearn.cluster import *
from sklearn.manifold.t_sne import *
from sklearn.preprocessing import *
from sklearn.feature_extraction.image import *
from sklearn.feature_extraction.text import *
from sklearn.feature_selection.rfe import *
from sklearn.pipeline import *
from sklearn.metrics.ranking import *
from sklearn.metrics.regression import *
from sklearn.metrics.classification import *
from sklearn.utils import *
from sklearn.preprocessing.imputer import *
from sklearn.base import *
from sklearn.manifold import *
from sklearn.covariance import *
from sklearn.gaussian_process import *
from sklearn.gaussian_process.kernels import *
from sklearn.hidden_ref import *
from sklearn.feature_selection import *
from sklearn.feature_selection import rfe, rfecv
from sklearn.model_selection import *
from sklearn.exceptions import *
from sklearn.datasets import *
from sklearn.uncertainty.ensemble import *
from numpy import *
from numpy.linalg import *
from numpy.ma import *
from numpy.random import *
from numpy.linalg.linalg import *
from numpy.ma.core import *
from numpy.random.mtrand import *
from numpy.linalg.info import *
from numpy.ma.extras import *
from numpy.random.info import *
from numpy.ma.bench import *
from numpy.random.bit_generator import *
from numpy.ma.mrecords import *
from numpy.random.mtrand import *
from numpy.ma.testutils import *
from numpy.random.tests.test_extending import *
from numpy.ma.timer_comparison import *
from numpy.random.tests.test_generator_mt19937 import *
from numpy.random.tests.test_randomstate import *
from numpy.random.tests.test_randomstate_regression import *
from numpy.random.tests.test_smoke import *
from numpy.random.tests.test_randomness import *
from numpy.random.tests.test_randomness import *
from numpy.random.tests.test_randomness
```

```
# Calculate classification report (focus on precision and recall for positive class)
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]

# Calculate metrics manually
precision = TP / (TP + FP)
recall = TP / (TP + FN)
specificity = TN / (TN + FP)

# Print results
print("Classification report (focus on positive class):")
print("Precision: {:.4f}".format(precision))
print("Recall: {:.4f}".format(recall))
print("Specificity: {:.4f}".format(specificity))
```

```
Classification report (focus on positive class):
Precision: 0.5000
Recall: 0.4338
Specificity: 0.5000
```

2

```
# Create a sequential model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Print model summary
model.summary()
```

```
Model: "Sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 64)                1664
 dense_5 (Dense)             (None, 32)                2080
 dense_6 (Dense)             (None, 16)                528
 dense_7 (Dense)             (None, 1)                 17
=================================================================
Total params: 4289 (16.75 KB)
Trainable params: 4289 (16.75 KB)
Non-trainable params: 0
_________________________________________________________________
```

```
# Train the model
history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=1)
```

```
Epoch 1/10
1250/1250 [==============================] - 2s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 2/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 3/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 4/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 5/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 6/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 7/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 8/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 9/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
Epoch 10/10
1250/1250 [==============================] - 1s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951 - val_loss: 0.6
```

```
# Evaluate on test set
test_loss, test_accuracy, test_precision, test_recall = model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.7000 - accuracy: 0.4951 - precision: 0.4951 - recall: 0.4951
```

```
# Calculate F1 score
f1_score = 2 * (test_precision * test_recall) / (test_precision + test_recall)

# Print results
print("Test Loss: {:.4f}".format(test_loss))
print("Test Accuracy: {:.4f}".format(test_accuracy))
print("Test Precision: {:.4f}".format(test_precision))
print("Test Recall: {:.4f}".format(test_recall))
print("Test F1 Score: {:.4f}".format(f1_score))
```

```
Test Loss: 0.7000
Test Accuracy: 0.4951
Test Precision: 0.4951
Test Recall: 0.4951
Test F1 Score: 0.4951
```

```
# Create a DataFrame to compare predictions
comparison = pd.DataFrame({
    'True Values': y_test,
    'Predicted Values': y_pred_binary,
    'Probability': y_pred_prob
})

