IMAGE PROCESSING

Image processing refers to the manipulation and analysis of digital images using computer algorithms. It involves various techniques and operations to enhance, transform, or extract useful information from images. Here are the simplified steps involved in image processing:

Image Acquisition:

The process of capturing or obtaining the image using devices such as cameras or scanners.

Preprocessing:

- Remove noise: Apply filters or denoising techniques to reduce unwanted artifacts or disturbances in the image.
- Resize and crop: Adjust the size or aspect ratio of the image or focus on specific regions of interest.
- Correct color balance: Normalize or adjust the color levels to improve the overall appearance of the image.
- Normalize intensity: Enhance or equalize the brightness and contrast of the image.

Segmentation:

Divide the image into meaningful or homogeneous regions based on properties such as color, texture, or intensity. Techniques include thresholding, edge detection, region growing, or clustering algorithms.

Feature Extraction:

Identify and extract relevant features from the segmented regions. Features can include shape, texture, color, or other characteristics that represent meaningful information for further analysis.

Image Enhancement:

Improve the visual quality or interpretability of the image. Techniques include sharpening, smoothing, histogram equalization, or contrast adjustment.

Object Recognition and Classification:

Identify and classify objects or patterns in the image based on extracted features. This can involve the use of machine learning algorithms or pattern recognition techniques.

Image Analysis and Interpretation:

Analyze the processed image to extract meaningful information or make decisions based on the desired application. This can include object tracking, image registration, object counting, or measurement of object properties.

Image Visualization and Presentation:

Display or represent the processed image and its analysis results in a suitable format for visualization, interpretation, or communication purposes. This can include displaying images, graphs, histograms, or annotated images.

In [1]:
```
from IPython.display import Image, display

display(Image(url='https://image.slidesharecdn.com/imageprocessing-141114231629-conversion-gate01/95/digital-image-pr
```

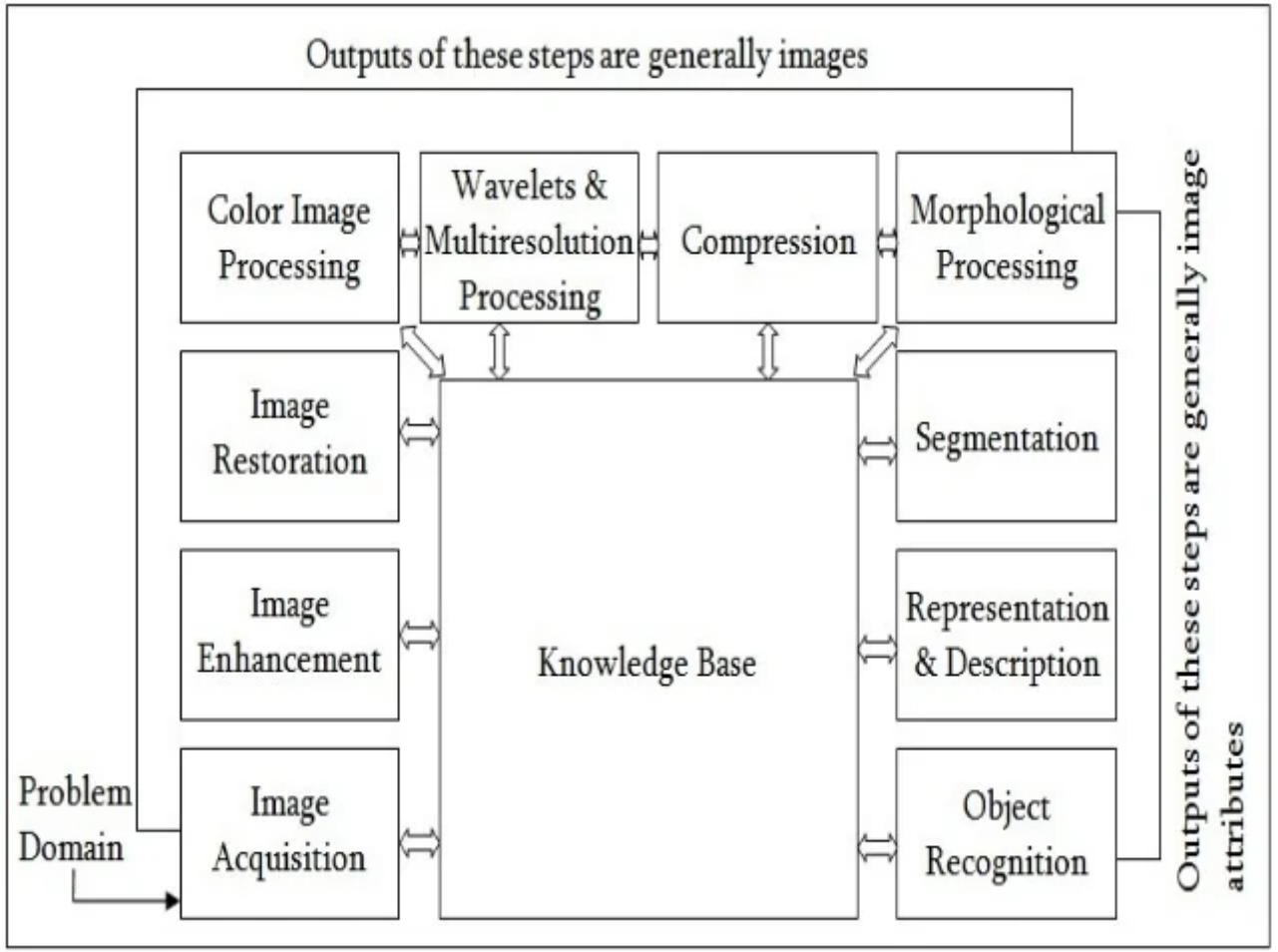


Image Data - Recognising Handwritten Alphabets

Dataset Link - mnist.zip

SPRINT 1 - Create DataFrame from raw ImageFiles

- Description MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. In this SPRINT, your goal is to preprocess the image files given to you and create a pandas dataframe.

Task A - Download the ‘mnist_data.zip’ and read the data in a pandas dataframe.

Task B - Use your Data Engineering skills to create a dataframe which can annotate each image into one of the 26 classes

Import necessary libraries:

In [2]:
```
#Import the NumPy library for numerical computations.
import numpy as np
#Import the Pandas library for data manipulation and analysis.
import pandas as pd
#Import the Matplotlib library for data visualization.
import matplotlib.pyplot as plt
#Import the Seaborn library for statistical data visualization.
import seaborn as sns
#Enable inline plotting in Jupyter Notebook.
%matplotlib inline
#Import the Warnings module to manage warning messages.
import warnings
warnings.filterwarnings("ignore")
#Import the OS module for interacting with the operating system.
import os
```

Extracting MNIST Dataset from ZIP File and Counting Files:

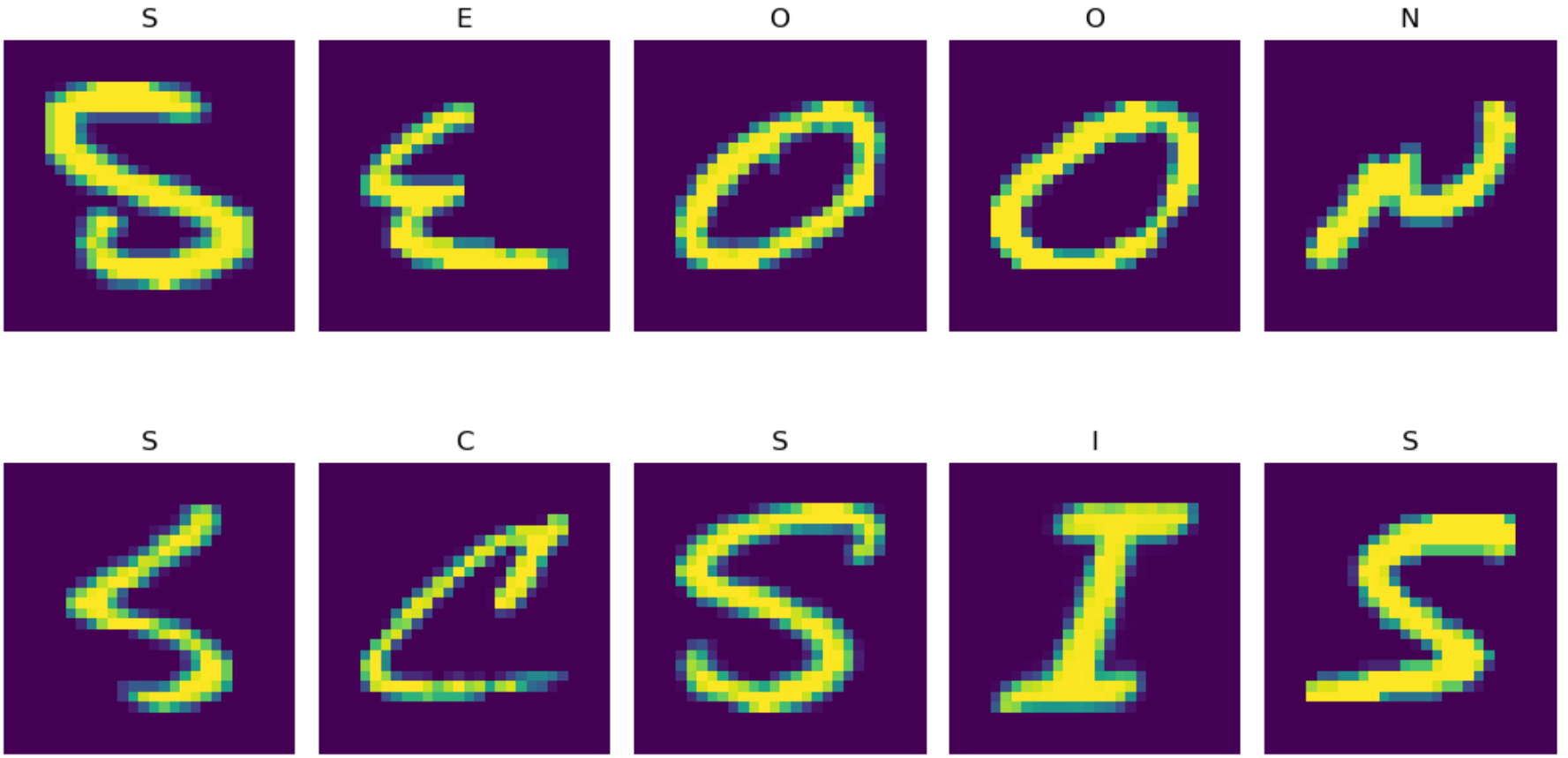
In [3]:
```
#Import the ZipFile class from the zipfile module to work with ZIP files.
import zipfile
#Open the zip file specified by the given path in read mode using the ZipFile class
#Extract all the contents of the zip file to the specified directory.
with zipfile.ZipFile(r'C:\Users\pamar\Downloads\New folder\mnist.zip') as zip_ref:
    zip_ref.extractall('C:/Users/pamar/Downloads/New folder/')
# Get the list of file names present in the specified directory using the listdir() function from the OS module.
file_names = os.listdir('C:/Users/pamar/Downloads/New folder/New folder')
#Print the total number of files in the dataset:
print('Total number of files in the dataset:', len(file_names))
```

```
Total number of files in the dataset: 26
```

Displaying Random Images and their Labels from a Directory:

In [4]:

```
from PIL import Image
import os
import random
import matplotlib.pyplot as plt

directory = 'C:/Users/pamar/Downloads/New folder/New folder'
# list comprehension
images = []
labels = []

for root, dirs, files in os.walk(directory):
    for file in files:
        if file.endswith('.jpg') or file.endswith('.png'):
            image_path = os.path.join(root, file)
            with Image.open(image_path) as img:
                images.append(img.copy())
            labels.append(file[0])

# Randomly select 10 images
random_images = random.sample(images, 10)
random_labels = [labels[images.index(img)] for img in random_images]

# Plot and show the randomly selected images
fig, axs = plt.subplots(2, 5, figsize=(10, 6))
axs = axs.flatten()

for i in range(len(random_images)):
    axs[i].imshow(random_images[i])
    axs[i].axis('off')
    axs[i].set_title(random_labels[i])

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



In [5]:
```
# check the length of images
len(images)
```

Out[5]: 372451

In [6]:
```
# show the images size
np.array(images[0]).size
```

Out[6]: 784

In [7]:
```
np.array(images[0])
```

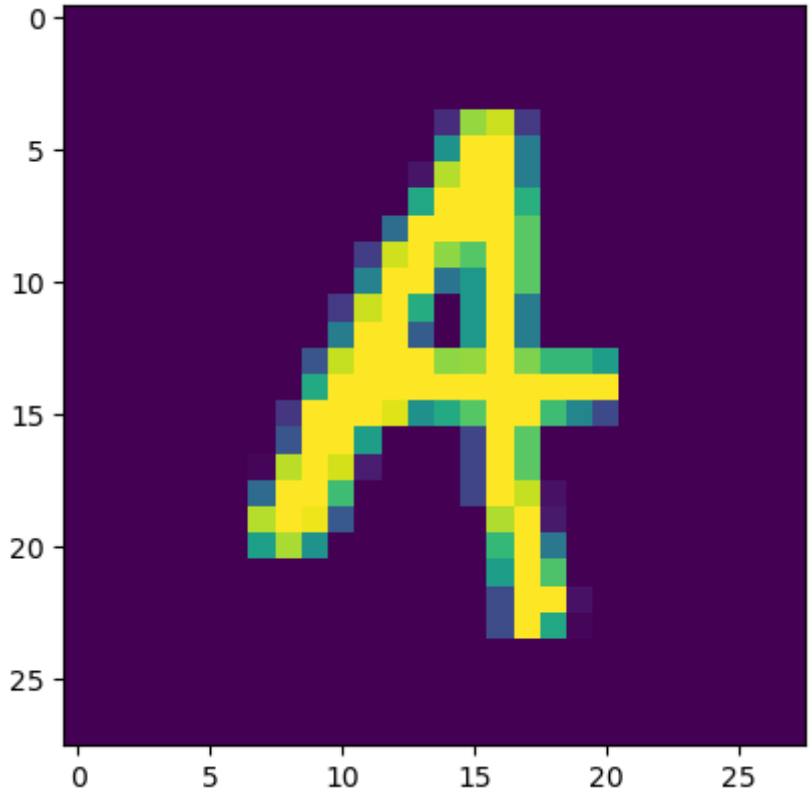
```
Out[7]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  32, 215, 235,  43,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 130, 255, 255, 107,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         14, 227, 255, 255, 107,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        152, 255, 255, 255, 162,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  91,
        255, 255, 255, 255, 190,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  47, 237,
        255, 212, 188, 255, 190,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 113, 255,
        255,  97, 136, 255, 190,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  43, 235, 255,
        158,   0, 136, 255, 107,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 107, 255, 255,
         75,   0, 136, 255, 107,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  67, 233, 255, 255,
        255, 212, 215, 255, 206, 170, 170, 142,   0,   0,   0,   0,   0,
```

```
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0, 156, 255, 255, 255,
 255, 255, 255, 255, 255, 255, 255, 255,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  40, 255, 255, 255, 243,
 128, 156, 188, 255, 255, 176, 117,  57,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  67, 255, 255, 142,   0,
   0,   0,  53, 255, 190,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   4, 229, 255, 239,  20,   0,
   0,   0,  53, 255, 190,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  89, 255, 255, 176,   0,   0,
   0,   0,  53, 255, 233,  12,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0, 227, 255, 249,  71,   0,   0,
   0,   0,   0, 225, 255,  18,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0, 144, 223, 130,   0,   0,   0,
   0,   0,   0, 170, 255, 101,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0, 142, 255, 184,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  59, 255, 255,  12,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  59, 255, 152,   4,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
       0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       0,   0]], dtype=uint8)
```

In [8]:
```
# show the images
plt.imshow(images[0])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x148dc19dfc0>
```



In [ ]:

In [9]:
```
# converts images by 15*15
image_resized=[]
for i in images:
   i.thumbnail((15,15))
   image_resized.append(i)
```

In [10]:
```
# check length of resized image
len(image_resized)
```

Out[10]: 372451

In [11]:
```
# check size of resized image
np.array(image_resized[0]).size
```

Out[11]: 225

Task A - Perform data preprocessing on the given image data and convert it into numerical vectors.

In [12]:
```
# converts the images into numerical by using flatten
flatten_list = [list(np.array(i).flatten()) for i in image_resized]
```

In [13]:
```
# create a data frame
letter_df = pd.DataFrame(flatten_list)
```

In [14]:
```
# assign labels to label column
letter_df['label'] = labels
```

In [15]:
```
# head of dataframe
letter_df.head()
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	216	217	218	219	220	221	222	223	224	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A

