ml-intern

July 22, 2025

1 Github Repository

[ ]:
```
%cd https://github.com/zoya4477/ML-intern.git
!git clone
!git config --global user.email "zoyahafeez785@gmail.com"
!git config --global user.name "zoya4477"
```

```
[Errno 2] No such file or directory: 'https://github.com/zoya4477/ML-intern.git'
/content
fatal: You must specify a repository to clone.

usage: git clone [<options>] [--] <repo> [<dir>]

    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --progress            force progress reporting
    --reject-shallow      don't clone shallow repository
    -n, --no-checkout     don't create a checkout
    --bare                create a bare repository
    --mirror              create a mirror repository (implies bare)
    -l, --local           to clone from a local repository
    --no-hardlinks        don't use local hardlinks, always copy
    -s, --shared          setup as shared repository
    --recurse-submodules[=<pathspec>]
                          initialize submodules in the clone
    --recursive …       alias of --recurse-submodules
    -j, --jobs <n>        number of submodules cloned in parallel
    --template <template-directory>
                          directory from which templates will be used
    --reference <repo>    reference repository
    --reference-if-able <repo>
                          reference repository
    --dissociate          use --reference only while cloning
    -o, --origin <name>   use <name> instead of 'origin' to track upstream
    -b, --branch <branch>
                          checkout <branch> instead of the remote's HEAD
    -u, --upload-pack <path>
                          path to git-upload-pack on the remote
```

```
    --depth <depth>       create a shallow clone of that depth
    --shallow-since <time>
                          create a shallow clone since a specific time
    --shallow-exclude <revision>
                          deepen history of shallow clone, excluding rev
    --single-branch       clone only one branch, HEAD or --branch
    --no-tags             don't clone any tags, and make later fetches not to
follow them
    --shallow-submodules  any cloned submodules will be shallow
    --separate-git-dir <gitdir>
                          separate git dir from working tree
    -c, --config <key=value>
                          set config inside the new repository
    --server-option <server-specific>
                          option to transmit
    -4, --ipv4            use IPv4 addresses only
    -6, --ipv6            use IPv6 addresses only
    --filter <args>       object filtering
    --remote-submodules   any cloned submodules will use their remote-tracking
branch
    --sparse              initialize sparse-checkout file to include only files
at root
```

2 Project 1: Predict House Prices Using Linear Regression

Use a simple dataset to predict house prices based on features like area, number of rooms, and location using linear regression.

[ ]:
```python
#Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[ ]:
```python
#Load and Inspect Data
df = pd.read_csv('/content/archiv.zip')
print(df.head())
print(df.info())
```

```
   Id  Area  Bedrooms  Bathrooms  Floors  YearBuilt  Location  Condition  \
0   1  1360         5          4       3       1970  Downtown  Excellent
1   2  4272         5          4       3       1958  Downtown  Excellent
2   3  3592         2          2       3       1938  Downtown       Good
```

```
3   4   966         4          2       2       1902  Suburban       Fair
4   5  4926         1          4       2       1975  Downtown       Fair

  Garage   Price
0     No  149919
1     No  424998
2     No  266746
3    Yes  244020
4    Yes  636056
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Id         2000 non-null   int64
 1   Area       2000 non-null   int64
 2   Bedrooms   2000 non-null   int64
 3   Bathrooms  2000 non-null   int64
 4   Floors     2000 non-null   int64
 5   YearBuilt  2000 non-null   int64
 6   Location   2000 non-null   object
 7   Condition  2000 non-null   object
 8   Garage     2000 non-null   object
 9   Price      2000 non-null   int64
dtypes: int64(7), object(3)
memory usage: 156.4+ KB
None
```

[ ]:
```python
#Preprocess the Data

#Check for missing values
print(df.isnull().sum())
```

```
Id           0
Area         0
Bedrooms     0
Bathrooms    0
Floors       0
YearBuilt    0
Location     0
Condition    0
Garage       0
Price        0
dtype: int64
```

[ ]:
```python
df = pd.get_dummies(df, columns=['Location', 'Condition', 'Garage'],␣
↪drop_first=True)
```

[ ]:
```
#Split data
X = df.drop('Price', axis=1)
y = df['Price']

