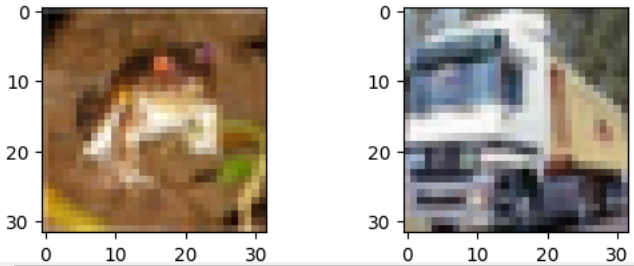
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
(train_images,train_label),(test_images,test_label)=cifar10.load_data()
print('Training data shape',train_images.shape)
print('Training label shape',train_label.shape)
print('Testing data shape',test_images.shape)
print('Testing label shape',test_label.shape)
plt.subplot(221)
plt.imshow(train_images[0],cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(train_images[1],cmap=plt.get_cmap('gray'))
```

```
Training data shape (50000, 32, 32, 3)
Training label shape (50000, 1)
Testing data shape (10000, 32, 32, 3)
Testing label shape (10000, 1)
<matplotlib.image.AxesImage at 0x7ac63946e0e0>
```




```
import cv2 as cv
# importing test images
image1 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image1.jpeg")
image2 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image2.jpeg")
image3 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image3.jpeg")
image4 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image4.jpeg")
image5 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image5.jpeg")
image6 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image6.jpeg")
image7 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image7.jpeg")
image8 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image8.jpeg")
image9 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image9.jpeg")
image10 = cv.imread("/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image10.jpeg")

validation_images = [image1,image2,image3,image4,image5,image6,image7,image8,image9,image10]


# Filter "Automobile" class images (label 1)
automobile_images = train_images[train_label.flatten() == 1]
automobile_labels = np.ones(len(automobile_images))

# Select an equal number of images from another class, e.g., "Airplane" (label 0)
non_automobile_class = 0  # Change this to any class you prefer
non_automobile_images = train_images[train_label.flatten() == non_automobile_class][:len(automobile_images)]
non_automobile_labels = np.zeros(len(non_automobile_images))

# Combine the images and labels into a single dataset
binary_images = np.vstack((automobile_images, non_automobile_images))
binary_labels = np.hstack((automobile_labels, non_automobile_labels))


# Print the shape of the new dataset
print('Binary dataset shape:', binary_images.shape)
print('Binary labels shape:', binary_labels.shape)
```

```
Binary dataset shape: (10000, 32, 32, 3)
Binary labels shape: (10000,)
```

```
# Shuffle the dataset
indices = np.random.permutation(len(binary_images))
binary_images = binary_images[indices]
binary_labels = binary_labels[indices]

# Flatten the images for classification algorithms
n_samples = binary_images.shape[0]
binary_images_flat = binary_images.reshape(n_samples, -1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(binary_images_flat, binary_labels, test_size=0.2, random_state=42)

# Initialize the classifiers
classifiers = {
    "Support Vector Classifier (SVC)": SVC(),
    "k-Nearest Neighbors (kNN)": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=10000)
}


# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.2f}")

# Plot some of the test images and their predicted labels
plt.figure(figsize=(10, 2))
for i in range(10):
    plt.subplot(1, 10, i + 1)
    plt.imshow(X_test[i].reshape(32, 32, 3))
    plt.axis('off')
    plt.title(f"Pred: {y_pred[i]}")
plt.show()
```

```
Support Vector Classifier (SVC) Accuracy: 0.90
k-Nearest Neighbors (kNN) Accuracy: 0.67
Decision Tree Accuracy: 0.76
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge
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(
Logistic Regression Accuracy: 0.71
```


```
import pandas as pd
# Load and preprocess the validation images
validation_images = []
for i in range(1, 11):
    img = cv.imread(f"/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/binaryclassimages/image{i}.jpeg")
    img_resized = cv.resize(img, (32, 32))  # Resize to 32x32 pixels
    validation_images.append(img_resized)

# Convert the list to a numpy array and flatten the images
validation_images = np.array(validation_images)
validation_images_flat = validation_images.reshape(len(validation_images), -1)

# Create a DataFrame to store results
results = {
    'Classifier': [],
    'Automobile Count': [],
    'Non-Automobile Count': [],
    'Accuracy': []
}

# Assuming the true labels for the validation set (you can replace with actual labels if available)
true_labels = np.array([1, 1, 1, 1, 1, 1, 1, 1, 0, 0])  # Replace this with actual validation labels if known
```

```
# Iterate over each classifier and make predictions
for name, clf in classifiers.items():
    predictions = clf.predict(validation_images_flat)

    # Count the number of "Automobile" and "Non-Automobile" predictions
    automobile_count = np.sum(predictions == 1)
    non_automobile_count = np.sum(predictions == 0)

    # Calculate accuracy (if true labels are available)
    accuracy = accuracy_score(true_labels, predictions)

    # Append results to the DataFrame
    results['Classifier'].append(name)
    results['Automobile Count'].append(automobile_count)
    results['Non-Automobile Count'].append(non_automobile_count)
    results['Accuracy'].append(accuracy)

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Display the results
results_df
```

	Classifier	Automobile Count	Non-Automobile Count	Accuracy
0	Support Vector Classifier (SVC)	9	1	0.9
1	k-Nearest Neighbors (kNN)	5	5	0.7
2	Decision Tree	8	2	0.8
3	Logistic Regression	7	3	0.5

Next steps: Generate code with `results_df` View recommended plots New interactive sheet

Start coding or generate with AI.

Key Observations:

- **Support Vector Classifier (SVC)**:
  - High accuracy (90%) with a strong bias towards predicting "Automobile" (9 out of 10 images).
- **k-Nearest Neighbors (kNN)**:
  - Balanced predictions between "Automobile" and "Non-Automobile," but lower accuracy (70%) compared to SVC.
- **Decision Tree**:
  - Good accuracy (80%) with a slight bias towards predicting "Automobile."
- **Logistic Regression**:
  - Lowest accuracy (50%), indicating it may struggle with this classification task and might be making near-random predictions.