5 rows × 226 columns

In [16]:
```
# tail
letter_df.tail()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	...	216	217	218	219	220	221	222	223	224	label
372446	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Z
372447	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Z
372448	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Z
372449	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Z
372450	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Z

5 rows × 226 columns

In [17]:
```
## checking the value counts of columns
letter_df[8].value_counts()
```

Out[17]:
```
0     372449
18         1
9          1
Name: 8, dtype: int64
```

In [18]:
```
# to _csv file to save
letter_df.to_csv('letters.csv')
```

In [19]:
```
# read the csv file
A_Z_df = pd.read_csv(r"C:\Users\pamar\Downloads\New folder\letters.csv")
```

In [20]:
```
# drop the uncessary column
A_Z_df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [21]:
```
# head
A_Z_df.head()
```

Out[21]:

	0	1	2	3	4	5	6	7	8	9	...	216	217	218	219	220	221	222	223	224	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	A

5 rows × 226 columns

In [22]:
```
# detail of data
A_Z_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 372451 entries, 0 to 372450
Columns: 226 entries, 0 to label
dtypes: int64(225), object(1)
memory usage: 642.2+ MB
```

In [23]:
```
# shape of data
A_Z_df.shape
```

Out[23]:
```
(372451, 226)
```

DATASET IS UNBALANCED

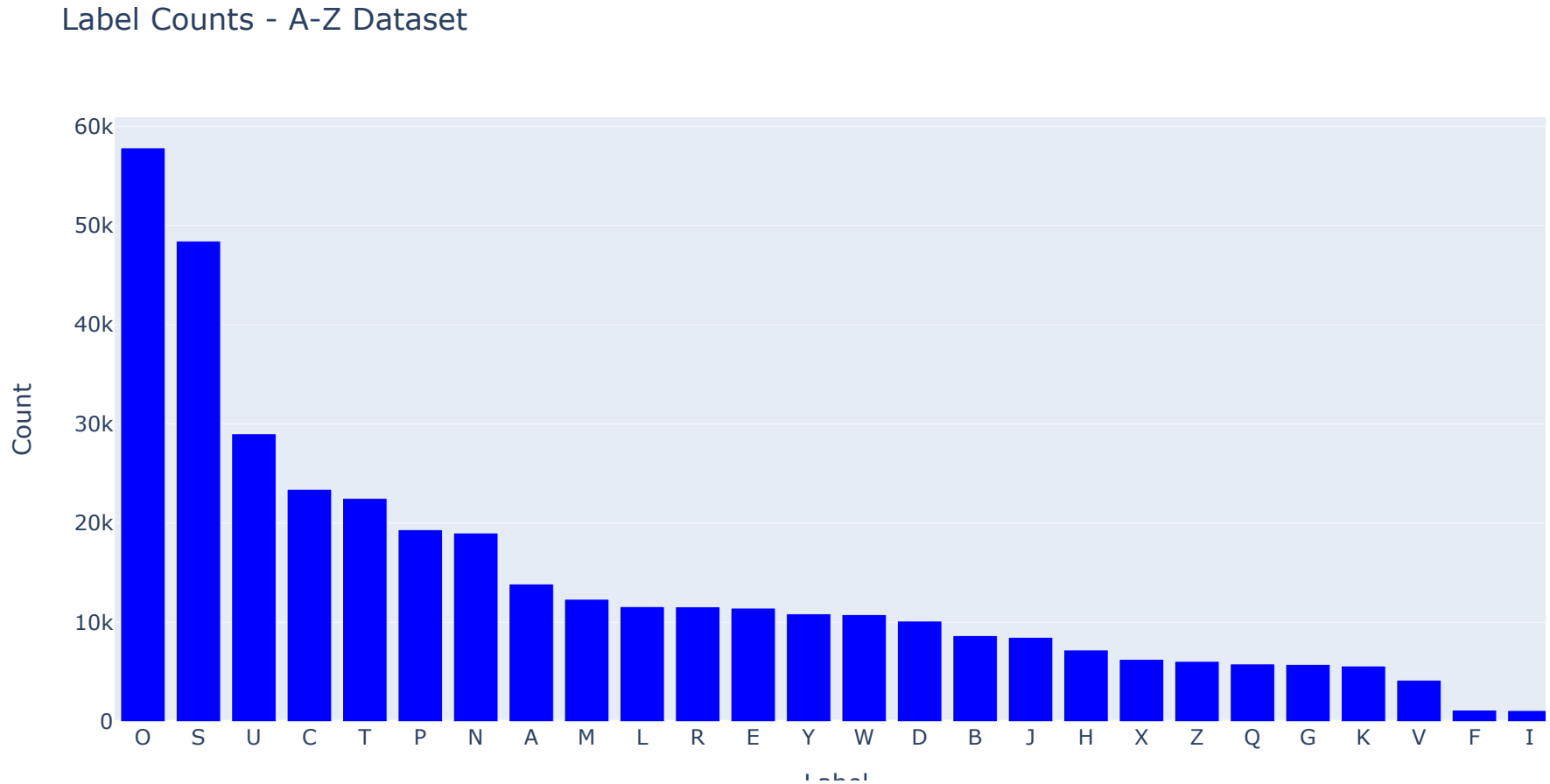
- Based on the label column counts, the dataset is unbalanced, meaning that some classes have significantly more instances than others.

In [24]:
```
# data of label cloumn counts checking
A_Z_df.label.value_counts()
```

Out[24]:
```
O    57825
S    48419
U    29008
C    23409
T    22495
P    19341
N    19010
A    13870
M    12336
L    11586
R    11566
E    11440
Y    10859
W    10784
D    10134
B     8668
J     8493
H     7218
X     6272
Z     6076
Q     5812
G     5762
K     5603
V     4182
F     1163
I     1120
Name: label, dtype: int64
```

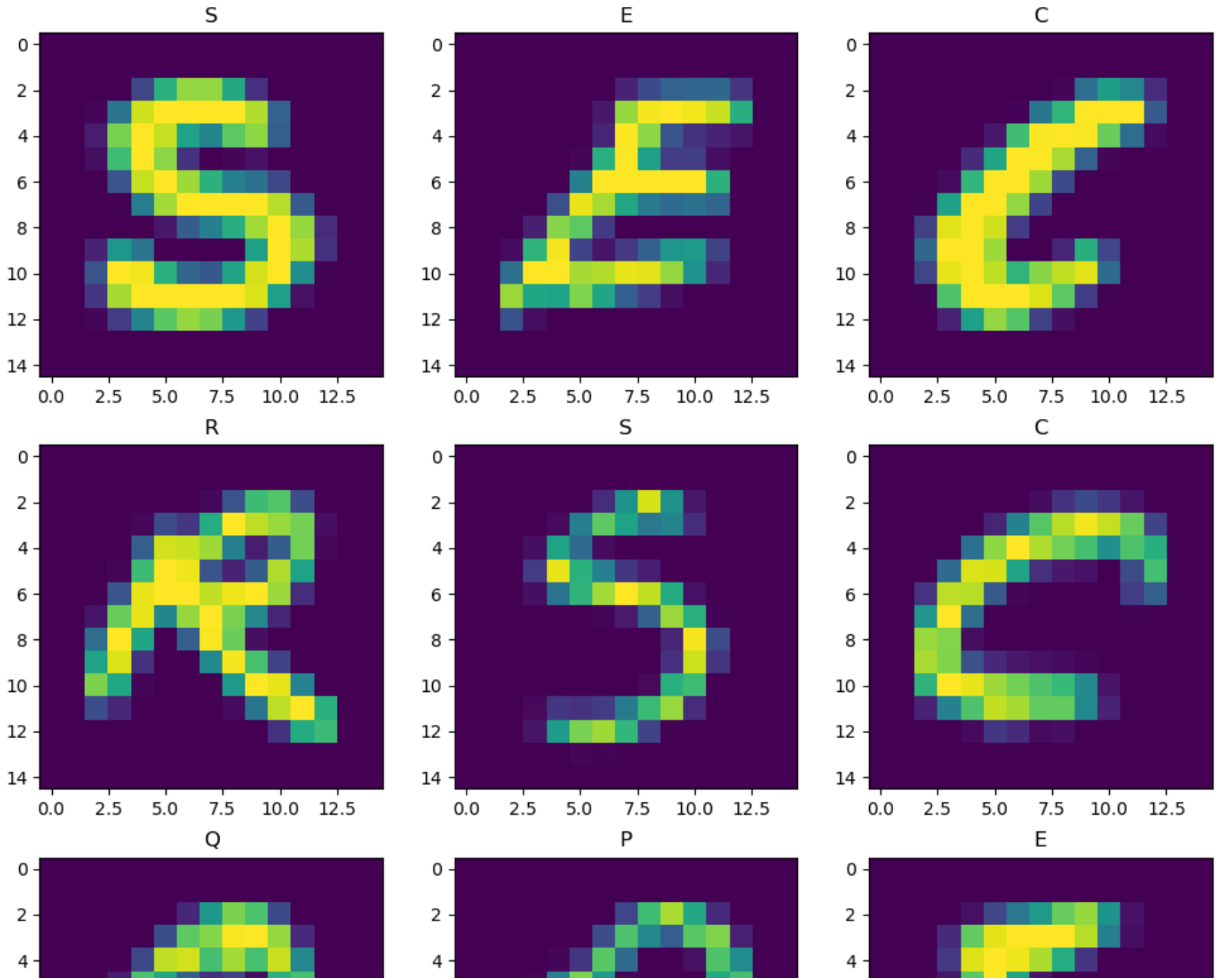
In [25]:

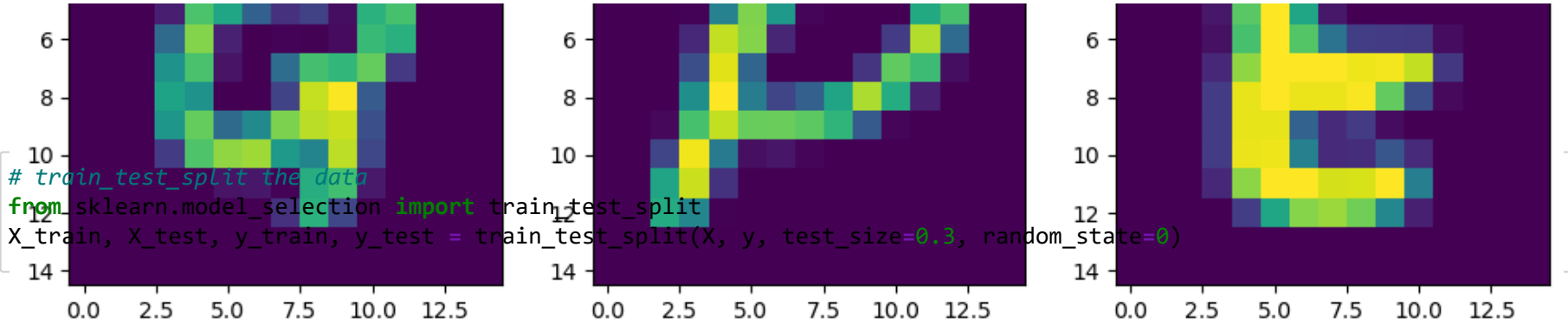
```
import plotly.graph_objects as go
import pandas as pd

# Create a DataFrame with label counts
label_counts = A_Z_df['label'].value_counts().reset_index()

# Create a bar chart
fig = go.Figure(data=go.Bar(
    x=label_counts['index'],
    y=label_counts['label'],
    marker_color='blue'
))

# Set the axis labels and title
fig.update_layout(
    xaxis=dict(title='Label'),
    yaxis=dict(title='Count'),
    title='Label Counts - A-Z Dataset'
)

# Show the chart
fig.show()
```



The countplot reveals that the label "O" has the highest count in the dataset, with approximately 57,825 occurrences. Following "O", the label "S" has the second highest count, with around 48,419 occurrences. These counts indicate the frequency of each label within the dataset

In [26]:
```
# create a x,y values
X = A_Z_df.drop('label', axis=1)
y = A_Z_df['label']
# To show images randomply after resized images
plt.figure(figsize=(12, 12))
idx = np.random.randint(0, len(X), 9)

for i in range(len(idx)):
    plt.subplot(3, 3, i+1)
    plt.title(y.values[idx[i]])
    img_grid = np.reshape(X.values[idx[i]], (15,15))
    plt.imshow(img_grid)
```