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

[ ]:
```
#Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

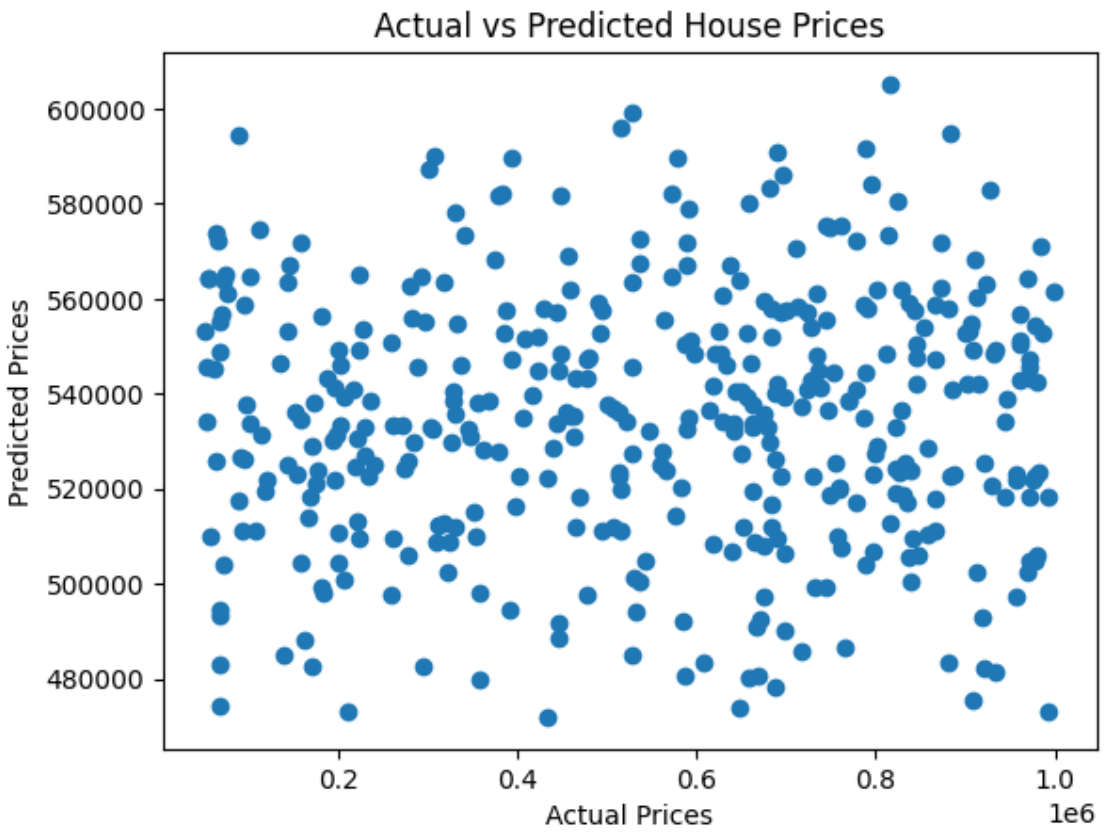
[ ]:
```
LinearRegression()
```

[ ]:
```
#Make Prediction
y_pred = model.predict(X_test)
```

[ ]:
```
#Evaluate the Model
print("R² Score:", r2_score(y_test, y_pred))
print("MAE:", mean_absolute_error(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
R² Score: -0.006181784611834162
MAE: 242867.44926338625
RMSE: 279785.21069002635
```

[ ]:
```
#Visulaize Result
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.show()
```



[ ]:

3 Project 2: Iris Flower Classification

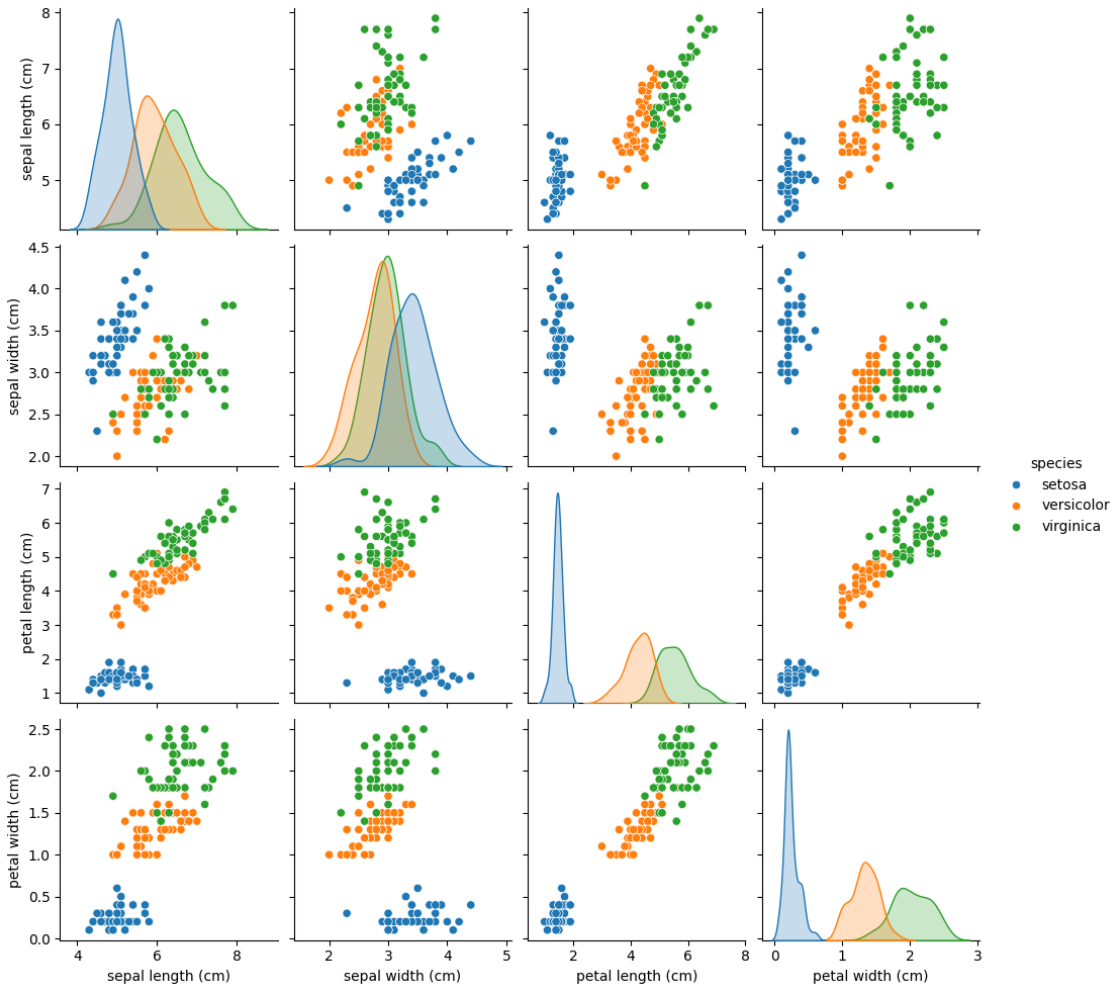
Build a classification model using the Iris dataset to predict the species of a flower based on its features (sepal and petal dimensions).

[ ]:
```
#Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
```

[ ]:
```
#Load the Dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2:␣
 ↪'virginica'})
print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa
```

[ ]:
```
#Visualize the Data
sns.pairplot(df, hue='species')
plt.show()
```



[ ]:
```
#Prepare the Data
X = df.drop('species', axis=1)
y = df['species']