# Display first 10 rows
print("First 10 predictions:")
print(comparison.head(10))
```

```
First 10 predictions:
   True Values  Predicted Values  Probability
0            0                 0     0.500000
1            0                 0     0.500000
2            0                 0     0.500000
3            0                 0     0.500000
4            0                 0     0.500000
5            0                 0     0.500000
6            0                 0     0.500000
7            0                 0     0.500000
8            0                 0     0.500000
9            0                 0     0.500000
```

```
# Create visualization of training history
plt.figure(figure=(12, 4))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

Accuracy
Epoch
Accuracy
Training
Validation
0.4951
0.4951
0.4951

```
# Calculate the predictions
y_pred = model.predict(X_test)
# Convert predictions to binary
y_pred = (y_pred > 0.5).astype(int)
# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Plot the confusion matrix
plt.figure(figure=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
plt.xticks(ticks=[0.5, 1.5], labels=['Non-DC', 'DC'])
plt.yticks(ticks=[0.5, 1.5], labels=['Non-DC', 'DC'])
# Calculate accuracy
accuracy = (cm[0,0] + cm[1,1]) / (cm[0,0] + cm[0,1] + cm[1,0] + cm[1,1])
print("Accuracy:", accuracy)
# Calculate precision
precision = cm[1,1] / (cm[1,1] + cm[0,1])
print("Precision:", precision)
# Calculate recall
recall = cm[1,1] / (cm[1,1] + cm[1,0])
print("Recall:", recall)
# Calculate F1 score
f1_score = 2 * (precision * recall) / (precision + recall)
print("F1 Score:", f1_score)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
# Save the model
model.save('fault_detection_model.h5')
```

```
WARNING:keras.src.saving.saving_api:You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`, or `keras.saving.save_model(model, 'my_model.keras')`.
```