S
E
C
R
S
C
Q
P
E



In [27]:
```
# train_test_split the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Machine Learning Algorithms Overview

Logistic Regression:

- Definition: Logistic Regression is a binary classification algorithm used to predict the probability of a binary outcome based on input features.
- Formula:Logistic Function (Sigmoid): σ(z) = 1 / (1 + e^(-z)).
- Hypothesis Function: h(x) = σ(β0 + β1x1 + β2x2 + ... + βn*xn).
- Loss Function: J(β) = (-1/m) * Σ[y*log(h(x)) + (1-y)*log(1-h(x))].
- Update Rule (Gradient Descent): βj := βj - α * (∂J(β)/∂βj).

Decision Tree:

- Definition: Decision Tree is a flowchart-like structure where internal nodes represent feature tests, branches represent feature outcomes, and leaf nodes represent class labels.
- Formula: Decision trees use conditional statements to split the data based on feature values until reaching leaf nodes that contain the class labels.

Random Forest:

- Definition: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. Each tree is trained on a random subset of the data, and predictions are aggregated through voting or averaging.
- Formula: Random Forest builds a collection of decision trees and combines their predictions using voting or averaging methods to determine the final class label.

k-Nearest Neighbors (k-NN):

- Definition: k-Nearest Neighbors is a non-parametric algorithm that classifies an unknown data point based on the majority class of its k nearest neighbors in the feature space.
- Formula: The class of an unknown data point is determined by the majority class of its k nearest neighbors using distance metrics such as Euclidean distance.

Support Vector Machines (SVM):

- Definition: Support Vector Machines is a binary classification algorithm that finds an optimal hyperplane in a high-dimensional feature space to separate the data into different classes while maximizing the margin between classes.
- Formula: SVM finds the hyperplane defined by $w \cdot x + b = 0$ that maximizes the margin, where w is the weight vector, x is the input vector, and b is the bias term.

Naive Bayes:

- Definition: Naive Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of independence between features. It calculates the posterior probability of a class given the input features.
- Formula: $P(y|X) = (P(X|y) * P(y)) / P(X)$, where $P(y|X)$ is the posterior probability, $P(X|y)$ is the likelihood, $P(y)$ is the prior probability, and $P(X)$ is the evidence.

XGBoost:

- Definition: XGBoost (Extreme Gradient Boosting) is an ensemble learning method that uses a gradient boosting framework to create a strong predictive model by combining multiple weak models called decision trees.
- Formula: XGBoost constructs a strong predictive model by iteratively training decision trees. Each tree is trained to minimize a loss function, and subsequent trees are built to correct the mistakes of the previous trees. The final prediction is made by summing the predictions of all the trees.

AdaBoost:

- Definition: AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. It focuses on the samples that were misclassified by previous classifiers, assigning them higher weights to improve their classification in subsequent iterations.
- Formula: AdaBoost assigns initial weights to training samples and trains a weak classifier. The weights are then adjusted based on the misclassified samples, giving higher weights to misclassified samples and lower weights to correctly classified samples. Multiple weak classifiers are trained in this manner, and their predictions are combined using weighted voting to obtain the final classification.

Metrics

Accuracy Score:

- Accuracy score measures the overall correctness of the classification model. It is the ratio of the correctly classified samples to the total number of samples.
- Accuracy = (Number of Correctly Classified Samples) / (Total Number of Samples)

Precision Score:

- Precision score measures the ability of a classification model to correctly classify positive instances. It is the ratio of the correctly classified positive samples to the total number of samples predicted as positive.
- Precision = (Number of True Positives) / (Number of True Positives + Number of False Positives)

Recall Score:

- Recall score (also known as sensitivity or true positive rate) measures the ability of a classification model to identify all positive instances. It is the ratio of the correctly classified positive samples to the total number of actual positive samples.
- Recall = (Number of True Positives) / (Number of True Positives + Number of False Negatives)

F1 Score:

- F1 score is the harmonic mean of precision and recall scores. It provides a single value that combines both precision and recall, making it useful for evaluating the overall performance of a classification model.
- F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Classification Report:

- A classification report is a summary of various evaluation metrics for each class in a classification problem. It includes precision, recall, F1 score, and support (the number of occurrences of each class in the true labels).

SPRINT 2:BUILD A MODEL

Training and Evaluating Multiple Classifiers:

In [28]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, precision_score, f1_score, recall_score
import warnings
warnings.filterwarnings("ignore")
import xgboost as xgb

Algorithm = []
Accuracy = []
Precision = []
F1_Score = []
Recall = []
y_predct=[]
X_train, X_test, y_train, y_test

def train_classifiers(classifier):
    # Create the specified classifier
    if classifier == 'Logistic Regression':
        model = LogisticRegression()
        Algorithm.append("Logistic Regression")
    elif classifier == 'Decision Tree':
        model = DecisionTreeClassifier()
        Algorithm.append("Decision Tree")
    elif classifier == 'Random Forest':
        model = RandomForestClassifier()
        Algorithm.append("Random Forest")
    elif classifier == 'KNN':
        model = KNeighborsClassifier()
        Algorithm.append("KNeighborsClassifier")
    elif classifier == 'SVM':
        model = SVC()
        Algorithm.append("Support Vector Machines Classifier")
    elif classifier == 'Naive Bayes':
        model = GaussianNB()
        Algorithm.append("Naive Bayes Classifier")
    elif classifier == 'AdaBoost':
```

```
        model = AdaBoostClassifier()
        Algorithm.append("AdaBoost Classifier")
    else:
        raise ValueError("Invalid classifier specified.")

    # Train the classifier
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    report = classification_report(y_test, y_pred)
    confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
    y_predct.append(y_pred)
    Accuracy.append(accuracy)
    Precision.append(precision)
    F1_Score.append(f1)
    Recall.append(recall)

    print(f'{model}:')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'F1 Score: {f1:.4f}')
    print(f'Recall: {recall:.4f}')
    print('Classification Report:\n', report)

    return confusion_matrix.style.background_gradient(cmap="tab10")
```

Visualizing Predicted and True Label Counts:

In [61]:

```
import plotly.graph_objects as go
import pandas as pd

def create_grouped_bar_chart(df):
    # Calculate the count of predicted labels and true labels
    predicted_counts = df["Predicted Label"].value_counts()
    true_counts = df["True Label"].value_counts()

    # Get the unique labels
    labels = df["Predicted Label"].unique()
    labels.sort()  # Sort the labels in a consistent order

    # Create a bar chart
    fig = go.Figure()

    # Add the predicted counts bar
    fig.add_trace(go.Bar(
        x=labels,
        y=predicted_counts.loc[labels],  # Use loc to ensure consistent label order
        name='Predicted',
        marker_color='blue'
    ))

    # Add the true counts bar
    fig.add_trace(go.Bar(
        x=labels,
        y=true_counts.loc[labels],  # Use loc to ensure consistent label order
        name='True',
        marker_color='green'
    ))

    # Set the axis labels and title
    fig.update_layout(
        xaxis=dict(title='Label'),
        yaxis=dict(title='Count'),
        title='Count of Predicted and True Labels',
        barmode='group'
    )

    # Show the chart
    fig.show()
```

LogisticRegression:

In [30]:
```
train_classifiers('Logistic Regression')
```

LogisticRegression():
Accuracy: 0.8738
Precision: 0.8727
F1 Score: 0.8730
Recall: 0.8738
Classification Report:

	precision	recall	f1-score	support
A	0.84	0.84	0.84	4204
B	0.86	0.79	0.82	2656
C	0.90	0.89	0.90	7048
D	0.82	0.76	0.79	3005
E	0.81	0.79	0.80	3402
F	0.91	0.80	0.85	351
G	0.82	0.75	0.78	1737
H	0.75	0.73	0.74	2172
I	0.87	0.79	0.83	350
J	0.77	0.73	0.75	2538
K	0.77	0.75	0.76	1650
L	0.93	0.94	0.93	3459
M	0.87	0.88	0.88	3694
N	0.81	0.81	0.81	5760
O	0.92	0.96	0.94	17569
P	0.90	0.92	0.91	5744
Q	0.84	0.76	0.80	1794
R	0.80	0.80	0.80	3466
S	0.93	0.93	0.93	14381
T	0.92	0.94	0.93	6671
U	0.87	0.89	0.88	8616
V	0.88	0.90	0.89	1230
W	0.82	0.79	0.80	3227
X	0.84	0.83	0.83	1894
Y	0.84	0.82	0.83	3275
Z	0.87	0.83	0.85	1843
accuracy			0.87	111736
macro avg	0.85	0.83	0.84	111736
weighted avg	0.87	0.87	0.87	111736

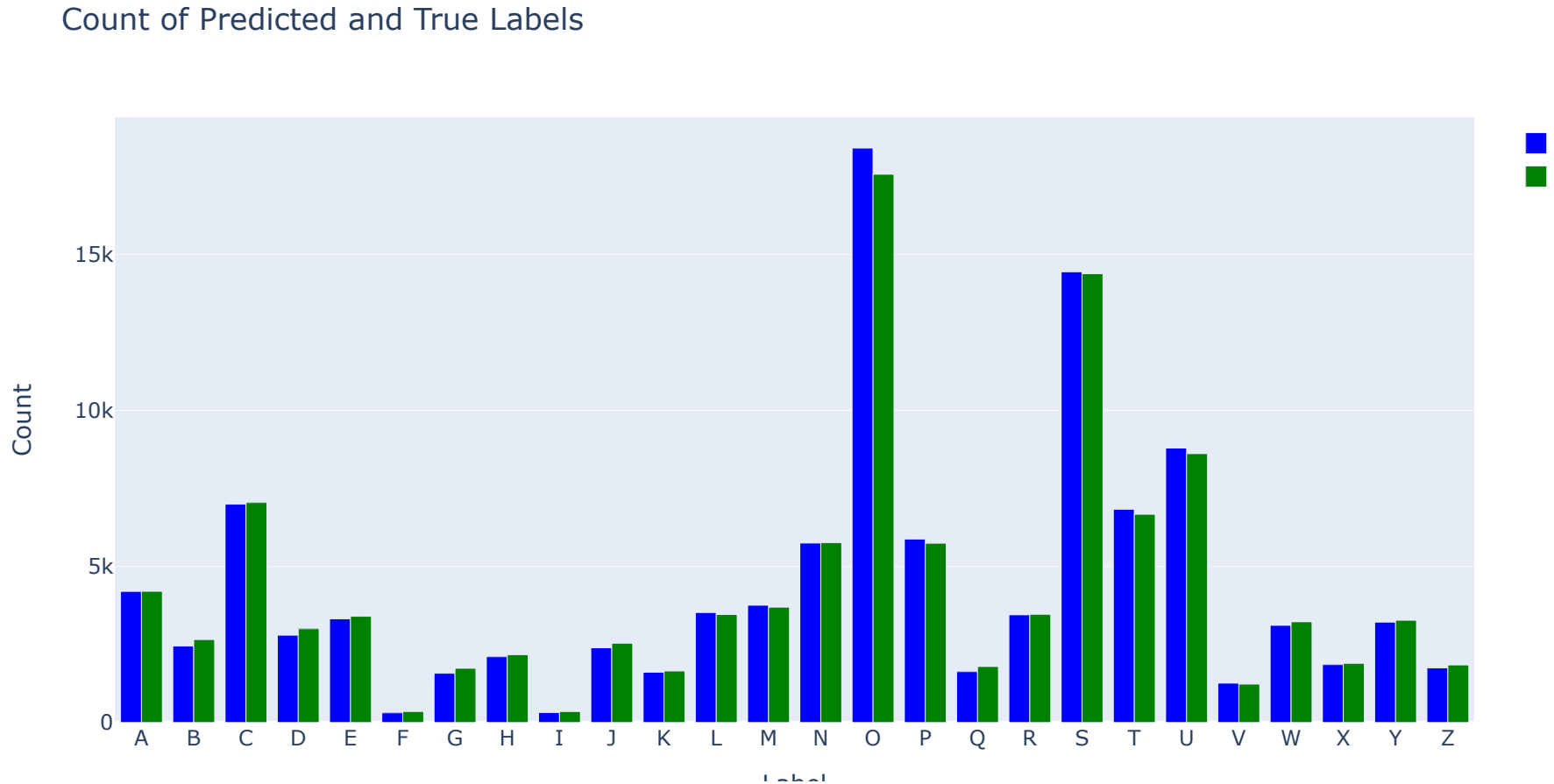
Out[30]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	3532	12	1	8	28	1	8	96	0	3	25	3	83	86	27	57	22	39	30	1	41	
B	20	2096	15	33	71	0	4	11	0	12	2	0	20	3	91	6	2	40	145	1	32	
C	2	27	6289	1	87	1	46	4	1	4	18	54	1	30	235	48	5	40	49	13	70	
D	9	24	0	2297	4	0	1	0	0	21	1	4	13	4	453	25	12	1	71	0	32	
E	15	46	194	7	2675	6	36	5	0	12	59	20	0	21	20	16	2	96	85	1	25	
F	4	0	0	0	12	281	1	0	0	0	0	0	0	0	0	18	0	4	0	30	0	
G	15	19	50	5	37	0	1298	2	0	3	0	11	3	4	37	1	64	6	125	3	38	
H	101	7	1	4	19	0	4	1588	0	5	2	3	71	172	3	13	3	15	7	2	90	
I	0	0	1	3	2	0	0	0	278	14	0	0	0	0	0	3	2	0	25	3	0	
J	0	2	34	45	17	0	7	3	23	1852	1	9	5	4	33	13	8	2	252	63	116	
K	8	3	59	0	35	1	0	15	0	0	1232	28	9	38	3	0	0	130	7	8	9	
L	1	2	42	4	7	0	13	0	2	6	15	3263	0	2	0	0	0	5	9	1	22	
M	55	10	0	5	1	2	8	79	0	0	2	0	3267	131	25	19	2	16	2	23	17	
N	155	8	5	20	24	0	1	151	0	2	23	23	132	4638	23	22	8	39	7	8	173	2
O	17	39	102	133	16	0	5	3	0	9	2	2	38	66	16919	21	5	11	74	1	85	
P	25	1	8	50	16	1	0	1	0	1	2	0	3	11	32	5269	8	27	6	190	4	
Q	19	0	11	9	6	0	41	0	0	2	0	0	1	4	140	35	1363	49	27	4	76	
R	103	43	56	7	111	1	2	8	0	2	92	4	23	48	8	59	34	2757	4	9	17	
S	14	65	84	70	35	2	68	1	7	262	33	12	7	15	117	16	36	37	13385	13	37	
T	18	1	11	3	12	11	1	31	0	15	9	0	21	1	1	139	12	1	14	6269	0	
U	22	4	18	57	31	0	22	46	0	13	12	36	22	127	211	1	20	61	25	2	7644	2
V	0	0	0	0	12	3	0	6	0	0	13	0	3	31	0	1	0	1	1	1	29	110
W	9	2	5	9	13	0	7	29	0	5	7	25	17	282	10	1	4	6	3	3	217	1
X	36	9	0	2	8	0	1	1	0	2	41	9	4	4	1	3	2	40	35	2	7	1

Predicted	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
Actual																						
Y	10	5	1	4	3	0	0	22	2	83	17	2	9	13	16	86	5	2	32	180	10	5
Z	10	23	8	21	39	0	2	6	6	65	2	15	6	16	3	5	11	25	22	2	2	
All	4200	2448	6995	2797	3321	310	1576	2108	319	2393	1610	3523	3758	5751	18408	5877	1630	3450	14442	6833	8793	126

In [62]:
```
df1 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[0]})
```

In [63]:
```
create_grouped_bar_chart(df1)
```



DecisionTreeClassifier:

In [33]:
```
train_classifiers('Decision Tree')
```

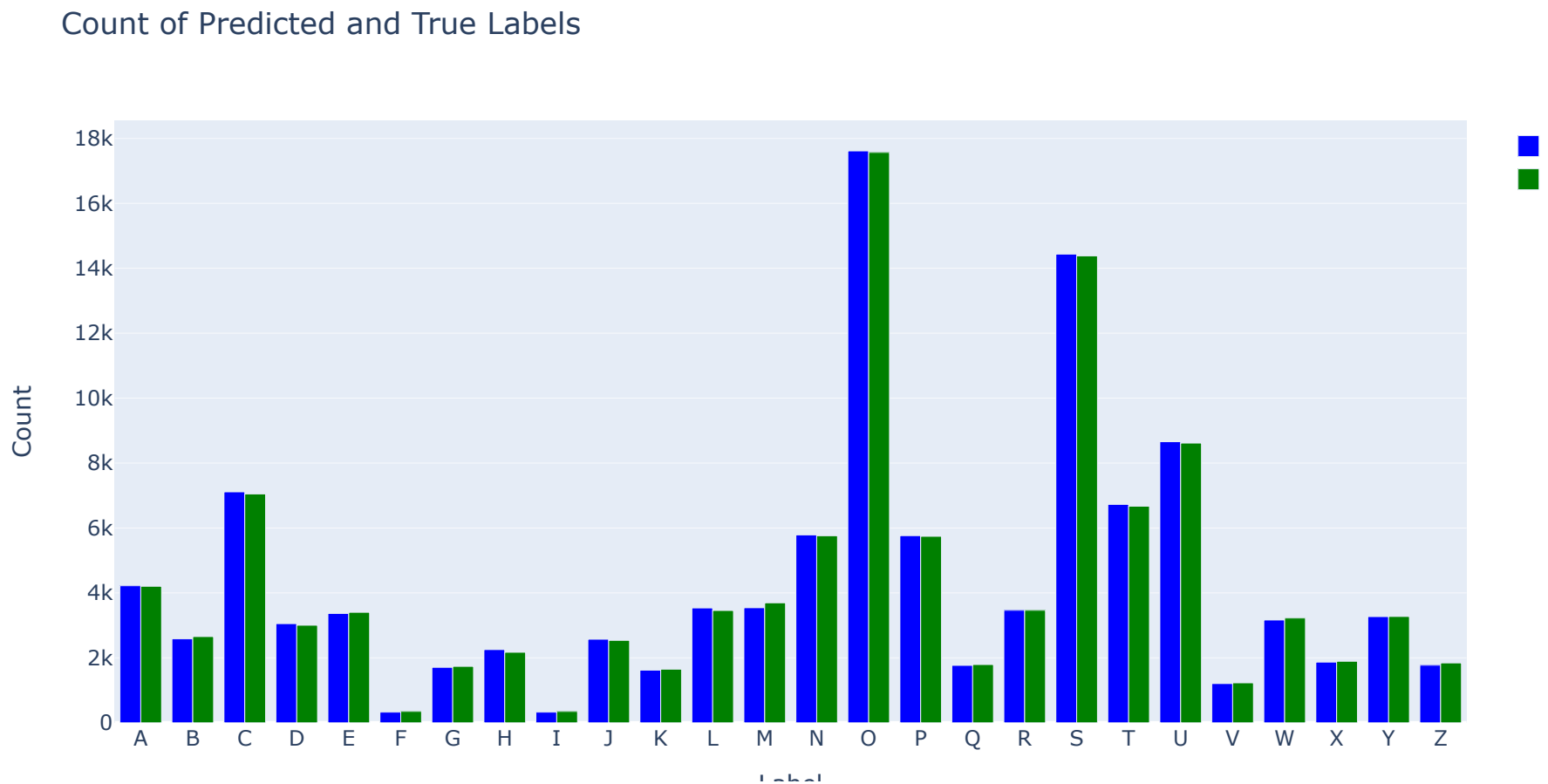
```
DecisionTreeClassifier():
Accuracy: 0.9478
Precision: 0.9477
F1 Score: 0.9477
Recall: 0.9478
Classification Report:
              precision    recall  f1-score   support

           A       0.93      0.93      0.93      4204
           B       0.91      0.89      0.90      2656
           C       0.96      0.97      0.96      7048
           D       0.89      0.91      0.90      3005
           E       0.93      0.92      0.92      3402
           F       0.91      0.84      0.87       351
           G       0.89      0.88      0.89      1737
           H       0.86      0.89      0.87      2172
           I       0.90      0.84      0.87       350
           J       0.92      0.94      0.93      2538
           K       0.91      0.89      0.90      1650
           L       0.95      0.97      0.96      3459
           M       0.94      0.90      0.92      3694
           N       0.93      0.94      0.94      5760
           O       0.98      0.98      0.98     17569
           P       0.96      0.97      0.97      5744
           Q       0.89      0.88      0.88      1794
           R       0.92      0.92      0.92      3466
           S       0.97      0.98      0.98     14381
           T       0.97      0.98      0.98      6671
           U       0.96      0.96      0.96      8616
           V       0.94      0.92      0.93      1230
           W       0.94      0.92      0.93      3227
           X       0.93      0.92      0.92      1894
           Y       0.94      0.94      0.94      3275
           Z       0.95      0.91      0.93      1843

    accuracy                           0.95    111736
   macro avg       0.93      0.92      0.93    111736
weighted avg       0.95      0.95      0.95    111736
```