# Standardize the features(Optional)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
#Train Classification Model
# model = LogisticRegression()
# model.fit(X_train, y_train)
```

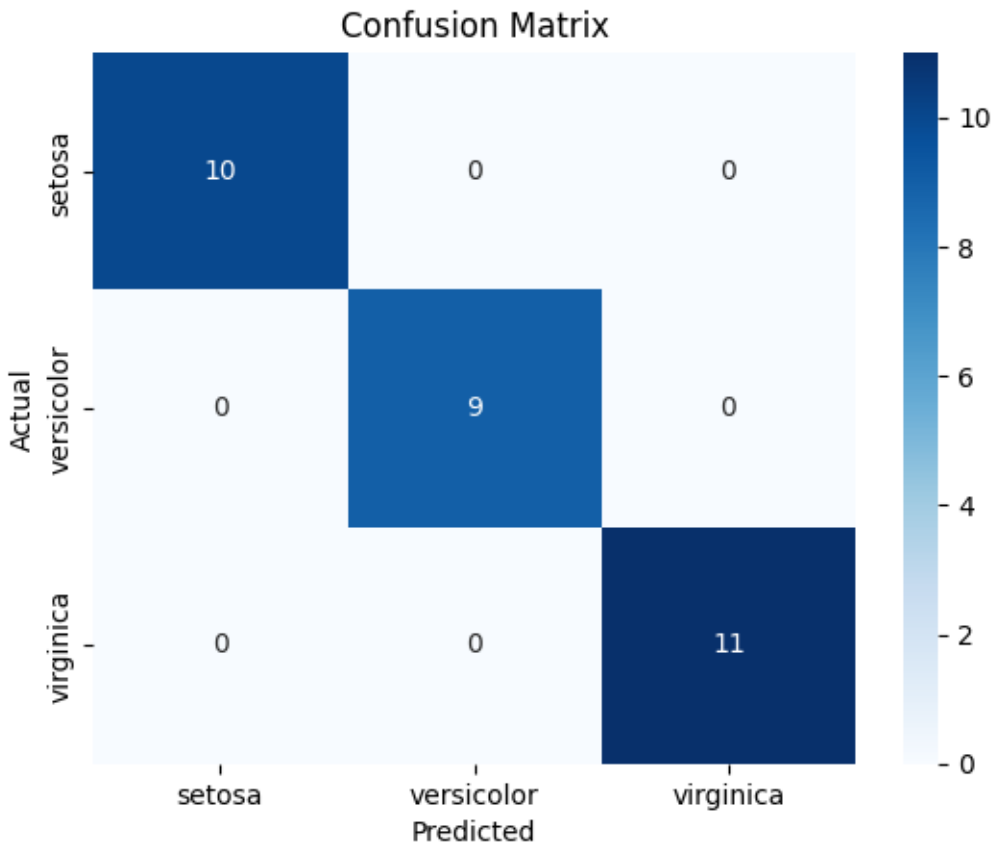
[ ]:
```
#Train Classification Model
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

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

[ ]:
```
#Evaluate the Model
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

[ ]:
```
#Visualize Confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[ ]:

4 Project 3: Sentiment Analysis on Tweets

Perform sentiment analysis on a dataset of tweets to classify them as positive, negative, or neutral using Natural Language Processing (NLP).