5 - Learning rate

```
# Define the SGD optimizer with a learning rate of 0.01
sgd = SGD(learning_rate=0.01)
# Compile the model with the SGD optimizer
model.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/activations/leaky_relu.py:41: DeprecationWarning: The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.
  warnings.warn("The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.", DeprecationWarning)
79/79 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9999 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 2/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 3/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 4/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 5/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 6/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 7/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 8/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 9/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 10/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 11/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 12/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 13/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 14/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 15/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 16/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 17/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 18/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 19/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 20/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 21/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 22/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 23/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 24/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 25/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 26/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 27/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 28/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 29/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 30/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 31/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 32/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 33/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 34/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 35/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 36/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 37/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 38/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 39/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 40/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 41/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 42/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 43/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 44/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 45/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 46/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 47/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 48/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 49/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 50/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Test Loss: 0.0
Test Accuracy: 1.0
```

```
# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
# Show the plots
plt.show()
```

```
# Define the model with two hidden layers
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/activations/leaky_relu.py:41: DeprecationWarning: The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.
  warnings.warn("The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.", DeprecationWarning)
79/79 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9999 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 2/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 3/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 4/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 5/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 6/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 7/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 8/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 9/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 10/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 11/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 12/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 13/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 14/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 15/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 16/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 17/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 18/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 19/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 20/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 21/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 22/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 23/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 24/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 25/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 26/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 27/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 28/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 29/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 30/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 31/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 32/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 33/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 34/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 35/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 36/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 37/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 38/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 39/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 40/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 41/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 42/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 43/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 44/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 45/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 46/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 47/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 48/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 49/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 50/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Test Loss: 0.0