Out[33]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	3907	19	8	7	12	1	10	33	0	4	9	2	35	20	7	18	18	50	12	3	10	
B	13	2354	9	24	20	0	19	10	0	6	6	0	17	4	42	8	23	17	48	0	13	
C	4	8	6820	2	27	2	18	4	0	2	6	43	0	4	29	5	8	17	15	5	10	
D	6	24	3	2720	1	0	2	6	0	13	2	2	4	2	142	19	12	2	15	4	16	
E	11	19	46	4	3126	9	26	3	6	11	16	21	0	4	11	13	0	27	29	2	6	
F	0	0	1	0	13	296	1	0	0	6	2	0	0	0	4	19	0	1	0	8	0	
G	19	21	26	1	20	0	1524	5	0	3	4	3	6	12	14	1	16	4	42	4	9	
H	30	7	4	3	6	0	8	1934	0	0	3	0	45	60	5	7	1	0	1	2	29	
I	0	3	0	0	4	1	0	0	293	15	0	2	0	1	1	0	0	1	8	4	0	
J	4	3	6	6	3	0	2	2	1	2374	0	0	2	8	5	3	4	0	56	24	6	
K	10	1	13	1	8	1	3	11	2	0	1473	22	7	9	0	2	0	22	4	3	19	
L	0	0	36	2	2	0	2	0	0	6	7	3358	0	0	2	1	0	12	2	3	6	
M	62	7	3	11	4	0	11	62	0	6	6	0	3318	69	17	12	5	15	8	13	32	
N	28	10	7	12	0	0	4	87	0	8	4	8	43	5399	8	7	5	12	8	1	37	
O	14	15	35	153	12	0	10	4	0	6	2	0	8	5	17183	12	40	4	20	0	36	
P	3	9	9	31	14	8	0	6	2	0	0	0	4	9	6	5561	4	10	8	25	3	
Q	15	7	11	15	8	0	11	2	2	4	0	2	5	4	58	7	1576	17	23	3	11	
R	50	11	24	3	14	1	4	15	3	4	30	17	6	7	7	8	14	3186	5	2	18	
S	4	35	10	16	29	4	25	4	6	52	8	10	0	4	32	0	10	12	14065	14	8	
T	0	2	3	3	2	2	0	0	1	14	0	1	1	3	1	22	0	4	13	6555	0	
U	11	0	17	20	4	0	14	19	0	19	11	2	15	62	31	0	13	14	4	2	8303	1
V	0	0	0	2	3	2	0	1	0	1	3	0	2	7	3	2	0	1	0	0	31	113
W	5	8	7	5	8	0	10	26	0	0	6	4	19	79	2	4	14	7	6	1	36	
X	16	2	3	2	10	0	1	5	2	0	17	12	2	8	1	2	1	12	15	2	13	

Predicted	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
Actual																						
Y	5	3	4	7	3	0	2	9	0	12	2	7	5	4	4	23	0	1	25	35	1	2
Z	4	17	8	3	14	0	0	7	9	8	4	21	0	2	0	7	4	18	4	11	5	
All	4221	2585	7113	3053	3367	327	1707	2255	327	2574	1621	3537	3544	5786	17615	5763	1768	3466	14436	6726	8658	120

In [64]:
```
df2 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[1]})
create_grouped_bar_chart(df2)
```



RandomForestClassifier:

In [35]:
```
train_classifiers('Random Forest')
```

RandomForestClassifier():
Accuracy: 0.9850
Precision: 0.9850
F1 Score: 0.9850
Recall: 0.9850
Classification Report:

	precision	recall	f1-score	support
A	0.97	0.99	0.98	4204
B	0.98	0.97	0.98	2656
C	0.99	0.99	0.99	7048
D	0.97	0.96	0.97	3005
E	0.99	0.97	0.98	3402
F	0.99	0.92	0.95	351
G	0.98	0.95	0.97	1737
H	0.98	0.95	0.96	2172
I	1.00	0.91	0.96	350
J	0.98	0.97	0.98	2538
K	0.98	0.97	0.97	1650
L	0.99	0.99	0.99	3459
M	0.98	0.97	0.97	3694
N	0.97	0.99	0.98	5760
O	0.99	1.00	0.99	17569
P	0.99	0.99	0.99	5744
Q	0.99	0.94	0.97	1794
R	0.98	0.97	0.98	3466
S	0.99	1.00	0.99	14381
T	0.99	1.00	0.99	6671
U	0.99	0.99	0.99	8616
V	0.99	0.98	0.99	1230
W	0.99	0.97	0.98	3227
X	0.99	0.97	0.98	1894
Y	0.98	0.98	0.98	3275
Z	0.99	0.97	0.98	1843
accuracy			0.99	111736
macro avg	0.98	0.97	0.98	111736
weighted avg	0.99	0.99	0.98	111736

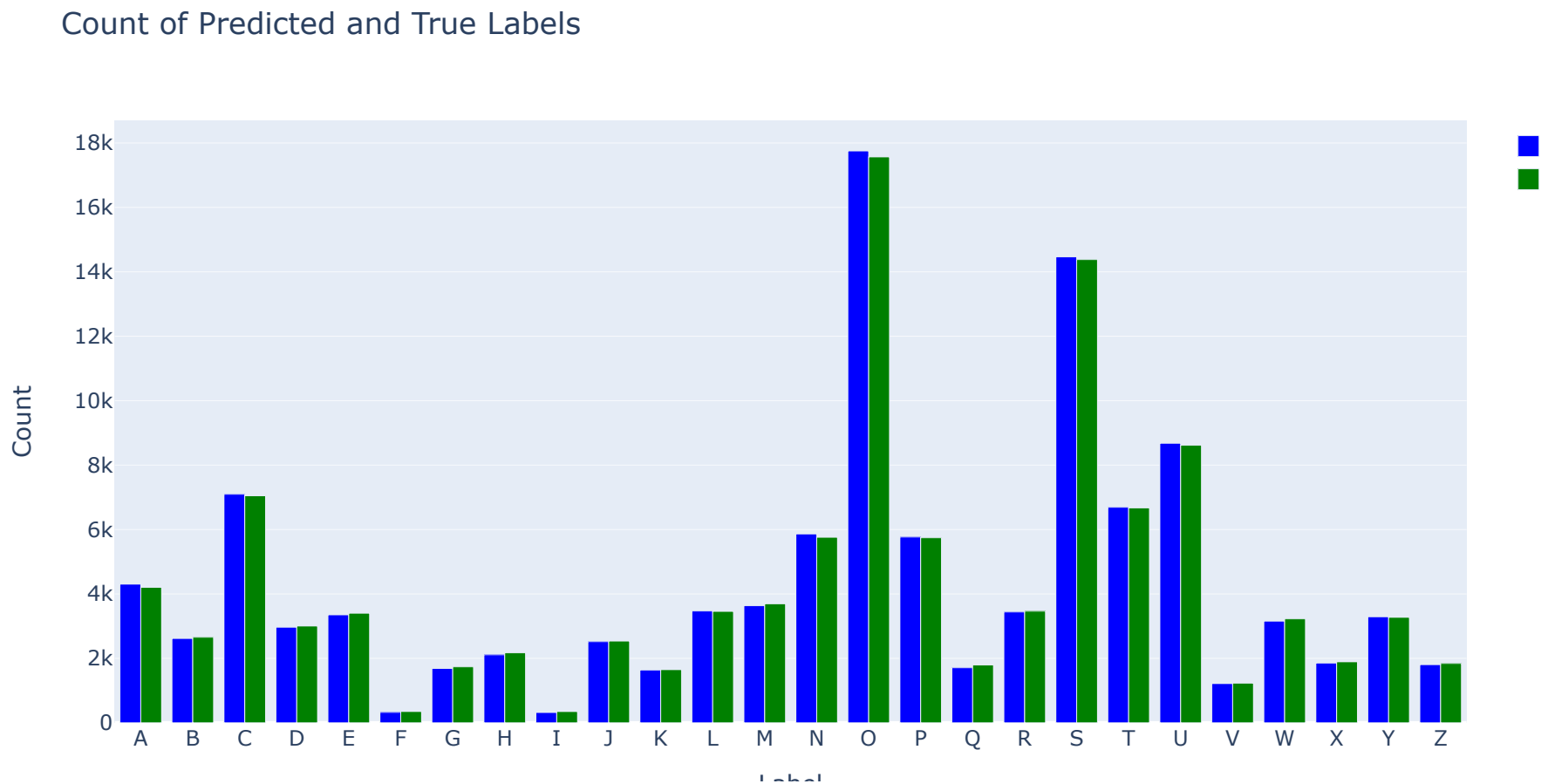
Out[35]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	4162	0	0	2	0	0	1	2	0	0	0	3	11	5	5	2	0	4	2	0	1	
B	14	2572	1	8	1	0	3	0	0	0	0	0	2	1	24	2	0	7	17	0	1	
C	2	2	6996	0	0	0	3	0	0	0	0	15	0	0	19	1	0	0	2	0	6	
D	0	5	0	2892	0	0	0	0	0	7	0	0	0	0	93	5	1	0	2	0	0	
E	0	2	43	0	3306	2	10	0	0	0	3	3	0	2	3	5	0	6	11	0	2	
F	1	0	0	0	4	322	0	0	0	0	0	0	0	0	0	17	0	0	1	6	0	
G	5	11	15	2	3	0	1656	0	0	2	0	0	0	0	16	0	7	0	13	2	5	
H	29	3	0	0	0	0	2	2061	0	0	0	0	15	33	0	6	0	2	2	1	10	
I	0	0	1	0	0	0	0	0	320	8	0	0	0	0	0	1	0	0	10	3	0	
J	0	3	0	1	4	0	0	0	0	2467	0	0	2	0	1	2	0	0	29	14	8	
K	0	0	6	0	3	0	0	3	0	0	1598	7	4	2	0	0	0	9	1	0	7	
L	0	0	12	0	2	0	0	0	0	2	0	3431	0	0	0	0	0	2	0	0	2	
M	33	3	0	2	0	0	0	29	0	0	0	0	3567	41	2	0	0	2	0	0	8	
N	11	0	0	0	0	0	0	14	0	2	0	0	8	5704	0	2	0	3	0	2	10	
O	0	2	2	33	0	0	0	0	0	2	0	0	2	0	17513	6	0	0	4	0	5	
P	2	0	0	16	0	0	0	0	0	2	0	0	0	0	5	5697	2	4	2	8	0	
Q	0	1	4	1	0	0	7	0	0	0	0	0	1	4	53	4	1692	12	4	0	11	
R	35	2	9	2	4	0	1	0	0	0	9	4	2	2	0	7	7	3376	1	0	2	
S	0	1	2	0	4	0	3	0	0	7	2	0	4	0	6	2	0	0	14341	2	2	
T	2	0	1	1	0	0	0	0	0	2	1	0	1	0	0	2	0	0	0	6644	0	
U	2	0	0	2	2	0	0	0	0	0	0	2	6	4	14	0	0	9	2	0	8567	
V	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	9	121
W	2	2	0	4	0	0	0	2	0	0	0	0	3	64	0	0	0	0	2	0	20	
X	2	0	0	0	0	0	0	0	0	1	25	1	3	0	0	0	0	0	2	0	0	

Predicted	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
Actual																						
Y	1	0	2	0	0	0	0	1	0	16	0	0	3	0	0	7	0	0	14	6	2	
Z	2	9	2	1	19	0	0	0	0	1	0	4	1	0	0	0	0	5	3	3	0	
All	4305	2618	7096	2967	3352	325	1686	2112	320	2520	1638	3470	3635	5862	17754	5768	1709	3441	14465	6691	8678	121

In [ ]:

In [65]:
```
df3 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[2]})
create_grouped_bar_chart(df3)
```



KNeighborsClassifier:

In [37]:
```
train_classifiers('KNN')
```

KNeighborsClassifier():
Accuracy: 0.9664
Precision: 0.9666
F1 Score: 0.9662
Recall: 0.9664
Classification Report:

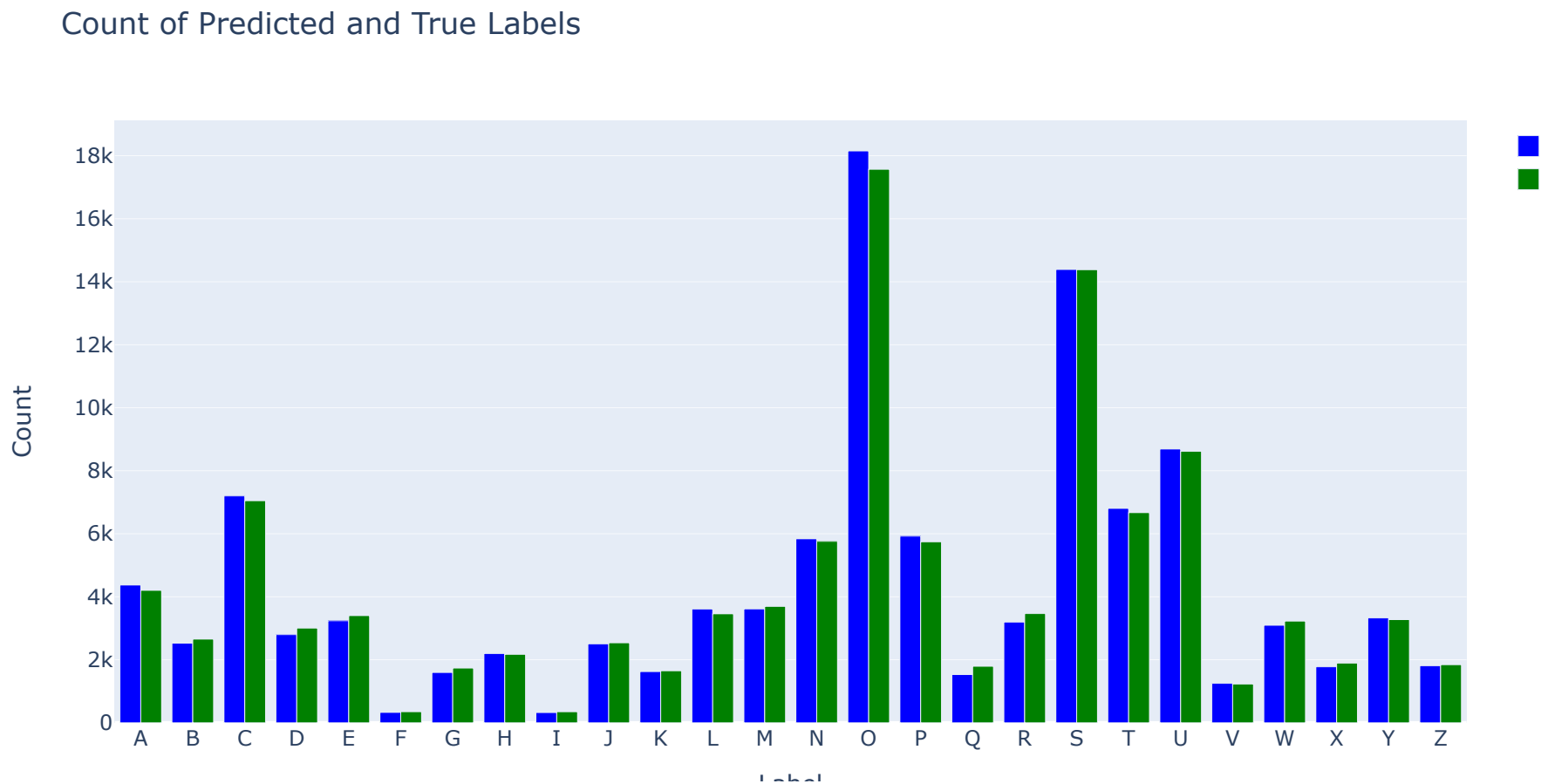
	precision	recall	f1-score	support
A	0.94	0.98	0.96	4204
B	0.97	0.93	0.95	2656
C	0.96	0.98	0.97	7048
D	0.94	0.87	0.90	3005
E	0.98	0.93	0.96	3402
F	0.96	0.91	0.93	351
G	0.97	0.89	0.93	1737
H	0.92	0.93	0.92	2172
I	0.98	0.91	0.94	350
J	0.95	0.93	0.94	2538
K	0.95	0.93	0.94	1650
L	0.94	0.98	0.96	3459
M	0.99	0.97	0.98	3694
N	0.96	0.97	0.97	5760
O	0.96	0.99	0.98	17569
P	0.95	0.98	0.96	5744
Q	0.97	0.83	0.89	1794
R	0.98	0.90	0.94	3466
S	0.99	0.99	0.99	14381
T	0.97	0.99	0.98	6671
U	0.98	0.99	0.98	8616
V	0.97	0.99	0.98	1230
W	0.99	0.95	0.96	3227
X	0.99	0.93	0.96	1894
Y	0.95	0.96	0.95	3275
Z	0.98	0.96	0.97	1843
accuracy			0.97	111736
macro avg	0.96	0.95	0.95	111736
weighted avg	0.97	0.97	0.97	111736

Out[37]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	4130	1	0	4	1	0	1	27	0	0	1	4	0	5	4	20	0	3	0	2	1	
B	19	2457	10	25	12	0	2	1	1	1	0	1	1	1	38	35	0	22	26	0	2	
C	2	3	6906	0	3	0	7	0	0	3	0	51	0	0	59	2	0	0	1	2	5	
D	8	10	1	2623	0	0	0	0	0	8	0	4	0	3	313	15	0	0	1	2	9	
E	2	6	103	4	3179	6	7	0	0	0	5	45	0	0	4	11	2	7	13	2	2	
F	0	0	0	0	3	318	0	0	0	1	1	0	0	0	0	13	0	0	0	15	0	
G	8	13	59	1	10	0	1551	0	0	2	0	2	2	0	30	3	28	0	19	2	6	
H	47	0	0	0	0	1	0	2021	0	1	1	1	13	49	0	3	0	0	0	5	9	
I	0	0	1	0	0	0	0	0	317	11	0	1	0	0	0	1	0	0	3	16	0	
J	1	0	1	6	2	0	0	0	3	2370	0	2	2	1	7	0	0	0	48	67	14	
K	5	0	12	0	4	2	0	17	0	0	1536	24	3	2	0	1	0	15	0	4	5	
L	0	0	26	0	0	0	0	1	3	2	6	3405	0	0	0	1	0	0	2	0	0	
M	15	0	0	0	0	1	2	45	0	0	0	0	3571	38	1	1	0	2	0	4	7	
N	30	0	0	2	0	0	0	53	0	4	3	2	11	5611	0	2	0	3	2	3	10	
O	2	2	13	65	3	0	4	0	0	4	0	1	0	0	17432	5	2	0	2	2	31	
P	9	0	1	37	1	1	0	0	0	6	0	2	0	1	6	5630	3	4	1	30	0	
Q	6	0	12	5	0	0	14	0	1	0	0	0	0	4	210	11	1486	14	3	0	23	
R	60	23	15	4	5	1	2	3	0	0	46	7	5	5	1	150	4	3120	0	0	6	
S	4	6	16	5	5	0	4	0	0	37	0	6	1	4	16	2	0	1	14263	2	6	
T	5	0	0	1	0	0	0	0	0	3	1	3	1	0	1	6	1	0	0	6625	0	
U	5	1	14	10	0	0	1	8	0	16	0	12	1	13	29	0	2	2	0	0	8491	
V	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0	0	3	121
W	4	0	2	4	0	0	1	8	0	0	2	2	3	97	0	2	0	0	0	0	52	
X	4	0	1	0	0	0	0	1	0	1	21	9	0	2	0	1	0	0	0	2	0	

Predicted	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
Actual																						
Y	7	2	2	0	1	0	0	13	0	30	1	2	0	4	1	7	1	0	4	15	9	2
Z	0	3	6	3	10	0	0	0	0	2	1	25	0	0	0	2	1	3	2	6	0	
All	4373	2527	7201	2799	3239	330	1596	2198	325	2505	1625	3611	3614	5840	18152	5925	1530	3196	14390	6806	8691	125

In [66]:
```
df4 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[3]})
create_grouped_bar_chart(df4)
```



Support Vector Machines:

In [39]:
```
train_classifiers('SVM')
```

SVC():
Accuracy: 0.9785
Precision: 0.9785
F1 Score: 0.9784
Recall: 0.9785
Classification Report:

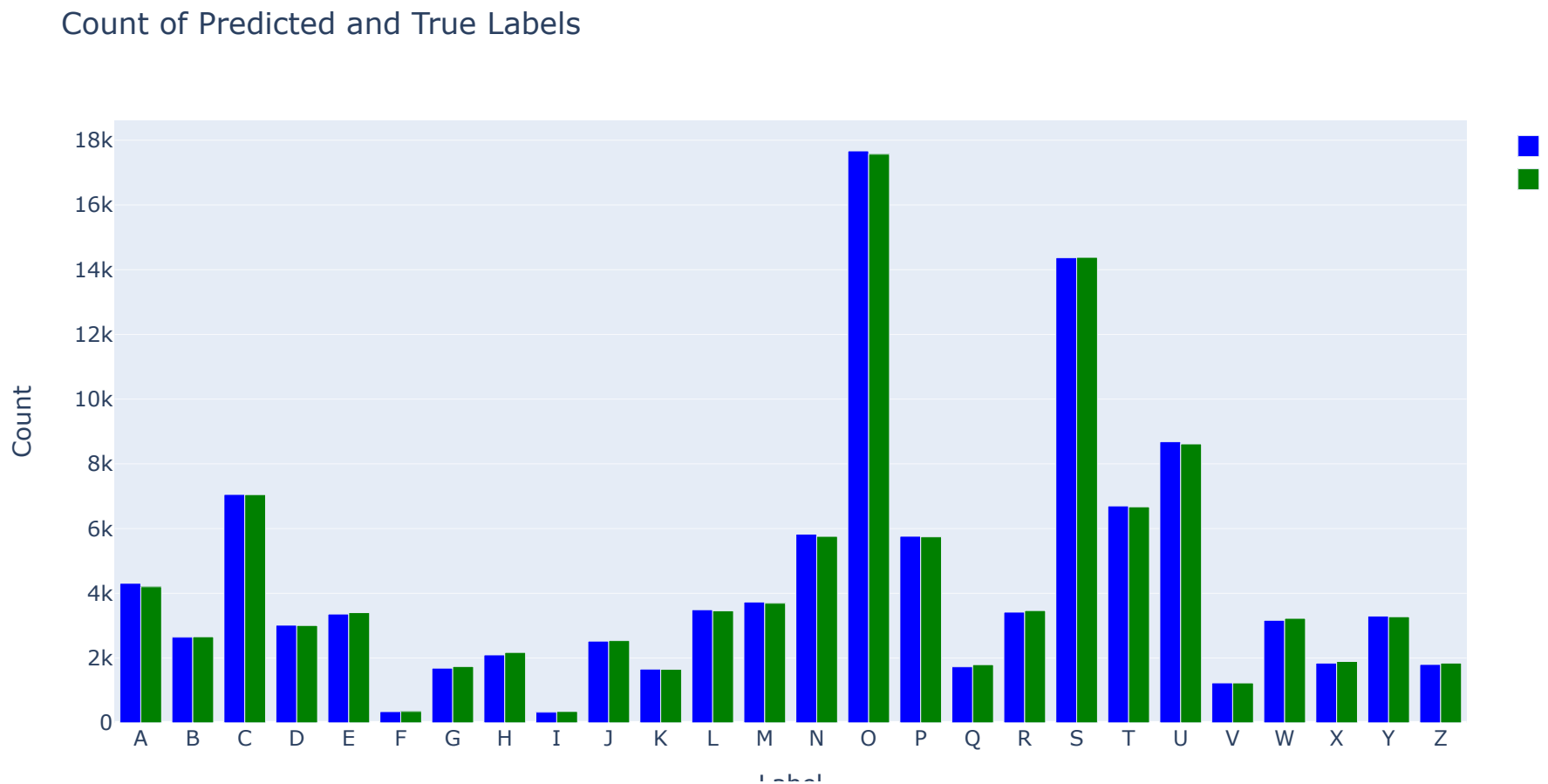
	precision	recall	f1-score	support
A	0.96	0.99	0.97	4204
B	0.97	0.97	0.97	2656
C	0.98	0.98	0.98	7048
D	0.94	0.95	0.95	3005
E	0.98	0.97	0.97	3402
F	0.98	0.97	0.97	351
G	0.97	0.94	0.95	1737
H	0.96	0.93	0.95	2172
I	0.99	0.92	0.96	350
J	0.96	0.96	0.96	2538
K	0.94	0.94	0.94	1650
L	0.98	0.98	0.98	3459
M	0.97	0.98	0.97	3694
N	0.97	0.98	0.97	5760
O	0.99	0.99	0.99	17569
P	0.98	0.98	0.98	5744
Q	0.98	0.94	0.96	1794
R	0.98	0.96	0.97	3466
S	0.99	0.99	0.99	14381
T	0.99	0.99	0.99	6671
U	0.98	0.99	0.99	8616
V	0.99	0.99	0.99	1230
W	0.98	0.96	0.97	3227
X	0.98	0.95	0.96	1894
Y	0.96	0.97	0.96	3275
Z	0.99	0.97	0.98	1843
accuracy			0.98	111736
macro avg	0.97	0.97	0.97	111736
weighted avg	0.98	0.98	0.98	111736

Out[39]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
A	4144	1	0	2	0	0	1	16	0	0	2	3	5	8	1	5	0	7	0	1	3	
B	5	2570	0	10	6	0	4	3	0	0	1	0	5	1	15	5	0	19	5	0	3	
C	3	4	6928	0	11	0	2	0	0	1	1	32	0	2	37	2	0	1	2	1	14	
D	4	10	0	2848	1	0	1	0	0	5	0	0	2	3	113	9	1	0	1	0	3	
E	1	5	42	1	3291	4	13	0	0	2	8	3	0	0	0	6	1	10	6	1	2	
F	0	0	0	0	2	339	0	0	0	0	0	0	0	1	0	5	0	0	0	3	0	
G	2	19	16	4	7	0	1631	1	0	4	0	0	1	2	12	1	15	1	13	2	3	
H	22	0	0	1	2	0	0	2018	0	1	2	2	27	54	0	3	0	2	3	2	9	
I	0	0	1	0	0	0	0	0	323	8	0	0	0	0	0	2	0	0	6	3	0	
J	0	1	0	8	3	0	1	0	3	2428	0	0	2	1	3	2	0	0	29	30	16	
K	5	0	12	0	5	0	0	6	0	0	1558	9	5	4	0	0	0	16	2	2	4	
L	0	2	20	0	2	0	0	0	0	7	4	3406	0	2	0	0	0	1	4	0	1	
M	13	1	0	2	0	0	1	21	0	0	2	0	3618	20	0	2	0	2	0	0	6	
N	26	1	1	5	0	0	0	17	0	3	12	4	15	5642	1	6	0	2	0	3	9	
O	4	4	5	70	3	0	6	1	0	0	1	3	3	0	17424	7	10	0	6	1	19	
P	10	3	0	34	0	1	0	1	0	1	1	0	0	1	2	5652	1	6	2	7	0	
Q	4	3	2	4	0	0	13	0	0	0	2	0	4	2	38	9	1688	8	2	0	9	
R	45	6	6	3	7	0	1	0	0	0	20	2	8	3	0	14	7	3335	0	0	2	
S	6	5	8	6	3	0	8	0	0	25	0	4	3	0	6	1	1	2	14284	1	11	
T	4	0	2	1	1	0	0	2	0	1	1	0	2	0	0	9	0	0	0	6621	0	
U	1	0	7	10	2	0	2	2	0	1	6	7	13	10	15	0	3	1	0	0	8521	
V	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5	121
W	5	0	0	0	0	0	3	4	0	0	4	2	8	66	2	0	0	1	0	2	32	
X	4	1	0	0	1	0	0	0	0	1	33	3	4	2	0	1	0	1	0	2	2	

Predicted	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
Actual																						
Y	1	3	3	3	0	0	0	5	0	30	0	1	2	2	0	20	1	0	0	15	8	1
Z	1	5	2	4	12	0	0	0	0	3	0	10	3	1	0	3	1	4	2	2	0	
All	4310	2644	7055	3016	3359	345	1687	2097	326	2522	1658	3491	3730	5827	17669	5764	1729	3419	14367	6699	8682	123

In [67]:
```
df5 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[4]})
create_grouped_bar_chart(df5)
```



Gaussian Naive_Bayes:

In [41]:
```
train_classifiers('Naive Bayes')
```

GaussianNB():
Accuracy: 0.5228
Precision: 0.5830
F1 Score: 0.4806
Recall: 0.5228
Classification Report:

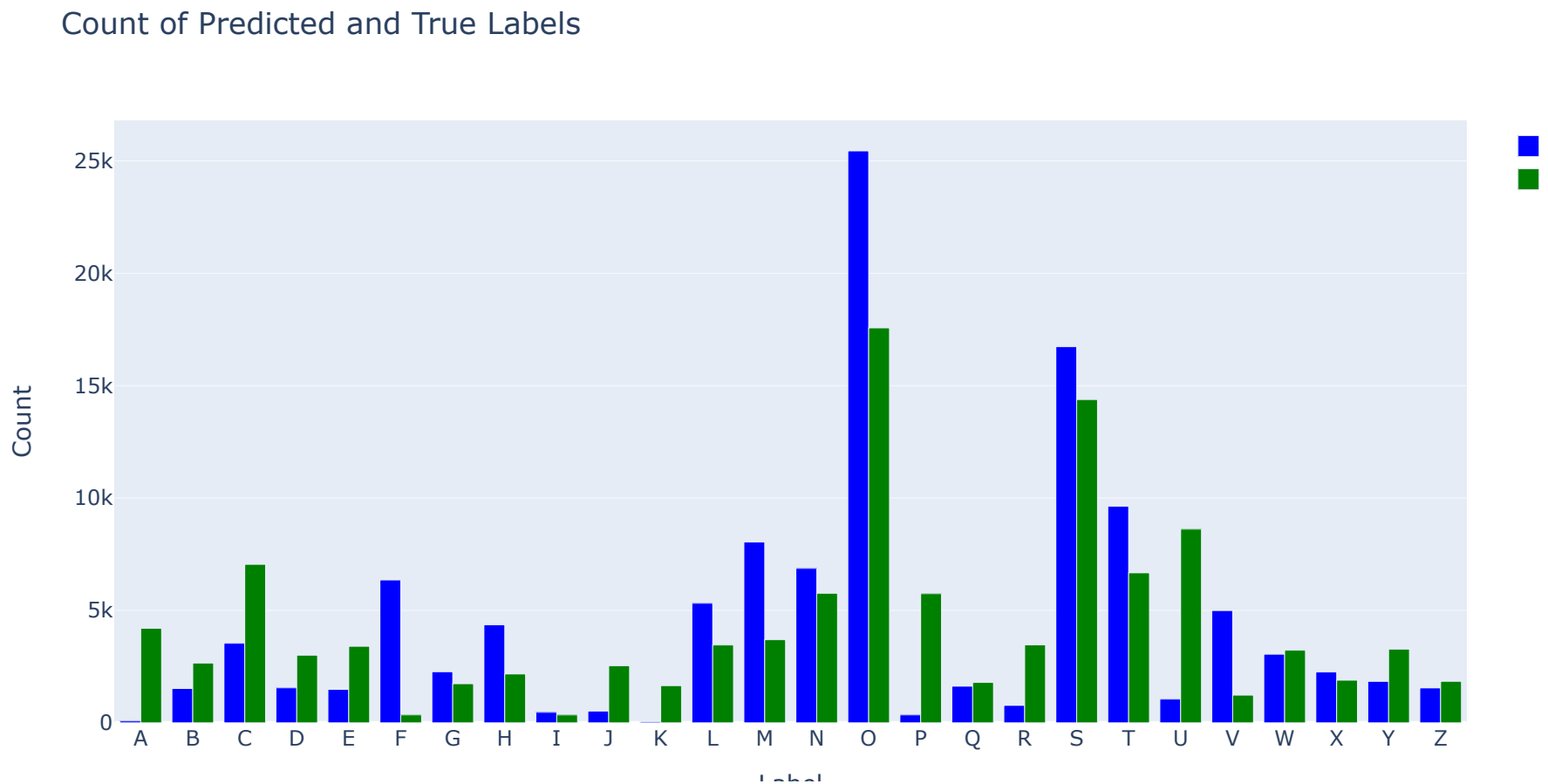
	precision	recall	f1-score	support
A	0.37	0.01	0.01	4204
B	0.43	0.25	0.32	2656
C	0.89	0.45	0.60	7048
D	0.63	0.32	0.43	3005
E	0.62	0.27	0.38	3402
F	0.05	0.96	0.10	351
G	0.40	0.52	0.45	1737
H	0.23	0.46	0.30	2172
I	0.49	0.66	0.56	350
J	0.39	0.08	0.13	2538
K	0.15	0.00	0.01	1650
L	0.52	0.80	0.63	3459
M	0.41	0.89	0.56	3694
N	0.48	0.57	0.52	5760
O	0.64	0.93	0.76	17569
P	0.60	0.04	0.07	5744
Q	0.48	0.43	0.45	1794
R	0.68	0.15	0.25	3466
S	0.70	0.81	0.75	14381
T	0.47	0.67	0.55	6671
U	0.77	0.09	0.17	8616
V	0.22	0.90	0.36	1230
W	0.59	0.55	0.57	3227
X	0.55	0.65	0.59	1894
Y	0.44	0.25	0.32	3275
Z	0.61	0.51	0.56	1843
accuracy			0.52	111736
macro avg	0.49	0.47	0.40	111736
weighted avg	0.58	0.52	0.48	111736

Out[41]:

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
74	1	11	25	13	150	1440	0	3	0	10	503	449	348	2	52	34	503	56	10	210	8	170	92
660	17	60	23	2	38	52	2	40	5	9	69	50	467	6	6	6	1003	22	13	14	18	7	21
42	3155	3	66	71	117	4	0	18	2	990	217	54	1817	26	96	5	111	94	3	14	81	0	55
96	1	972	1	9	34	4	2	26	0	0	87	44	1292	10	18	5	277	4	18	3	51	3	20
77	60	10	919	242	74	47	4	12	3	212	145	150	447	5	17	12	702	184	9	8	11	12	22
1	0	0	3	336	0	0	0	2	1	0	0	0	0	3	0	1	0	3	0	0	0	0	0
11	23	5	2	33	909	7	0	5	0	16	33	26	293	1	47	3	276	6	4	7	16	3	7
10	1	3	6	22	22	992	1	1	0	0	122	468	25	1	7	10	68	35	22	168	26	44	114
2	0	0	1	6	0	0	230	0	0	12	0	0	0	0	0	0	44	28	0	0	0	7	3
34	23	23	4	140	40	1	41	203	2	2	114	7	132	1	24	0	711	598	8	227	39	2	70
4	12	4	65	12	1	154	0	0	8	396	133	207	8	1	4	53	20	159	14	94	8	236	43
5	26	5	7	3	9	1	19	2	3	2750	15	41	49	0	6	6	37	88	9	94	53	86	79
3	1	4	1	1	0	108	0	0	0	0	3295	148	39	0	14	5	10	0	20	5	33	1	3
9	8	12	6	29	5	367	0	2	1	1	995	3286	164	0	6	11	35	66	26	428	200	51	48
79	79	147	14	9	120	21	0	2	0	0	177	101	16302	3	135	0	220	19	22	51	27	4	34
13	1	93	3	3472	9	128	1	20	1	2	145	82	130	212	17	7	68	1233	3	13	16	5	47
11	22	13	7	21	232	27	1	7	2	8	36	11	423	6	774	11	135	12	5	1	14	2	6
78	30	13	260	16	55	466	5	9	8	349	384	261	93	11	109	526	258	150	12	23	6	179	34
210	37	35	23	43	242	12	27	109	0	53	116	140	789	9	67	13	11670	533	0	68	59	28	52
7	0	11	1	1512	4	13	61	15	5	2	290	8	1	13	5	0	88	4488	4	9	6	20	22
50	30	86	23	47	195	347	2	17	2	91	652	959	2541	6	203	50	208	50	811	1526	573	34	98
2	0	4	3	59	0	7	0	0	0	0	1	19	0	1	0	1	0	6	7	1111	9	0	0
5	3	14	6	21	4	59	0	1	2	2	327	313	40	0	13	0	21	6	16	575	1786	3	9
4	0	3	2	9	1	34	15	1	5	35	44	26	0	2	4	6	44	155	6	95	3	1231	138

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
5	2	3	1	223	4	58	2	15	2	0	8	7	16	32	1	4	117	1612	4	248	1	89	819
32	3	21	14	3	4	11	53	9	0	378	133	12	28	1	3	5	106	25	6	0	7	41	5
1524	3535	1555	1486	6354	2269	4360	466	519	52	5318	8041	6869	25444	352	1628	774	16732	9632	1052	4992	3051	2258	1841

In [68]:
```
df6 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[5]})
create_grouped_bar_chart(df6)
```



AdaBoostClassifier:

In [43]:
```
train_classifiers('AdaBoost')
```

AdaBoostClassifier():
Accuracy: 0.5786
Precision: 0.5836
F1 Score: 0.5746
Recall: 0.5786
Classification Report:

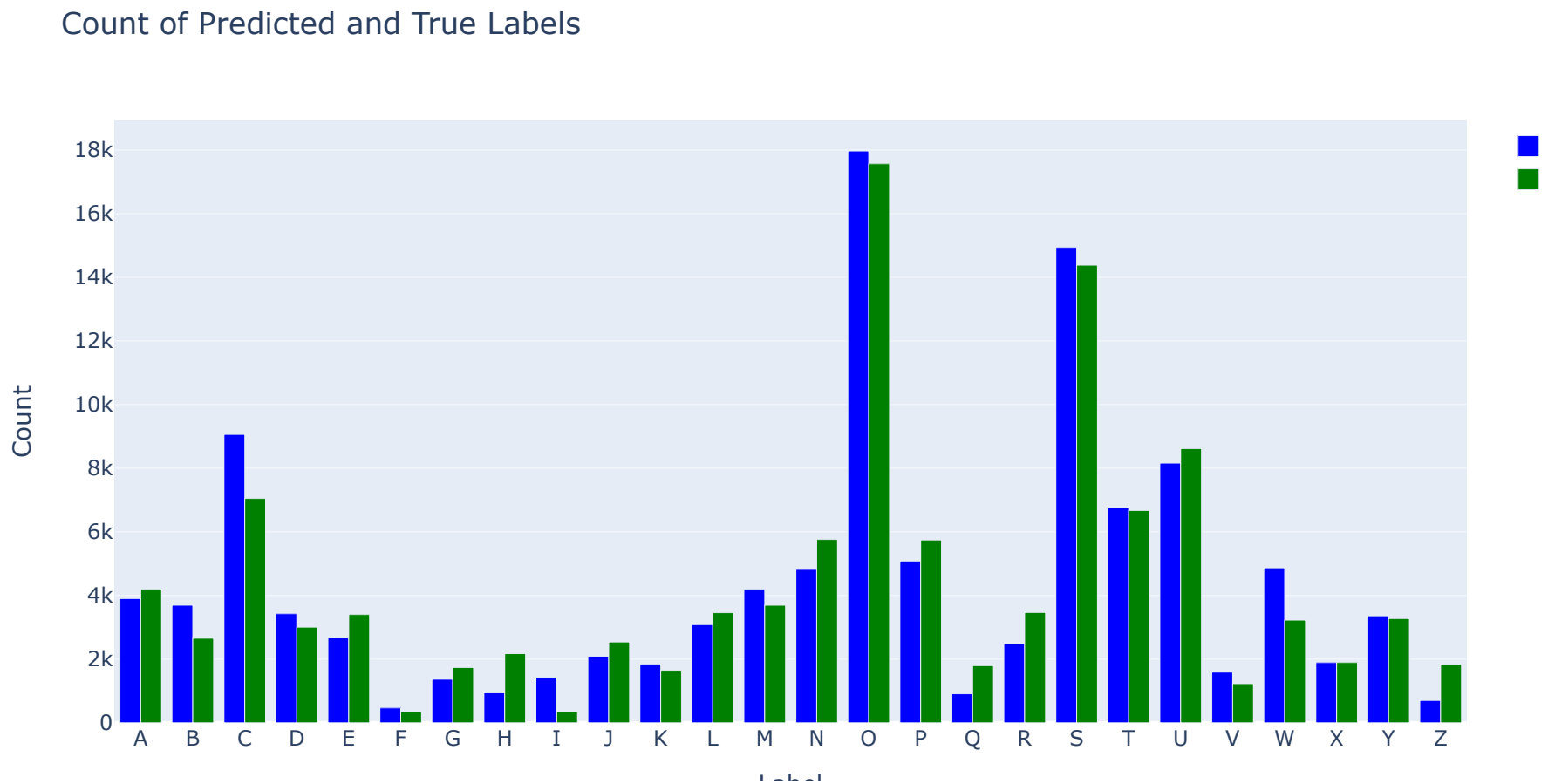
	precision	recall	f1-score	support
A	0.53	0.49	0.51	4204
B	0.32	0.44	0.37	2656
C	0.57	0.73	0.64	7048
D	0.46	0.53	0.49	3005
E	0.45	0.36	0.40	3402
F	0.48	0.64	0.55	351
G	0.15	0.12	0.13	1737
H	0.42	0.18	0.26	2172
I	0.18	0.75	0.29	350
J	0.45	0.37	0.41	2538
K	0.38	0.43	0.40	1650
L	0.67	0.60	0.63	3459
M	0.59	0.67	0.62	3694
N	0.37	0.31	0.34	5760
O	0.73	0.74	0.73	17569
P	0.75	0.66	0.70	5744
Q	0.47	0.24	0.31	1794
R	0.61	0.44	0.51	3466
S	0.66	0.69	0.67	14381
T	0.71	0.72	0.72	6671
U	0.62	0.58	0.60	8616
V	0.46	0.60	0.52	1230
W	0.38	0.57	0.46	3227
X	0.54	0.54	0.54	1894
Y	0.54	0.56	0.55	3275
Z	0.67	0.26	0.37	1843
accuracy			0.58	111736
macro avg	0.51	0.51	0.49	111736
weighted avg	0.58	0.58	0.57	111736

Out[43]:

Predicted / Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
A	2067	43	13	32	13	4	28	51	0	4	48	6	382	582	228	51	126	127	89	1	32	7
B	46	1179	29	104	122	2	38	14	4	6	2	3	9	76	378	19	22	31	411	0	69	0
C	10	28	5126	3	202	2	74	0	4	119	34	257	14	21	473	29	10	17	388	37	159	3
D	11	90	75	1589	5	1	68	1	4	26	1	3	1	34	508	38	5	4	321	3	109	0
E	29	258	300	16	1213	44	57	10	8	12	110	52	7	67	336	67	17	129	460	10	134	8
F	8	0	0	0	14	225	3	1	0	0	7	0	0	1	2	43	0	7	3	33	1	2
G	25	48	188	12	35	5	201	0	0	13	3	9	4	8	221	3	42	5	833	1	74	0
H	277	39	12	14	37	1	12	397	0	0	5	3	284	474	83	18	5	18	43	6	135	67
I	0	0	0	0	1	0	0	0	262	8	1	1	0	0	1	7	0	2	44	7	0	0
J	2	34	27	132	24	3	4	2	265	944	11	52	2	24	19	11	0	2	441	255	142	3
K	19	0	129	0	42	0	0	32	0	1	708	140	14	55	3	16	0	205	23	35	92	57
L	6	2	439	4	18	0	18	0	19	32	79	2060	0	3	0	0	0	9	388	41	119	11
M	158	3	9	5	4	1	1	48	0	1	5	0	2457	263	145	4	32	20	2	54	271	21
N	330	18	89	81	50	2	34	174	0	2	31	18	405	1783	180	16	19	31	31	36	427	271
O	9	238	1676	643	41	25	576	16	0	15	2	3	78	195	13052	4	42	5	451	1	255	9
P	325	2	20	64	115	65	6	29	1	11	25	4	4	123	228	3798	8	46	104	585	12	11
Q	40	49	98	64	37	1	96	0	0	16	4	2	8	22	613	8	424	41	152	5	94	0
R	286	57	99	18	260	2	7	7	0	0	152	9	113	301	168	142	116	1523	56	8	39	0
S	30	1452	166	306	236	13	21	2	467	536	32	168	5	36	527	48	14	42	9873	80	135	6
T	14	1	39	5	11	65	0	3	264	49	209	11	11	60	18	608	0	1	68	4824	12	1
U	82	77	472	249	91	5	100	57	0	120	95	54	219	446	678	7	14	44	134	9	5040	122
V	0	0	3	7	0	1	0	8	0	16	13	0	0	52	1	2	0	0	0	8	199	733
W	41	5	27	32	23	0	18	26	0	9	17	30	167	133	65	4	8	7	36	12	534	120
X	49	6	4	4	11	0	0	24	1	12	110	85	1	20	2	22	0	62	102	17	14	36