[ ]:
```
#Libraries
import pandas as pd
import numpy as np
import re
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix
```

[ ]:
```
#Load Dataset
df = pd.read_csv("/content/Tweets.csv")
print(df.head())
```

```
             tweet_id airline_sentiment  airline_sentiment_confidence  \
0  570306133677760513           neutral                        1.0000
1  570301130888122368          positive                        0.3486
2  570301083672813571           neutral                        0.6837
3  570301031407624196          negative                        1.0000
4  570300817074462722          negative                        1.0000

  negativereason  negativereason_confidence         airline  \
0            NaN                        NaN  Virgin America
1            NaN                     0.0000  Virgin America
2            NaN                        NaN  Virgin America
3     Bad Flight                     0.7033  Virgin America
4     Can't Tell                     1.0000  Virgin America

  airline_sentiment_gold        name negativereason_gold  retweet_count  \
0                    NaN     cairdin                 NaN              0
1                    NaN    jnardino                 NaN              0
2                    NaN  yvonnalynn                 NaN              0
3                    NaN    jnardino                 NaN              0
4                    NaN    jnardino                 NaN              0

                                                text tweet_coord  \
0                @VirginAmerica What @dhepburn said.         NaN
1  @VirginAmerica plus you've added commercials t…         NaN
2  @VirginAmerica I didn't today… Must mean I n…         NaN
3  @VirginAmerica it's really aggressive to blast…         NaN
4  @VirginAmerica and it's a really big bad thing…         NaN

              tweet_created tweet_location               user_timezone
0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US & Canada)
1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US & Canada)
2  2015-02-24 11:15:48 -0800      Lets Play  Central Time (US & Canada)
3  2015-02-24 11:15:36 -0800            NaN  Pacific Time (US & Canada)
4  2015-02-24 11:14:45 -0800            NaN  Pacific Time (US & Canada)
```

[ ]:
```
#Text Preprocessing
# Select relevant columns and drop missing values
df = df[['text', 'airline_sentiment']].dropna()

# Clean the text
def clean_text(text):
  text = text.lower() # Convert to lowercase
  text = re.sub(r'http\S+', '', text) # Remove URLs
```

```
  text = re.sub(r'@\w+', '', text) # Remove mentions
  text = re.sub(r'#', '', text) # Remove hashtags
  text = re.sub(r'[^\w\s]', '', text) # Remove punctuation
  return text

df['cleaned_text'] = df['text'].apply(clean_text)
```

[ ]:
```
#vectorization
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['cleaned_text'])
y = df['airline_sentiment']
```

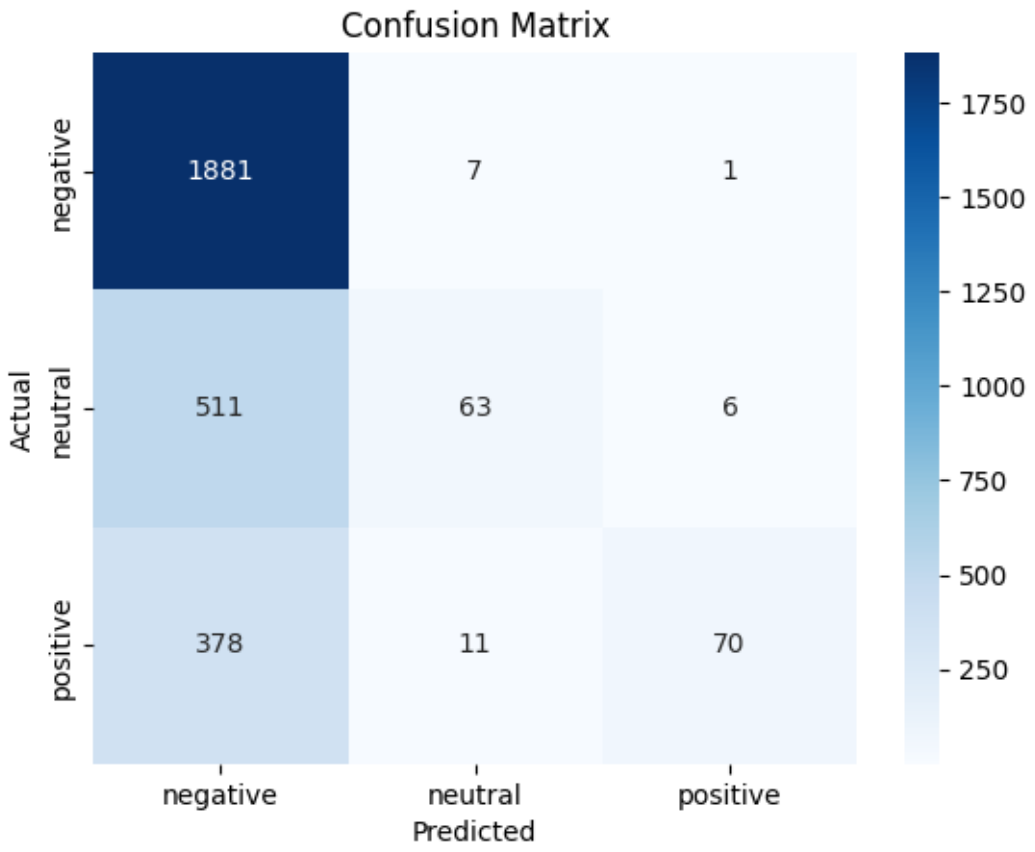
[ ]:
```
#split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
#Train the Model
model = MultinomialNB()
model.fit(X_train, y_train)
```