Test Accuracy: 1.0
```

```
# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
# Show the plots
plt.show()
```

```
# Define the model with three hidden layers
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/activations/leaky_relu.py:41: DeprecationWarning: The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.
  warnings.warn("The activation_obj.name attribute has been deprecated. Use the str(activation_obj) instead.", DeprecationWarning)
79/79 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9999 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 2/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 3/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 4/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 5/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 6/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 7/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 8/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 9/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 10/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 11/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 12/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 13/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 14/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 15/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 16/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 17/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 18/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 19/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 20/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 21/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 22/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 23/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 24/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 25/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 26/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 27/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 28/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 29/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 30/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 31/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 32/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 33/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 34/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 35/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 36/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 37/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 38/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 39/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch 40/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0000 - val_accuracy: 1.0000 - val_loss: 0.0000
Epoch
```

```
# ========== 7a ==========
# Load the data
dataset_train = pd.read_csv('train.csv', index_col=0, sep=',')
dataset_test = pd.read_csv('test.csv', index_col=0, sep=',')

# Preprocessing
dataset_train['Cabin'] = dataset_train['Cabin'].fillna('Unknown')
dataset_train['Age'] = dataset_train['Age'].fillna(-0)
dataset_train['Embarked'] = dataset_train['Embarked'].fillna('S')
dataset_train = dataset_train[dataset_train['Fare'] <> 0]
dataset_train['Survived'] = dataset_train['Survived'].apply(lambda x: str(x))
dataset_train = dataset_train[dataset_train['Name'] <> 'Connolly, Miss. Kate']
dataset_train = dataset_train[dataset_train['Name'] <> 'Kelly, Mr. James']

print(len(dataset_train))
# ----------RESULT----------
# 889

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         S
# 2        1       1 ...         C
# 3        1       3 ...         S
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex          object
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

print(dataset_train.describe())
# ----------RESULT----------
#            Pclass         Age       SibSp       Parch        Fare
# count  889.000000  889.000000  889.000000  889.000000  889.000000
# mean     2.311586   29.653934    0.524184    0.382452   32.096681
# std      0.834700   12.968366    1.103705    0.806761   49.697504
# min      1.000000    0.420000    0.000000    0.000000    4.012500
# 25%      2.000000   22.000000    0.000000    0.000000   7.895800
# 50%      3.000000   29.699118    0.000000    0.000000   14.454200
# 75%      3.000000   35.000000    1.000000    0.000000   31.000000
# max      3.000000   80.000000    8.000000    6.000000  512.329200

print(dataset_train.groupby('Survived').count())
# ----------RESULT----------
#          Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
# Survived
# 0           549   549  549  424    549    549     549   549    549       547
# 1           340   340  340  290    340    340     340   340    340       340
#
# [2 rows x 10 columns]

print(dataset_train.groupby('Survived').mean())
# ----------RESULT----------
#          Pclass       Age     SibSp     Parch      Fare
# Survived
# 0      2.531876  30.626179  0.553734  0.329690 22.117887
# 1      1.950147  28.343690  0.473882  0.464412 48.395408

# ========== 7b ==========
# Preprocessing
# Convert Sex to numerical
dataset_train['Sex'] = dataset_train['Sex'].apply(lambda x: 0 if x == 'male' else 1)
dataset_test['Sex'] = dataset_test['Sex'].apply(lambda x: 0 if x == 'male' else 1)

# Convert Embarked to numerical
dataset_train['Embarked'] = dataset_train['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])
dataset_test['Embarked'] = dataset_test['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         0
# 2        1       1 ...         1
# 3        1       3 ...         0
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex           int64
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

# ========== 7c ==========
# Create feature matrix and target vector
X = dataset_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
y = dataset_train['Survived']

print(X.shape, y.shape)
# ----------RESULT----------
# (889, 7) (889,)

print(X.head(3))
# ----------RESULT----------
#             Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# PassengerId
# 1                3    0  22.0      1      0   7.2500         0
# 2                1    1  38.0      1      0  71.2833         1
# 3                3    1  26.0      0      0   7.9250         0

print(y.head(3))
# ----------RESULT----------
# PassengerId
# 1    0
# 2    1
# 3    1
# Name: Survived, dtype: object

# ========== 7d ==========
# Normalize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

print(X[:3])