Predicted Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
Y	18	4	4	18	16	0	0	35	4	52	120	32	12	40	5	82	2	5	69	678	40	103
Z	22	62	18	26	45	0	6	4	131	88	23	81	1	2	37	36	5	110	418	11	22	0
All	3904	3695	9062	3428	2666	467	1368	941	1434	2092	1847	3083	4198	4821	17971	5083	911	2493	14940	6757	8160	1591

In [69]:
```
df7 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[6]})
create_grouped_bar_chart(df7)
```



XGBClassifier:

In [45]:

```
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Encode the target variable to numeric labels
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Train the XGBoost classifier
model = XGBClassifier()
Algorithm.append("XGBoost Classifier")
model.fit(X_train, y_train_encoded)

# Predict using the trained model
y_pred_encoded = model.predict(X_test)

# Decode the predicted labels back to alphabetical labels
y_pred = label_encoder.inverse_transform(y_pred_encoded)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Append the evaluation metrics to the respective lists
y_predct.append(y_pred)
Accuracy.append(accuracy)
Precision.append(precision)
F1_Score.append(f1)
Recall.append(recall)
# Print the evaluation metrics
print(f'{model}:')
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-Score: {f1:.4f}')
```

```python
# Generate the confusion matrix
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)

# Display the styled confusion matrix
confusion_matrix.style.background_gradient(cmap="tab10")
```

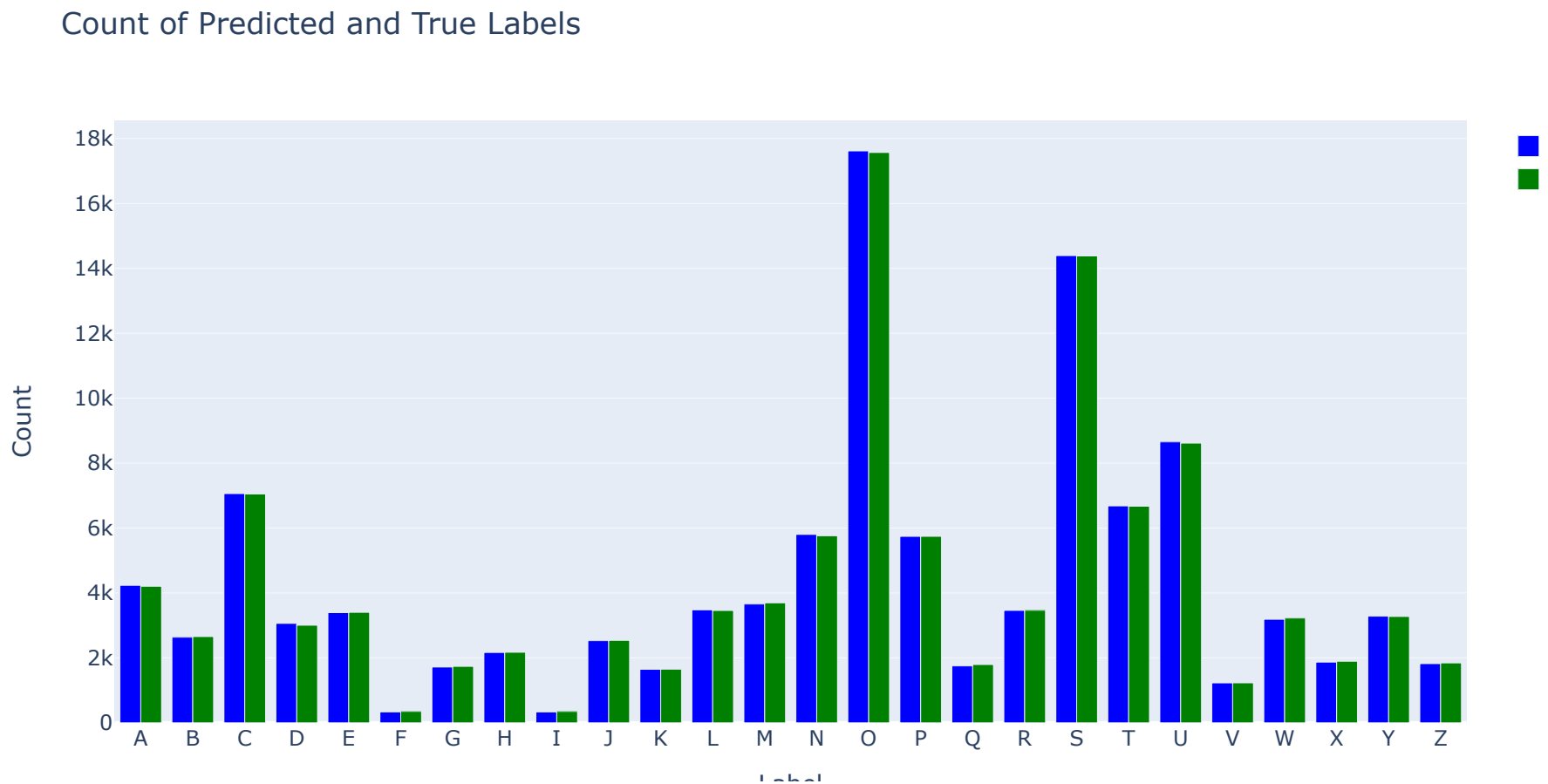
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...):
Accuracy: 0.9883
Precision: 0.9883
Recall: 0.9883
F1-Score: 0.9883
```

Out[45]:

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
0	1	3	0	0	1	10	0	0	2	2	6	8	3	2	2	1	3	1	3	0	1	0	2
0	1	9	3	0	3	0	0	0	2	0	1	1	13	2	0	4	4	0	1	0	0	0	1
3	6986	0	5	0	3	0	0	0	6	20	0	2	16	1	0	2	0	0	4	0	0	0	0
6	0	2946	0	0	0	0	0	7	0	0	0	0	36	8	1	0	0	0	0	0	0	0	0
2	23	2	3350	1	3	0	0	0	3	2	0	0	2	0	0	2	6	0	0	0	4	0	1
0	0	1	8	330	0	0	0	0	0	0	0	0	0	6	0	0	0	4	0	0	0	0	1
4	9	1	1	0	1692	0	0	4	0	0	0	0	5	0	9	0	5	2	1	0	0	0	0
1	0	0	0	0	0	2111	0	0	0	0	8	20	0	2	0	2	2	0	7	0	1	0	4
0	1	1	0	0	0	0	332	6	0	0	0	0	0	0	1	0	7	0	0	0	0	0	0
0	0	4	0	0	1	2	1	2476	0	2	0	3	0	2	0	0	21	12	5	0	0	0	3
0	3	0	3	0	0	6	0	0	1599	9	2	0	0	0	0	10	0	0	5	0	0	8	3
0	15	2	2	0	0	0	0	0	2	3432	0	0	0	0	0	2	2	0	0	0	2	0	0
1	0	2	0	0	1	19	0	0	0	1	3610	26	5	4	0	3	0	0	6	0	8	0	1
0	0	3	2	0	0	11	0	2	2	0	4	5701	0	2	0	0	0	2	10	0	10	1	2
2	4	46	2	0	0	0	0	2	0	0	2	0	17491	2	4	0	0	0	14	0	0	0	0
2	0	16	0	0	0	0	0	0	0	0	0	2	3	5699	1	6	0	6	0	2	0	0	5
0	4	1	0	0	9	0	0	0	0	0	2	0	27	2	1727	7	6	0	5	0	2	0	0
2	3	2	6	0	0	0	0	0	4	2	2	0	0	0	2	3409	1	0	2	0	2	1	4
6	4	8	3	0	3	0	0	12	0	0	2	2	4	2	2	0	14326	0	2	0	2	0	0
0	2	1	0	0	0	2	0	3	0	0	0	0	1	4	0	0	0	6650	0	0	2	0	6
0	0	8	0	0	2	4	0	0	3	0	4	2	11	0	2	6	2	0	8565	3	2	0	0
0	0	0	0	3	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1212	0	0	13
0	0	2	0	0	0	0	0	0	0	0	7	36	0	0	0	0	0	0	23	4	3152	0	1
0	0	0	0	0	0	0	0	0	19	3	4	0	0	0	0	0	1	0	0	1	0	1850	12

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	2	0	0	0	0	0	0	14	4	0	2	0	0	5	0	0	4	4	0	6	0	8	3224
6	2	4	8	0	0	0	0	2	0	5	1	0	0	0	0	8	2	0	0	0	0	0	0
6	7060	3062	3393	334	1718	2165	333	2530	1646	3478	3657	5803	17617	5743	1751	3462	14392	6681	8653	1228	3188	1868	3283

In [70]:
```
df8 = pd.DataFrame({'True Label': y_test, 'Predicted Label': y_predct[7]})
create_grouped_bar_chart(df8)
```



In [52]:
```
import pickle
# Save the model and evaluation metrics to a pickle file
xgboost = {
    'model': model,
    'accuracy': accuracy,
    'precision': precision,
    'recall': recall,
    'f1_score': f1
}

with open('xgboost_model_results.pkl', 'wb') as file:
    pickle.dump(xgboost, file)
```

DataFrame to store algorithm names and corresponding accuracies :

In [47]:
```
# Create a DataFrame to store algorithm names and corresponding accuracies
df= pd.DataFrame({"Algorithm": Algorithm, "Accuracy": Accuracy,"Precision":Precision,"F1_Score":F1_Score,"Recall":Rec

# Print the DataFrame
df
```

Out[47]:

	Algorithm	Accuracy	Precision	F1_Score	Recall
0	Logistic Regression	0.873792	0.872685	0.872952	0.873792
1	Decision Tree	0.947806	0.947743	0.947721	0.947806
2	Random Forest	0.985009	0.985030	0.984962	0.985009
3	KNeighborsClassifier	0.966421	0.966630	0.966156	0.966421
4	Support Vector Machines Classifier	0.978458	0.978470	0.978423	0.978458
5	Naive Bayes Classifier	0.522822	0.583013	0.480586	0.522822
6	AdaBoost Classifier	0.578587	0.583618	0.574587	0.578587
7	XGBoost Classifier	0.988276	0.988288	0.988267	0.988276

In [53]:
```python
import pickle

# Create a dictionary to store the results
results = {
    'Algorithm': 'Random Forest',
    'Accuracy': 0.985009,
    'Precision': 0.985030,
    'F1_Score': 0.984962,
    'Recall': 0.985009
}