[ ]:
```
MultinomialNB()
```

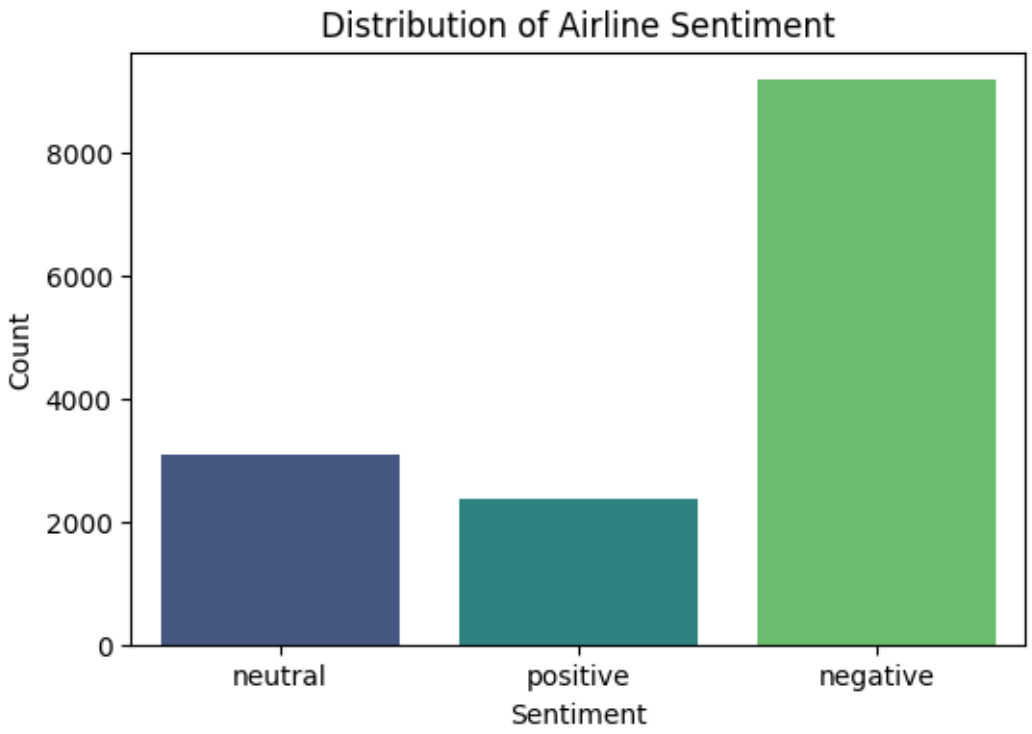
[ ]:
```
#Evaluate the model
y_pred = model.predict(X_test)

print("Classification Report:\n", classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_,␣
 ↪yticklabels=model.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.68      1.00      0.81      1889
     neutral       0.78      0.11      0.19       580
    positive       0.91      0.15      0.26       459

    accuracy                           0.69      2928
   macro avg       0.79      0.42      0.42      2928
weighted avg       0.73      0.69      0.60      2928
```



[ ]:
```
import matplotlib.pyplot as plt
#Visualize the sentiment distribution
plt.figure(figsize=(6,4))
sns.countplot(x='airline_sentiment', data=df, palette='viridis')
plt.title('Distribution of Airline Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```

```
/tmp/ipython-input-28-1360570956.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='airline_sentiment', data=df, palette='viridis')
```



5 Project 4: Handwritten Digit Recognition Using MNIST

Use the MNIST dataset to train a neural network for recognizing handwritten digits.

[ ]:
```
#Libraries
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

[ ]:
```
#MNIST (Built into Keras)
from keras.datasets import mnist
```

[ ]:
```
#Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434            2s
0us/step
```

[ ]:
```
#Preprocess
# Normalize the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

[ ]:
```
#Build Neural Network
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

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

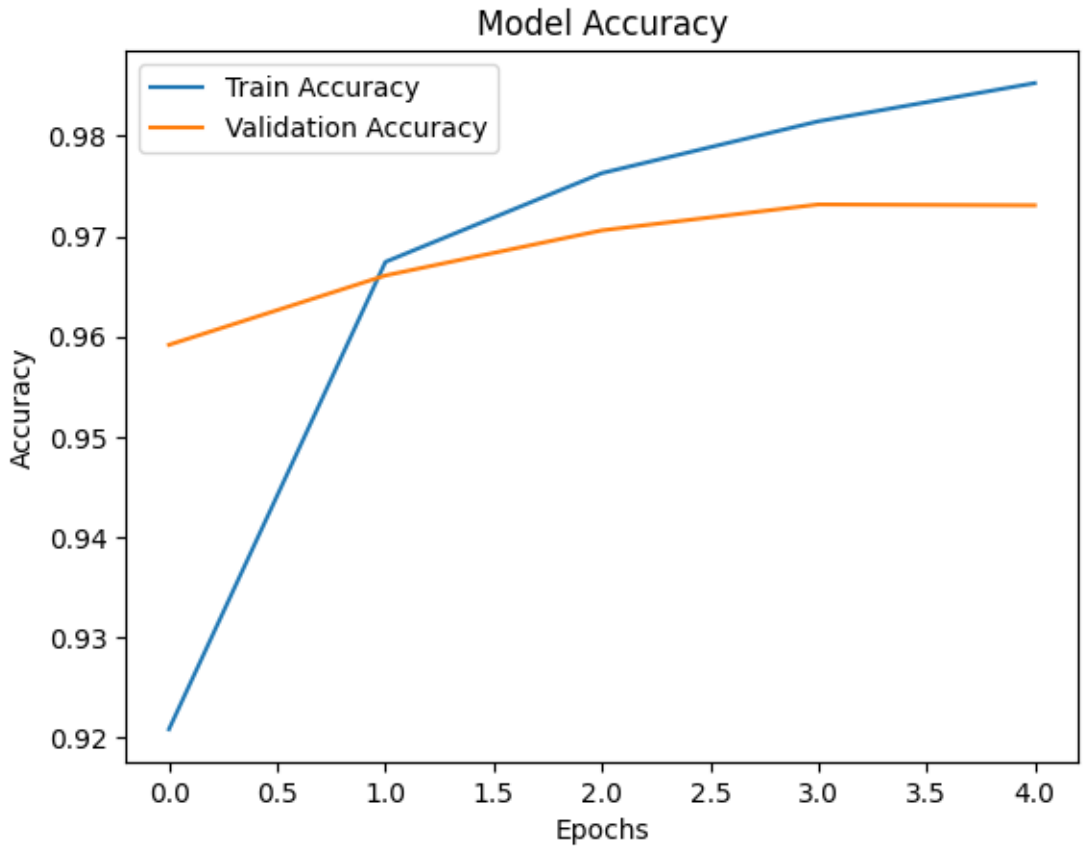
[ ]:
```
#Train the model
history = model.fit(x_train, y_train, epochs=5, validation_split=0.2)
```

```
Epoch 1/5
1500/1500            7s 3ms/step -
accuracy: 0.8623 - loss: 0.4687 - val_accuracy: 0.9592 - val_loss: 0.1365
Epoch 2/5
1500/1500            5s 3ms/step -
accuracy: 0.9657 - loss: 0.1164 - val_accuracy: 0.9661 - val_loss: 0.1112
Epoch 3/5
1500/1500            4s 3ms/step -
accuracy: 0.9776 - loss: 0.0744 - val_accuracy: 0.9706 - val_loss: 0.0978
Epoch 4/5
1500/1500            5s 3ms/step -
accuracy: 0.9827 - loss: 0.0529 - val_accuracy: 0.9732 - val_loss: 0.0903
Epoch 5/5
1500/1500            5s 4ms/step -
accuracy: 0.9858 - loss: 0.0433 - val_accuracy: 0.9731 - val_loss: 0.0993
```

[ ]:
```
#Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc}")
```

```
313/313             1s 3ms/step -
accuracy: 0.9700 - loss: 0.1031
Test accuracy: 0.9735999703407288
```

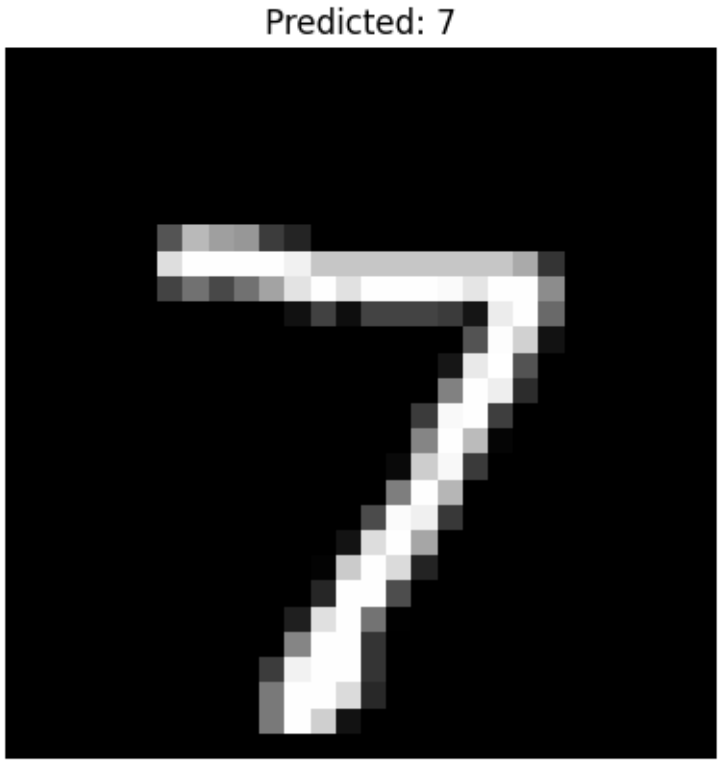
[ ]:
```
#Plot accuracy and Loss
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[ ]:
```
#prediction
predictions = model.predict(x_test)
plt.imshow(x_test[0], cmap='gray')
```

```
plt.title(f"Predicted: {np.argmax(predictions[0])}")
plt.axis('off')
plt.show()
```

```
313/313             1s 2ms/step
```



[ ]:

6 Project 5: Customer Churn Prediction

Build a machine learning model to predict whether a customer will churn based on behavioral and demographic data.

[ ]:
```
#Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
```

[ ]:
```
#Load dataset
df = pd.read_csv("/content/churn detection.zip")
print(df.head())
```

```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
0  No phone service             DSL             No  …               No
1                No             DSL            Yes  …              Yes
2                No             DSL            Yes  …               No
3  No phone service             DSL            Yes  …              Yes
4                No     Fiber optic             No  …               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

[ ]:
```
#Preprocessing
# Drop customer ID column
df.drop('customerID', axis=1, inplace=True)

# Convert 'TotalCharges' to numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Handle missing values
df.dropna(inplace=True)
```

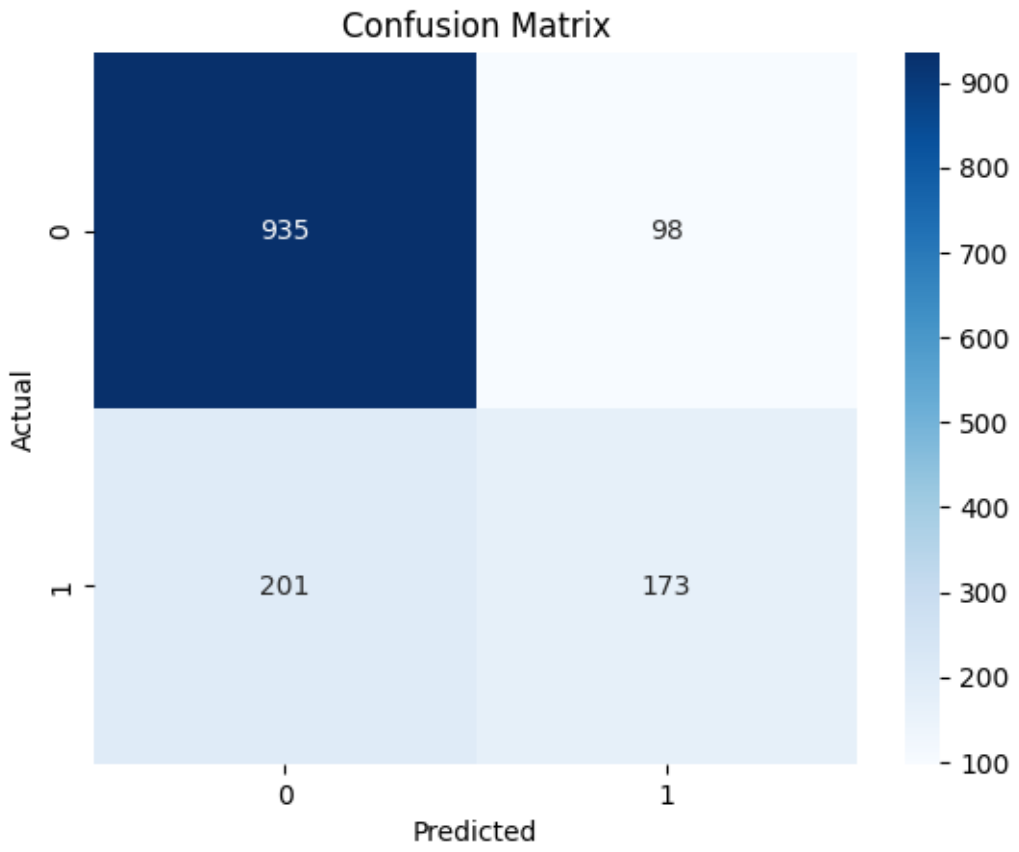
```
# Encode binary categorical columns
binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService',␣
 ↪'PaperlessBilling', 'Churn']
for col in binary_cols:
    df[col] = df[col].map({'Yes': 1, 'No': 0, 'Female': 0, 'Male': 1})

# One-hot encode remaining categorical variables
df = pd.get_dummies(df, drop_first=True)
```

[ ]:
```
#Split and Scale Data
X = df.drop('Churn', axis=1)
y = df['Churn']

scaler = StandardScaler()
X = scaler.fit_transform(X)

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

[ ]:
```
#Train Data
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

[ ]:
```
#Evaluate Model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.7874911158493249
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.91      0.86      1033
           1       0.64      0.46      0.54       374

    accuracy                           0.79      1407
   macro avg       0.73      0.68      0.70      1407
weighted avg       0.77      0.79      0.78      1407
```



7 Project 6: Image Classification Using Transfer Learning

Use a pre-trained convolutional neural network (like ResNet or VGG16) to classify images from a custom dataset.

[ ]:
```
#Libraries
import os
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG16
```

[ ]:
```
#Download the Dataset
!wget https://github.com/Horea94/Fruit-Images-Dataset/archive/refs/heads/master.
 ↪zip
!unzip -q master.zip
!mv Fruit-Images-Dataset-master Fruit_360
```

```
--2025-07-22 14:22:53--  https://github.com/Horea94/Fruit-Images-
Dataset/archive/refs/heads/master.zip
Resolving github.com (github.com)… 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://codeload.github.com/Horea94/Fruit-Images-
Dataset/zip/refs/heads/master [following]
--2025-07-22 14:22:53--  https://codeload.github.com/Horea94/Fruit-Images-
Dataset/zip/refs/heads/master
Resolving codeload.github.com (codeload.github.com)… 20.205.243.165
Connecting to codeload.github.com (codeload.github.com)|20.205.243.165|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: unspecified [application/zip]
Saving to: ‘master.zip’

master.zip              [   <=>              ] 761.30M  17.4MB/s    in 46s

2025-07-22 14:23:39 (16.7 MB/s) - ‘master.zip’ saved [798281972]
```

[ ]:
```
#Check Directory
import os

# List classes in training set
train_dir = 'Fruit_360/Training'
print("Classes in training set:", os.listdir(train_dir)[:10])
```

```
Classes in training set: ['Fig', 'Peach Flat', 'Cherry 1', 'Hazelnut', 'Tomato
not Ripened', 'Pear Abate', 'Apple Red 3', 'Grape Blue', 'Pear 2', 'Pear
Monster']
```

[ ]:
```
#Load the Dataset Using ImageDataGenerator

# Setup data generators
train_path = 'Fruit_360/Training'
test_path = 'Fruit_360/Test'

train_gen = ImageDataGenerator(rescale=1./255)
test_gen = ImageDataGenerator(rescale=1./255)
```

```
train_data = train_gen.flow_from_directory(train_path,
                                           target_size=(100, 100),
                                           class_mode='categorical',
                                           batch_size=32)

test_data = test_gen.flow_from_directory(test_path,
                                         target_size=(100, 100),
                                         class_mode='categorical',
                                         batch_size=32)
```

```
Found 67692 images belonging to 131 classes.
Found 22688 images belonging to 131 classes.
```

[ ]:
```
#Transfer Learning

# Load base model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(100,␣
 ↪100, 3))
for layer in base_model.layers:
    layer.trainable = False

# Create model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dense(train_data.num_classes, activation='softmax')
])

model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train model
model.fit(train_data, validation_data=test_data, epochs=3)
```

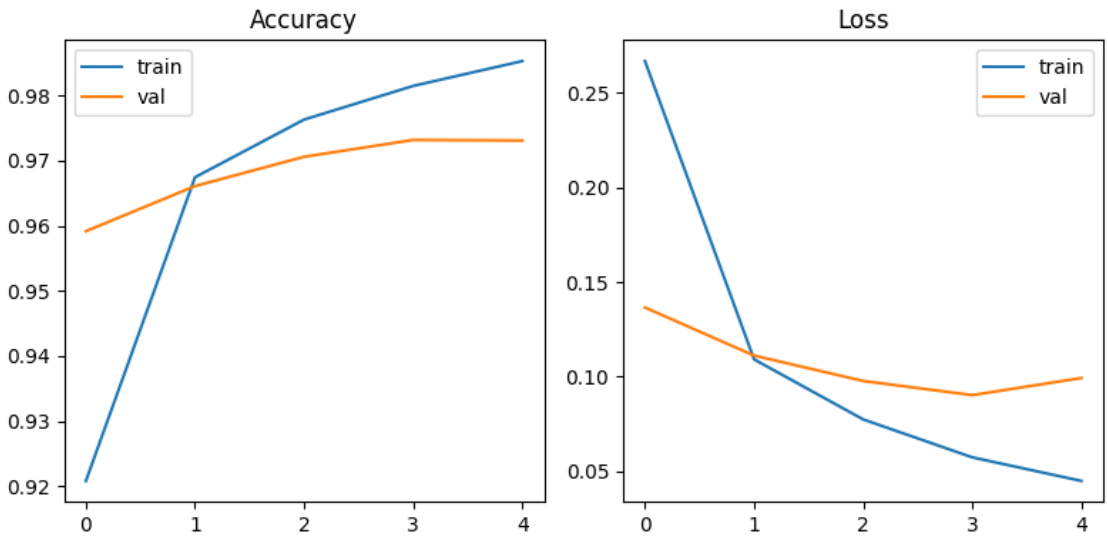
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256             4s
0us/step
Epoch 1/3
/usr/local/lib/python3.11/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
2116/2116              142s 64ms/step
- accuracy: 0.5301 - loss: 2.3444 - val_accuracy: 0.8400 - val_loss: 0.6655
Epoch 2/3
2116/2116              142s 67ms/step
- accuracy: 0.9644 - loss: 0.2081 - val_accuracy: 0.8978 - val_loss: 0.4110
Epoch 3/3
2116/2116              143s 68ms/step
- accuracy: 0.9905 - loss: 0.0704 - val_accuracy: 0.9272 - val_loss: 0.2969
```

[ ]:
```
<keras.src.callbacks.history.History at 0x7e3469df1d10>
```

[ ]:
```python
#Evaluation and Plot
loss, acc = model.evaluate(test_data)
print(f"\nValidation accuracy: {acc:.3f}")

plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.title('Accuracy'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.title('Loss'); plt.legend()
plt.tight_layout(); plt.show()
```

```
709/709              36s 51ms/step -
accuracy: 0.9270 - loss: 0.2981

Validation accuracy: 0.927
```



[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

#Prediction on a sample image

# Get a batch of test images and labels
test_images, test_labels = next(test_data)

# Choose a random image from the batch
img_index = np.random.randint(0, len(test_images))
sample_image = test_images[img_index]
true_label_one_hot = test_labels[img_index]
true_label_index = np.argmax(true_label_one_hot)
true_label_name = list(test_data.class_indices.keys())[true_label_index]

# Reshape the image to add batch dimension for prediction
sample_image_input = np.expand_dims(sample_image, axis=0)

# Make a prediction
prediction = model.predict(sample_image_input)
predicted_label_index = np.argmax(prediction)
predicted_label_name = list(test_data.class_indices.
 ↪keys())[predicted_label_index]
confidence = np.max(prediction) * 100

# Display the image and prediction
plt.imshow(sample_image)
plt.title(f"True: {true_label_name}\nPredicted: {predicted_label_name}␣
 ↪({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
1/1             2s 2s/step
```

True: Cantaloupe 2
Predicted: Cantaloupe 2 (99.86%)