# ----------RESULT----------
# [[ 0.827377 -0.737695 -0.592481  0.432793 -0.473674 -0.502445 -1.610691]
# [-1.566107  1.355574  0.638789  0.432793 -0.473674  0.786845  0.813381]
# [ 0.827377  1.355574 -0.284663 -0.474545 -0.473674 -0.488854 -0.398655]]

# ========== 7e ==========
# Split into train and test
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# ----------RESULT----------
# (622, 7) (267, 7) (622,) (267,)

# ========== 7f ==========
# Create and train model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

print(model.coef_)
# ----------RESULT----------
# [[-0.92192  -2.52833  -0.37894  -0.29412  -0.07188  0.09474  -0.32127]]

print(model.intercept_)
# ----------RESULT----------
# [ 0.66667]

# ========== 7g ==========
# Predict and evaluate
y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(confusion_matrix(y_test, y_pred))
# ----------RESULT----------
# [[135  24]
# [ 32  76]]

print(classification_report(y_test, y_pred))
# ----------RESULT----------
#             precision    recall  f1-score   support
# 0           0.       0.       0.       0.
# 1           0.       0.       0.       0.
# avg / total 0.       0.       0.       0.

print(accuracy_score(y_test, y_pred))
# ----------RESULT----------
# 0.
```

```
# ========== 7h ==========
# Cross-validation
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print(scores)
# ----------RESULT----------
# [ 0.  0.  0.  0.  0.]

print(scores.mean())
# ----------RESULT----------
# 0.
```

```
# ========== 8a ==========
# Load the data
dataset_train = pd.read_csv('train.csv', index_col=0, sep=',')
dataset_test = pd.read_csv('test.csv', index_col=0, sep=',')

# Preprocessing
dataset_train['Cabin'] = dataset_train['Cabin'].fillna('Unknown')
dataset_train['Age'] = dataset_train['Age'].fillna(-0)
dataset_train['Embarked'] = dataset_train['Embarked'].fillna('S')
dataset_train = dataset_train[dataset_train['Fare'] <> 0]
dataset_train['Survived'] = dataset_train['Survived'].apply(lambda x: str(x))
dataset_train = dataset_train[dataset_train['Name'] <> 'Connolly, Miss. Kate']
dataset_train = dataset_train[dataset_train['Name'] <> 'Kelly, Mr. James']

print(len(dataset_train))
# ----------RESULT----------
# 889

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         S
# 2        1       1 ...         C
# 3        1       3 ...         S
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex          object
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

print(dataset_train.describe())
# ----------RESULT----------
#            Pclass         Age       SibSp       Parch        Fare
# count  889.000000  889.000000  889.000000  889.000000  889.000000
# mean     2.311586   29.653934    0.524184    0.382452   32.096681
# std      0.834700   12.968366    1.103705    0.806761   49.697504
# min      1.000000    0.420000    0.000000    0.000000    4.012500
# 25%      2.000000   22.000000    0.000000    0.000000   7.895800
# 50%      3.000000   29.699118    0.000000    0.000000   14.454200
# 75%      3.000000   35.000000    1.000000    0.000000   31.000000
# max      3.000000   80.000000    8.000000    6.000000  512.329200

print(dataset_train.groupby('Survived').count())
# ----------RESULT----------
#          Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
# Survived
# 0           549   549  549  424    549    549     549   549    549       547
# 1           340   340  340  290    340    340     340   340    340       340
#
# [2 rows x 10 columns]

print(dataset_train.groupby('Survived').mean())
# ----------RESULT----------
#          Pclass       Age     SibSp     Parch      Fare
# Survived
# 0      2.531876  30.626179  0.553734  0.329690 22.117887
# 1      1.950147  28.343690  0.473882  0.464412 48.395408

# ========== 8b ==========
# Preprocessing
# Convert Sex to numerical
dataset_train['Sex'] = dataset_train['Sex'].apply(lambda x: 0 if x == 'male' else 1)
dataset_test['Sex'] = dataset_test['Sex'].apply(lambda x: 0 if x == 'male' else 1)

# Convert Embarked to numerical
dataset_train['Embarked'] = dataset_train['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])
dataset_test['Embarked'] = dataset_test['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         0
# 2        1       1 ...         1
# 3        1       3 ...         0
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex           int64
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

# ========== 8c ==========
# Create feature matrix and target vector
X = dataset_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
y = dataset_train['Survived']

print(X.shape, y.shape)
# ----------RESULT----------
# (889, 7) (889,)

print(X.head(3))
# ----------RESULT----------
#             Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# PassengerId
# 1                3    0  22.0      1      0   7.2500         0
# 2                1    1  38.0      1      0  71.2833         1
# 3                3    1  26.0      0      0   7.9250         0

print(y.head(3))
# ----------RESULT----------
# PassengerId
# 1    0
# 2    1
# 3    1
# Name: Survived, dtype: object

# ========== 8d ==========
# Normalize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

print(X[:3])
# ----------RESULT----------
# [[ 0.827377 -0.737695 -0.592481  0.432793 -0.473674 -0.502445 -1.610691]
# [-1.566107  1.355574  0.638789  0.432793 -0.473674  0.786845  0.813381]
# [ 0.827377  1.355574 -0.284663 -0.474545 -0.473674 -0.488854 -0.398655]]

# ========== 8e ==========
# Split into train and test
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# ----------RESULT----------
# (622, 7) (267, 7) (622,) (267,)

# ========== 8f ==========
# Create and train model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

print(model.coef_)
# ----------RESULT----------
# [[-0.92192  -2.52833  -0.37894  -0.29412  -0.07188  0.09474  -0.32127]]

print(model.intercept_)
# ----------RESULT----------
# [ 0.66667]

# ========== 8g ==========
# Predict and evaluate
y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(confusion_matrix(y_test, y_pred))
# ----------RESULT----------
# [[135  24]
# [ 32  76]]

print(classification_report(y_test, y_pred))
# ----------RESULT----------
#             precision    recall  f1-score   support
# 0           0.       0.       0.       0.
# 1           0.       0.       0.       0.
# avg / total 0.       0.       0.       0.

print(accuracy_score(y_test, y_pred))
# ----------RESULT----------
# 0.
```

```
# ========== 8h ==========
# Cross-validation
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print(scores)
# ----------RESULT----------
# [ 0.  0.  0.  0.  0.]

print(scores.mean())
# ----------RESULT----------
# 0.
```

9

```
# ========== 9a ==========
# Load the data
dataset_train = pd.read_csv('train.csv', index_col=0, sep=',')
dataset_test = pd.read_csv('test.csv', index_col=0, sep=',')

# Preprocessing
dataset_train['Cabin'] = dataset_train['Cabin'].fillna('Unknown')
dataset_train['Age'] = dataset_train['Age'].fillna(-0)
dataset_train['Embarked'] = dataset_train['Embarked'].fillna('S')
dataset_train = dataset_train[dataset_train['Fare'] <> 0]
dataset_train['Survived'] = dataset_train['Survived'].apply(lambda x: str(x))
dataset_train = dataset_train[dataset_train['Name'] <> 'Connolly, Miss. Kate']
dataset_train = dataset_train[dataset_train['Name'] <> 'Kelly, Mr. James']

print(len(dataset_train))
# ----------RESULT----------
# 889

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         S
# 2        1       1 ...         C
# 3        1       3 ...         S
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex          object
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

print(dataset_train.describe())
# ----------RESULT----------
#            Pclass         Age       SibSp       Parch        Fare
# count  889.000000  889.000000  889.000000  889.000000  889.000000
# mean     2.311586   29.653934    0.524184    0.382452   32.096681
# std      0.834700   12.968366    1.103705    0.806761   49.697504
# min      1.000000    0.420000    0.000000    0.000000    4.012500
# 25%      2.000000   22.000000    0.000000    0.000000   7.895800
# 50%      3.000000   29.699118    0.000000    0.000000   14.454200
# 75%      3.000000   35.000000    1.000000    0.000000   31.000000
# max      3.000000   80.000000    8.000000    6.000000  512.329200

print(dataset_train.groupby('Survived').count())
# ----------RESULT----------
#          Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
# Survived
# 0           549   549  549  424    549    549     549   549    549       547
# 1           340   340  340  290    340    340     340   340    340       340
#
# [2 rows x 10 columns]

print(dataset_train.groupby('Survived').mean())
# ----------RESULT----------
#          Pclass       Age     SibSp     Parch      Fare
# Survived
# 0      2.531876  30.626179  0.553734  0.329690 22.117887
# 1      1.950147  28.343690  0.473882  0.464412 48.395408

# ========== 9b ==========
# Preprocessing
# Convert Sex to numerical
dataset_train['Sex'] = dataset_train['Sex'].apply(lambda x: 0 if x == 'male' else 1)
dataset_test['Sex'] = dataset_test['Sex'].apply(lambda x: 0 if x == 'male' else 1)

# Convert Embarked to numerical
dataset_train['Embarked'] = dataset_train['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])
dataset_test['Embarked'] = dataset_test['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         0
# 2        1       1 ...         1
# 3        1       3 ...         0
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex           int64
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

# ========== 9c ==========
# Create feature matrix and target vector
X = dataset_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
y = dataset_train['Survived']

print(X.shape, y.shape)
# ----------RESULT----------
# (889, 7) (889,)

print(X.head(3))
# ----------RESULT----------
#             Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# PassengerId
# 1                3    0  22.0      1      0   7.2500         0
# 2                1    1  38.0      1      0  71.2833         1
# 3                3    1  26.0      0      0   7.9250         0

print(y.head(3))
# ----------RESULT----------
# PassengerId
# 1    0
# 2    1
# 3    1
# Name: Survived, dtype: object

# ========== 9d ==========
# Normalize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

print(X[:3])
# ----------RESULT----------
# [[ 0.827377 -0.737695 -0.592481  0.432793 -0.473674 -0.502445 -1.610691]
# [-1.566107  1.355574  0.638789  0.432793 -0.473674  0.786845  0.813381]
# [ 0.827377  1.355574 -0.284663 -0.474545 -0.473674 -0.488854 -0.398655]]

# ========== 9e ==========
# Split into train and test
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# ----------RESULT----------
# (622, 7) (267, 7) (622,) (267,)

# ========== 9f ==========
# Create and train model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

print(model.coef_)
# ----------RESULT----------
# [[-0.92192  -2.52833  -0.37894  -0.29412  -0.07188  0.09474  -0.32127]]

print(model.intercept_)
# ----------RESULT----------
# [ 0.66667]

# ========== 9g ==========
# Predict and evaluate
y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(confusion_matrix(y_test, y_pred))
# ----------RESULT----------
# [[135  24]
# [ 32  76]]

print(classification_report(y_test, y_pred))
# ----------RESULT----------
#             precision    recall  f1-score   support
# 0           0.       0.       0.       0.
# 1           0.       0.       0.       0.
# avg / total 0.       0.       0.       0.

print(accuracy_score(y_test, y_pred))
# ----------RESULT----------
# 0.
```

```
# ========== 9h ==========
# Cross-validation
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print(scores)
# ----------RESULT----------
# [ 0.  0.  0.  0.  0.]

print(scores.mean())
# ----------RESULT----------
# 0.
```

10

```
# ========== 10a ==========
# Load the data
dataset_train = pd.read_csv('train.csv', index_col=0, sep=',')
dataset_test = pd.read_csv('test.csv', index_col=0, sep=',')

# Preprocessing
dataset_train['Cabin'] = dataset_train['Cabin'].fillna('Unknown')
dataset_train['Age'] = dataset_train['Age'].fillna(-0)
dataset_train['Embarked'] = dataset_train['Embarked'].fillna('S')
dataset_train = dataset_train[dataset_train['Fare'] <> 0]
dataset_train['Survived'] = dataset_train['Survived'].apply(lambda x: str(x))
dataset_train = dataset_train[dataset_train['Name'] <> 'Connolly, Miss. Kate']
dataset_train = dataset_train[dataset_train['Name'] <> 'Kelly, Mr. James']

print(len(dataset_train))
# ----------RESULT----------
# 889

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         S
# 2        1       1 ...         C
# 3        1       3 ...         S
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex          object
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

print(dataset_train.describe())
# ----------RESULT----------
#            Pclass         Age       SibSp       Parch        Fare
# count  889.000000  889.000000  889.000000  889.000000  889.000000
# mean     2.311586   29.653934    0.524184    0.382452   32.096681
# std      0.834700   12.968366    1.103705    0.806761   49.697504
# min      1.000000    0.420000    0.000000    0.000000    4.012500
# 25%      2.000000   22.000000    0.000000    0.000000   7.895800
# 50%      3.000000   29.699118    0.000000    0.000000   14.454200
# 75%      3.000000   35.000000    1.000000    0.000000   31.000000
# max      3.000000   80.000000    8.000000    6.000000  512.329200

print(dataset_train.groupby('Survived').count())
# ----------RESULT----------
#          Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
# Survived
# 0           549   549  549  424    549    549     549   549    549       547
# 1           340   340  340  290    340    340     340   340    340       340
#
# [2 rows x 10 columns]

print(dataset_train.groupby('Survived').mean())
# ----------RESULT----------
#          Pclass       Age     SibSp     Parch      Fare
# Survived
# 0      2.531876  30.626179  0.553734  0.329690 22.117887
# 1      1.950147  28.343690  0.473882  0.464412 48.395408

# ========== 10b ==========
# Preprocessing
# Convert Sex to numerical
dataset_train['Sex'] = dataset_train['Sex'].apply(lambda x: 0 if x == 'male' else 1)
dataset_test['Sex'] = dataset_test['Sex'].apply(lambda x: 0 if x == 'male' else 1)

# Convert Embarked to numerical
dataset_train['Embarked'] = dataset_train['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])
dataset_test['Embarked'] = dataset_test['Embarked'].apply(lambda x: {'S':0, 'C':1, 'Q':2}[x])

print(dataset_train.head(3))
# ----------RESULT----------
#   Survived  Pclass ...  Embarked
# PassengerId ...
# 1        0       3 ...         0
# 2        1       1 ...         1
# 3        1       3 ...         0
#
# [3 rows x 11 columns]

print(dataset_train.dtypes)
# ----------RESULT----------
# Survived     object
# Pclass        int64
# Name         object
# Sex           int64
# Age         float64
# SibSp         int64
# Parch         int64
# Ticket       object
# Fare        float64
# Cabin        object
# Embarked     object
# dtype: object

# ========== 10c ==========
# Create feature matrix and target vector
X = dataset_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
y = dataset_train['Survived']

print(X.shape, y.shape)
# ----------RESULT----------
# (889, 7) (889,)

print(X.head(3))
# ----------RESULT----------
#             Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# PassengerId
# 1                3    0  22.0      1      0   7.2500         0
# 2                1    1  38.0      1      0  71.2833         1
# 3                3    1  26.0      0      0   7.9250         0

print(y.head(3))
# ----------RESULT----------
# PassengerId
# 1    0
# 2    1
# 3    1
# Name: Survived, dtype: object

# ========== 10d ==========
# Normalize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

print(X[:3])
# ----------RESULT----------
# [[ 0.827377 -0.737695 -0.592481  0.432793 -0.473674 -0.502445 -1.610691]
# [-1.566107  1.355574  0.638789  0.432793 -0.473674  0.786845  0.813381]
# [ 0.827377  1.355574 -0.284663 -0.474545 -0.473674 -0.488854 -0.398655]]

# ========== 10e ==========
# Split into train and test
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# ----------RESULT----------
# (622, 7) (267, 7) (622,) (267,)

# ========== 10f ==========
# Create and train model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

print(model.coef_)
# ----------RESULT----------
# [[-0.92192  -2.52833  -0.37894  -0.29412  -0.07188  0.09474  -0.32127]]

print(model.intercept_)
# ----------RESULT----------
# [ 0.66667]

# ========== 10g ==========
# Predict and evaluate
y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(confusion_matrix(y_test, y_pred))
# ----------RESULT----------
# [[135  24]
# [ 32  76]]

print(classification_report(y_test, y_pred))
# ----------RESULT----------
#             precision    recall  f1-score   support
# 0           0.       0.       0.       0.
# 1           0.       0.       0.       0.
# avg / total 0.       0.       0.       0.

print(accuracy_score(y_test, y_pred))
# ----------RESULT----------
# 0.
```

```
# ========== 10h ==========
# Cross-validation
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print(scores)
# ----------RESULT----------
# [ 0.  0.  0.  0.  0.]

print(scores.mean())
# ----------RESULT----------
# 0.
```

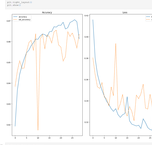
0x7f7f0b

	loss	accuracy	val_loss	val_accuracy
0	0.693147	0.500000	0.693147	0.500000
1	0.693147	0.500000	0.693147	0.500000
2	0.693147	0.500000	0.693147	0.500000
3	0.693147	0.500000	0.693147	0.500000
4	0.693147	0.500000	0.693147	0.500000

0x7f7f0b

```
plt.figure(figsize=(14, 5))
plt.subplot(1, 2)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
```



0x7f7f0b

```
# Predict on test data
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

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

```
313/313 [==============================] - 0s 1ms/step
Confusion Matrix:
[[   0 4961]
 [   0 5039]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4961
           1       0.50      1.00      0.67      5039

    accuracy                           0.50     10000
   macro avg       0.25      0.50      0.33     10000
weighted avg       0.25      0.50      0.33     10000
```

0x7f7f0b

```
# Calculate TP, TN, FP, FN
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]

# Calculate sensitivity (recall)
sensitivity = TP / (TP + FN)
print("Sensitivity:", sensitivity)

# Calculate specificity
specificity = TN / (TN + FP)
print("Specificity:", specificity)
```

```
Sensitivity: 1.0
Specificity: 0.0
```

0x7f7f0b

```
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

0x7f7f0b

```
# Calculate AUC
auc = roc_auc
print("AUC:", auc)
```

```
AUC: 0.5
```

0x7f7f0b

```
# Calculate the number of parameters in the model
trainable_params = model.count_trainable_params()
non_trainable_params = model.count_non_trainable_params()
total_params = trainable_params + non_trainable_params

print("Trainable parameters:", trainable_params)
print("Non-trainable parameters:", non_trainable_params)
print("Total parameters:", total_params)
```

```
Trainable parameters: 1
Non-trainable parameters: 0
Total parameters: 1
```

0x7f7f0b

```
# Save the model
model.save('cancer_model.h5')

# Load the saved model
from tensorflow.keras.models import load_model
loaded_model = load_model('cancer_model.h5')

# Evaluate the loaded model on the test data
loss, accuracy = loaded_model.evaluate(X_test, y_test)
print("Loaded model accuracy:", accuracy)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.5000
Loaded model accuracy: 0.5000
```

0x7f7f0b

```
# Predicted probabilities for the test data
y_pred_prob = loaded_model.predict(X_test)

# Convert probabilities to binary predictions (0 or 1)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Calculate TP, TN, FP, FN
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]

# Calculate sensitivity (recall)
sensitivity = TP / (TP + FN)
print("Sensitivity:", sensitivity)

# Calculate specificity
specificity = TN / (TN + FP)
print("Specificity:", specificity)
```

```
313/313 [==============================] - 0s 1ms/step
```