# Save the results to a pickle file
with open('random_forest_results.pkl', 'wb') as file:
    pickle.dump(results, file)
```

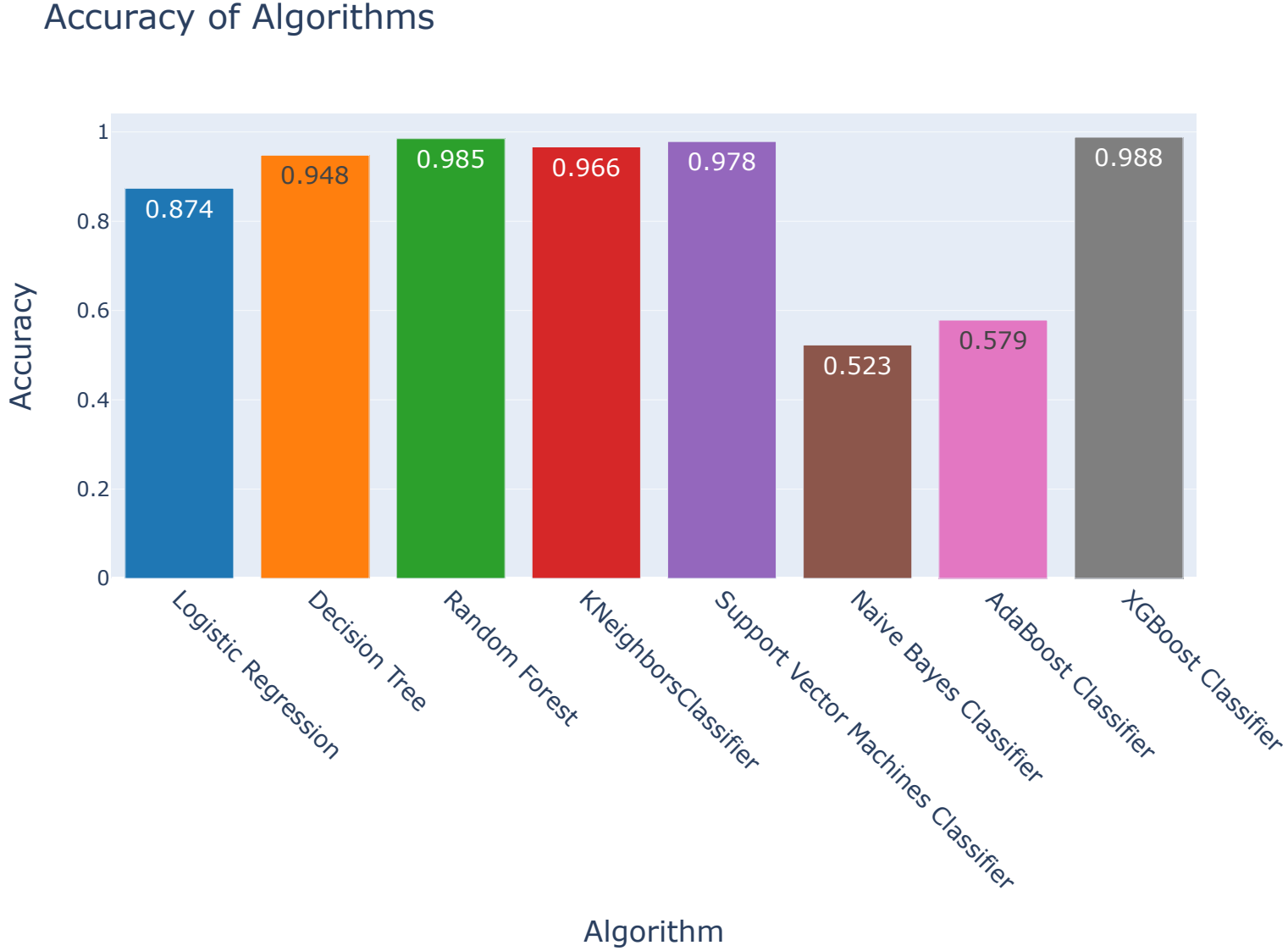
Accuracy of Algorithms:

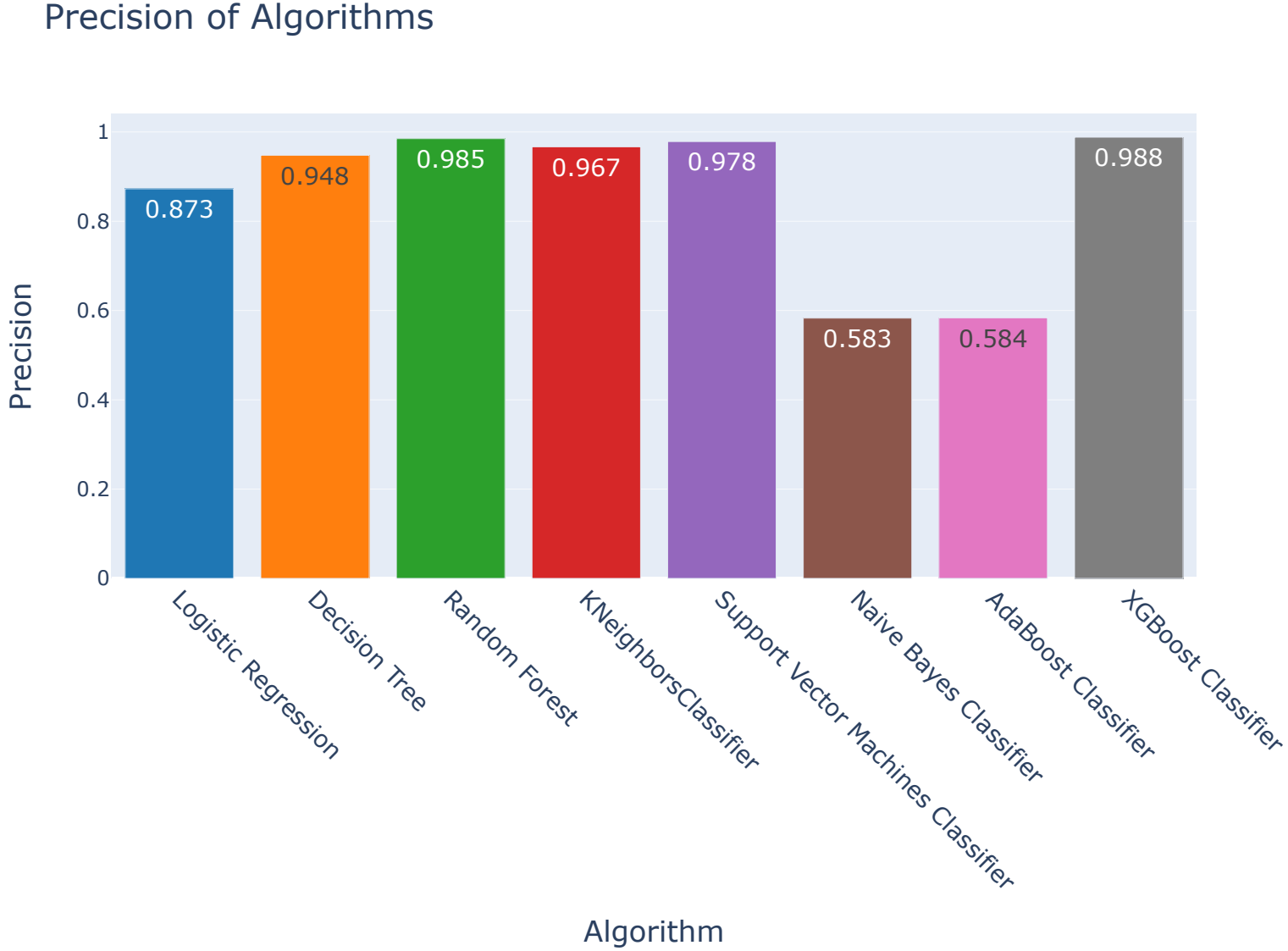
In [49]:

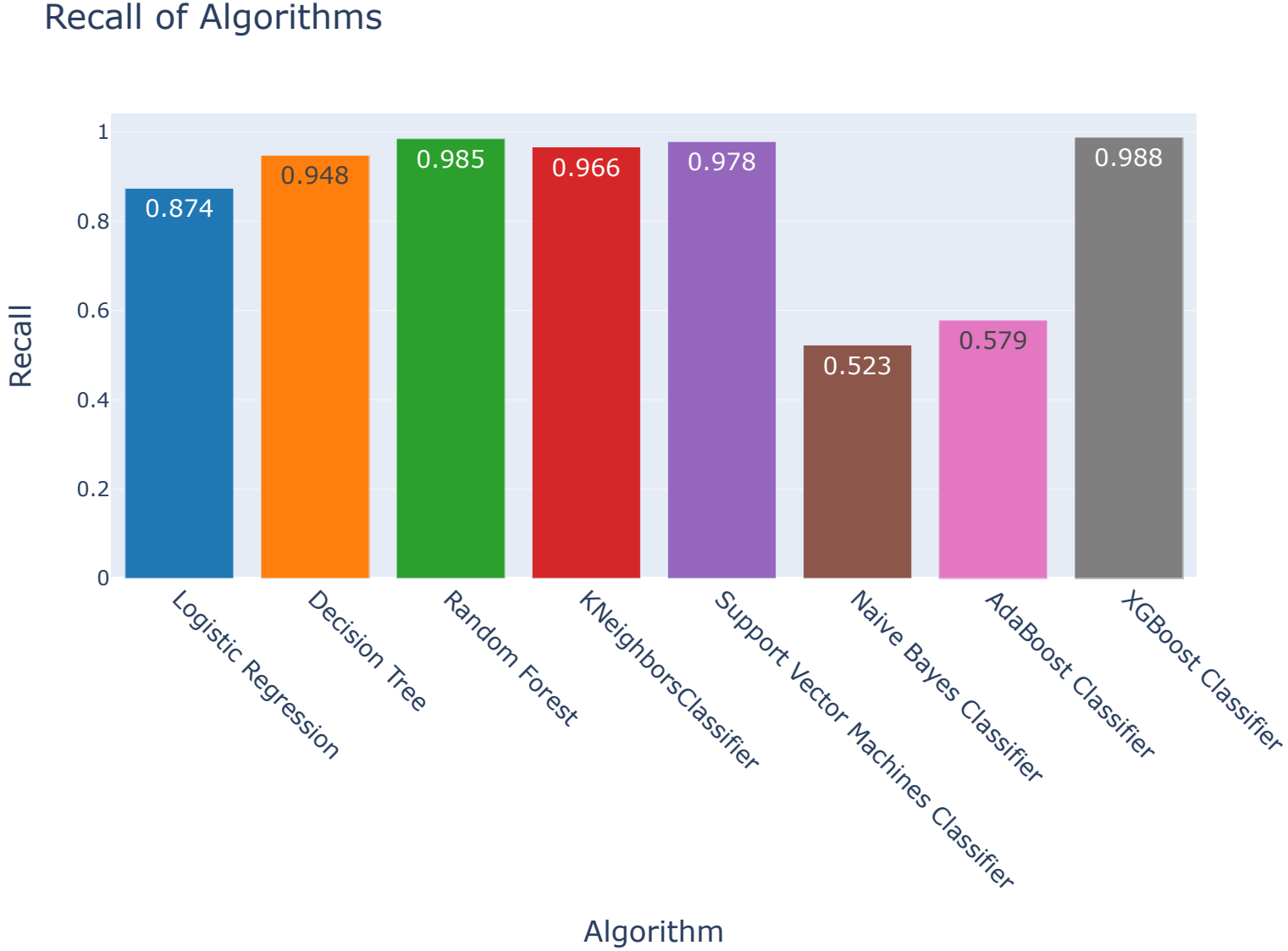
```
import plotly.graph_objects as go
import plotly.colors as colors

# Define the colors for each algorithm
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f']

# Create the Accuracy plot using Plotly
fig_accuracy = go.Figure()
fig_accuracy.add_trace(go.Bar(
    x=df['Algorithm'],
    y=df['Accuracy'],
    name='Accuracy',
    marker_color=colors
))
fig_accuracy.update_layout(
    title='Accuracy of Algorithms',
    xaxis_title='Algorithm',
    yaxis_title='Accuracy',
    width=800,
    height=600,
    font=dict(size=14)
)
fig_accuracy.update_xaxes(tickangle=45)
fig_accuracy.update_yaxes(tickfont=dict(size=12))

# Create the Precision plot using Plotly
fig_precision = go.Figure()
fig_precision.add_trace(go.Bar(
    x=df['Algorithm'],
    y=df['Precision'],
    name='Precision',
    marker_color=colors
))
fig_precision.update_layout(
    title='Precision of Algorithms',
    xaxis_title='Algorithm',
    yaxis_title='Precision',
    width=800,
    height=600,
    font=dict(size=14)
)
```

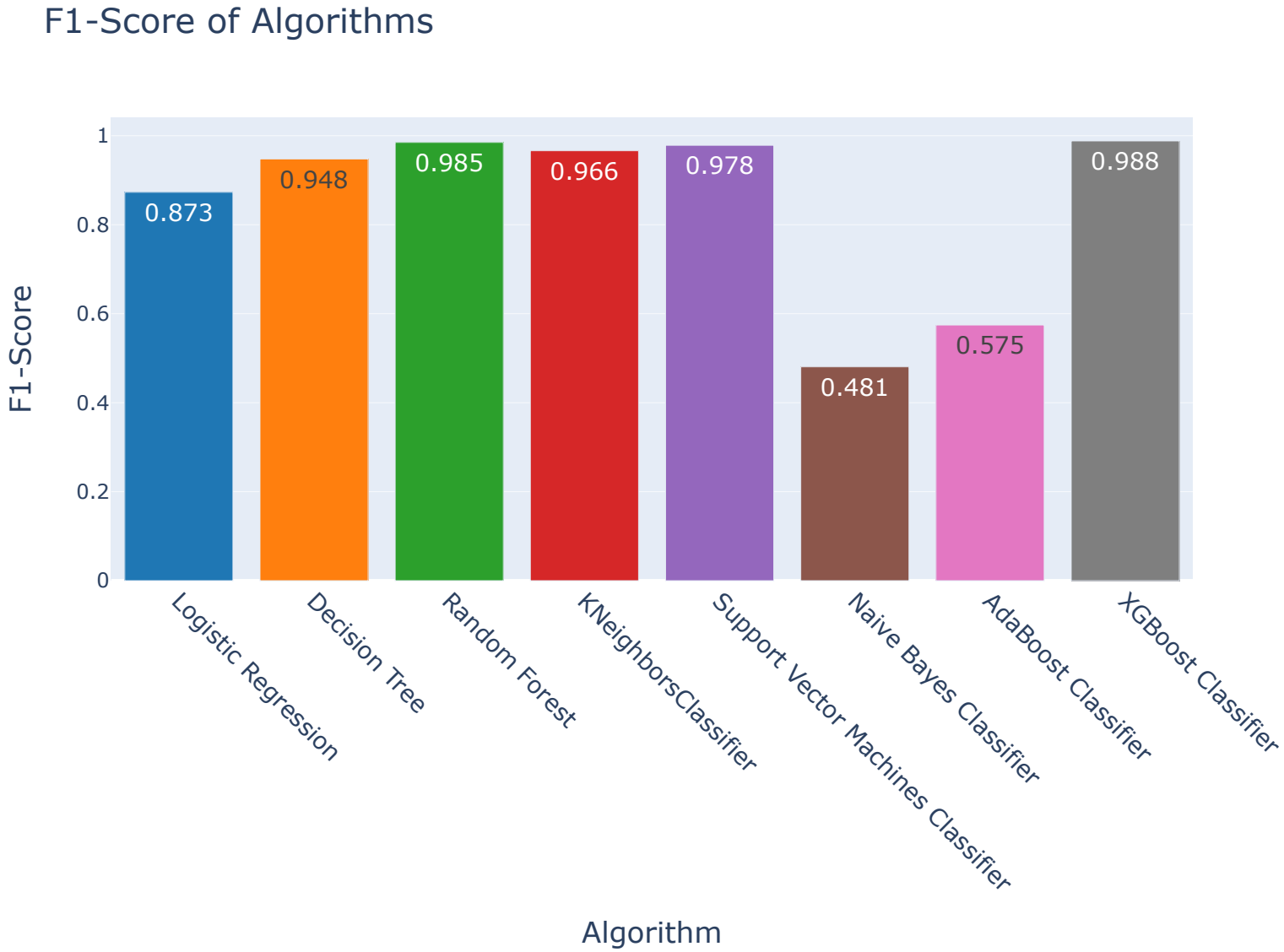
```
fig_precision.update_xaxes(tickangle=45)
fig_precision.update_yaxes(tickfont=dict(size=12))

# Create the Recall plot using Plotly
fig_recall = go.Figure()
fig_recall.add_trace(go.Bar(
    x=df['Algorithm'],
    y=df['Recall'],
    name='Recall',
    marker_color=colors
))
fig_recall.update_layout(
    title='Recall of Algorithms',
    xaxis_title='Algorithm',
    yaxis_title='Recall',
    width=800,
    height=600,
    font=dict(size=14)
)
fig_recall.update_xaxes(tickangle=45)
fig_recall.update_yaxes(tickfont=dict(size=12))

# Create the F1-Score plot using Plotly
fig_f1_score = go.Figure()
fig_f1_score.add_trace(go.Bar(
    x=df['Algorithm'],
    y=df['F1_Score'],
    name='F1-Score',
    marker_color=colors
))
fig_f1_score.update_layout(
    title='F1-Score of Algorithms',
    xaxis_title='Algorithm',
    yaxis_title='F1-Score',
    width=800,
    height=600,
    font=dict(size=14)
)
fig_f1_score.update_xaxes(tickangle=45)
fig_f1_score.update_yaxes(tickfont=dict(size=12))

# Add value labels on top of each bar
```

```
for fig in [fig_accuracy, fig_precision, fig_recall, fig_f1_score]:
    for trace in fig.data:
        fig.update_traces(text=trace.y, textposition='auto', texttemplate='%{text:.3f}')

# Display the plots
fig_accuracy.show()
fig_precision.show()
fig_recall.show()
fig_f1_score.show()
```


Accuracy of Algorithms
Accuracy
1
0.8
0.6
0.4
0.2
0
0.874
0.948
0.985
0.966
0.978
0.523
0.579
0.988
Logistic Regression
Decision Tree
Random Forest
KNeighborsClassifier
Support Vector Machines Classifier
Naive Bayes Classifier
AdaBoost Classifier
XGBoost Classifier
Algorithm


Precision of Algorithms
Precision
1
0.8
0.6
0.4
0.2
0
0.873
0.948
0.985
0.967
0.978
0.583
0.584
0.988
Logistic Regression
Decision Tree
Random Forest
KNeighborsClassifier
Support Vector Machines Classifier
Naive Bayes Classifier
AdaBoost Classifier
XGBoost Classifier
Algorithm


Recall of Algorithms
Recall
1
0.8
0.6
0.4
0.2
0
0.874
0.948
0.985
0.966
0.978
0.523
0.579
0.988
Logistic Regression
Decision Tree
Random Forest
KNeighborsClassifier
Support Vector Machines Classifier
Naive Bayes Classifier
AdaBoost Classifier
XGBoost Classifier
Algorithm



Observations:

- The algorithms with the highest accuracy, precision, F1-score, and recall are XGBoost Classifier and Random Forest.

- Naive Bayes Classifier and AdaBoost Classifier have lower accuracy, precision, F1-score, and recall compared to other algorithms.
- Logistic Regression, Decision Tree, KNeighborsClassifier, and Support Vector Machines Classifier perform well with reasonably high scores in all metrics.
- XGBoost Classifier has the highest scores in all metrics, indicating its superior performance in this evaluation.
- Naive Bayes Classifier has significantly lower scores compared to other algorithms, suggesting that it may not be suitable for this specific dataset.

Conclusions:

- Based on the evaluation of different algorithms on the given dataset, it can be concluded that XGBoost Classifier and Random Forest are the top-performing algorithms.
- These algorithms consistently exhibit high accuracy, precision, F1-score, and recall, indicating their effectiveness in predicting the target variable.
- On the other hand, Naive Bayes Classifier performs poorly in comparison to the other algorithms.
- Therefore, for this dataset, it is recommended to consider XGBoost Classifier or Random Forest as the preferred algorithms for accurate predictions.

Model should be light for deployment.

For deployment purposes, it is recommended to choose a model that is light in terms of size. This means selecting a model that has a smaller memory footprint and requires less storage space. Models like Decision Tree, Logistic Regression, and Naive Bayes Classifier, which have lower accuracy but are relatively simpler and smaller, can be considered in scenarios where lightweight models are preferred.

Model should have very less latency.

If low latency is a critical requirement, it is advisable to select models that can make predictions quickly without significant delays. In this case, algorithms like Decision Tree, Support Vector Machines (SVM), and Logistic Regression can be suitable choices, as they have relatively high accuracy and are known for their fast prediction times.

In [60]:
```python
import pickle

# Load the results from the pickle file
with open('xgboost_model_results.pkl', 'rb') as file:
    results = pickle.load(file)

# Access the loaded results
algorithm = results['model']
accuracy = results['accuracy']
precision = results['precision']
f1_score = results['f1_score']
recall = results['recall']

# Print the loaded results
print(f'Algorithm: {algorithm}')
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'F1 Score: {f1_score}')
print(f'Recall: {recall}')
```

```
Algorithm: XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...)
Accuracy: 0.9882759361351757
Precision: 0.9882879448467651
F1 Score: 0.9882672361837063
Recall: 0.9882759361351757
```

In [ ]:

In [ ]: