INTRODUCTION TO DATA SCIENCE

PROJECT REPORT

LNMIIT

The LNM Institute of Information Technology

Course Instructors:

Dr. Lal Upendra Pratap Singh

Dr. Subrat Das

Dr. Aloke Dutta

Team Members:

Soham Khatod(21UCS206)

Chinmay Moondra (21UCS252)

Arnav Gaur (21UCS242)

Yuvraj Chawla (21UCS239)

Introduction

Problem Statement:

Applying ML Classification algorithms on the data set and getting inferences from the data. You may use the appropriate ML algorithm and know the concept behind it.

Dataset chosen :-

Link - https://archive.ics.uci.edu/dataset/59/letter+recognition

Dataset name = Letter Recognition

Dataset Information: -

The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. We typically train on the first 16000 items and then use the resulting model to predict the letter category for the remaining 4000. See the article cited above for more details.

Dataset Variable/Column Information: -

	Variable	Description	Datatype
1	lettr	capital letter	(26 values from A to Z)
2	x-box	horizontal position of box	(integer)
3	y-box	vertical position of box	(integer)
4	width	width of box	(integer)
5	high	height of box	(integer)
6	onpix	total # on pixels	(integer)
7	x-bar	mean x of on pixels in box	(integer)
8	y-bar	mean y of on pixels in box	(integer)
9	x2bar	mean x variance	(integer)
10	y2bar	mean y variance	(integer)
11	xybar	mean x y correlation	(integer)
12	x2ybr	mean of x * x * y	(integer)
13	xy2br	mean of x * y * y	(integer)

14	x-ege	mean edge count left to right	(integer)
15	xegvy	correlation of x-ege with y	(integer)
16	y-ege	mean edge count bottom to top	(integer)
17	yegvx	correlation of y-ege with x	(integer)

Importing dataset:-

Install the ucimlrepo package

```
pip install ucimlrepo
```

Import the dataset into your code

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
letter_recognition = fetch_ucirepo(id=59)

# data (as pandas dataframes)
X = letter_recognition.data.features
y = letter_recognition.data.targets

# metadata
print(letter_recognition.metadata)

# variable information
print(letter_recognition.variables)
```

View the full documentation

```
pip install ucimlrepo
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
letter_recognition = fetch_ucirepo(id=59)

# data (as pandas dataframes)
X = letter_recognition.data.features
y = letter_recognition.data.targets

# metadata
print(letter_recognition.metadata)

# variable information
print(letter_recognition.variables)
```

Dataset metadata:

	Name	Role	Type	Demographic	Description
0	lettr	Target	Categorical	None	capital letter
1	x-box	Feature	Integer	None	horizontal position of box
2	y-box	Feature	Integer	None	vertical position of box
3	width	Feature	Integer	None	width of box
4	high	Feature	Integer	None	height of box
5	onpix	Feature	Integer	None	total # on pixels
6	x-bar	Feature	Integer	None	mean x of on pixels in box
7	y-bar	Feature	Integer	None	mean y of on pixels in box
8	x2bar	Feature	Integer	None	mean x variance
9	y2bar	Feature	Integer	None	mean y variance
10	xybar	Feature	Integer	None	mean x y correlation
11	x2ybr	Feature	Integer	None	mean of x * x * y
12	xy2br	Feature	Integer	None	mean of x * y * y
13	x-ege	Feature	Integer	None	mean edge count left to right
14	xegvy	Feature	Integer	None	correlation of x-ege with y
15	y-ege	Feature	Integer	None	mean edge count bottom to top
16	yegvx	Feature	Integer	None	correlation of y-ege with x

Units Missing

	Units Missing	values
0	None	no
1	None	no
2	None	no
3	None	no
4	None	no
5	None	no
6	None	no
7	None	no
8	None	no
9	None	no
10	None	no
11	None	no
12	None	no
13	None	no
14	None	no
15	None	no
16	None	no

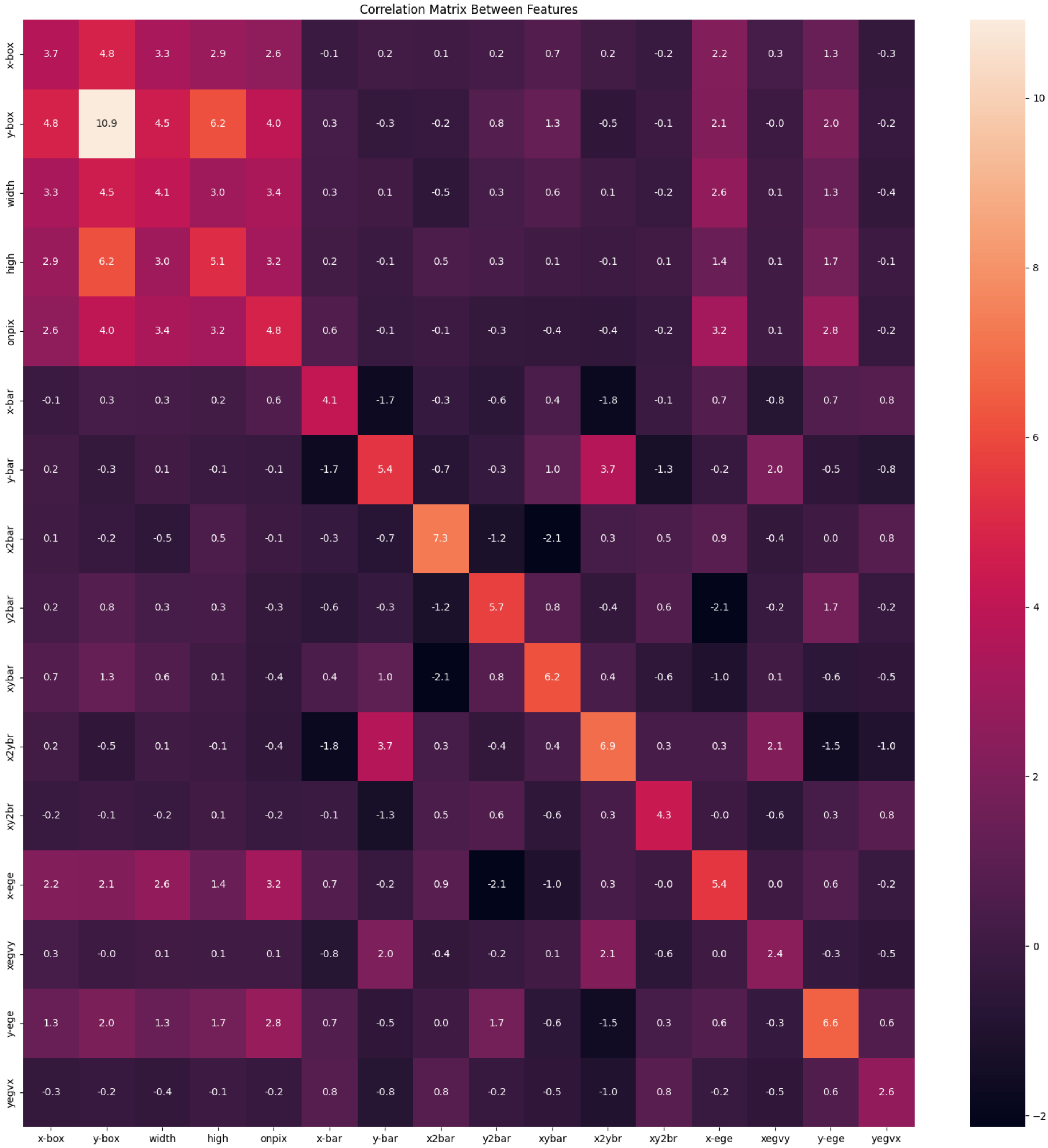
Glimpse of the dataset

```
import pandas as pd
df = pd.concat([X,y],axis =1)
df.head()
```

	x-box	y-box	width	high	onpix	x-bar	y-bar	x2bar	y2bar	xybar	x2ybr	xy2br	x-ege	xegvy	y-ege	yegvx	lettr
0	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8	T
1	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10	I
2	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9	D
3	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8	N
4	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10	G

Analysis

Covariance Matrix (Heatmap):-



Strong Positive Covariances:

Features with strong positive covariances indicate that as one feature increases, the other tends to increase as well. This suggests a positive linear relationship between these features.

Strong Negative Covariances:

Negative covariances between features suggest an inverse relationship, where an increase in one feature coincides with a decrease in the other. This indicates a negative linear relationship.

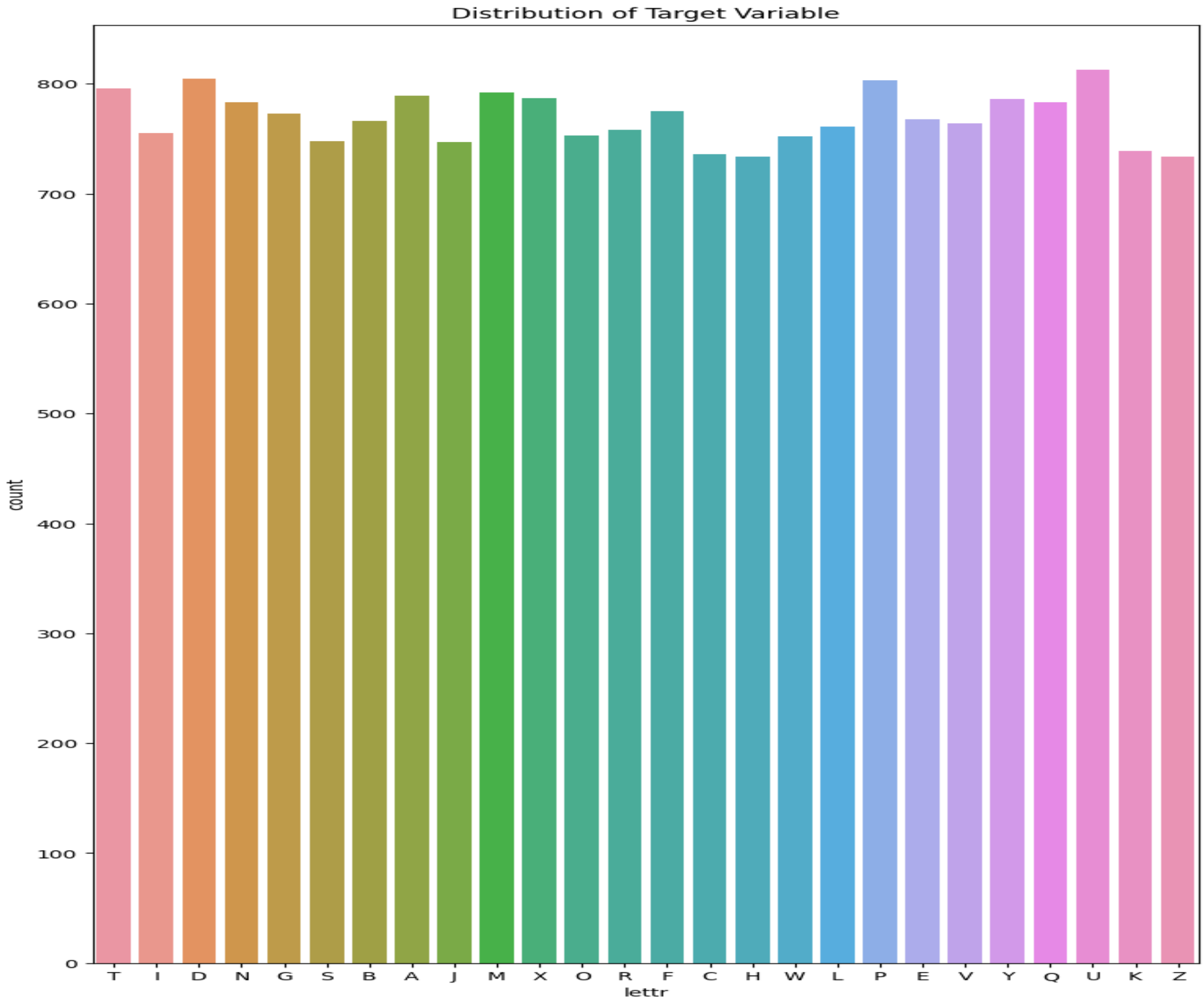
Observation

Clearly we can see that features [x-box , y-box , width , high] are having strong positive covariances between themselves.

Also there are a few strong negative covariance between [xybar and x2bar] and [y2bar,xegvy]

Bar plot: -

Bar plot for the count/frequency of letter (target variable).



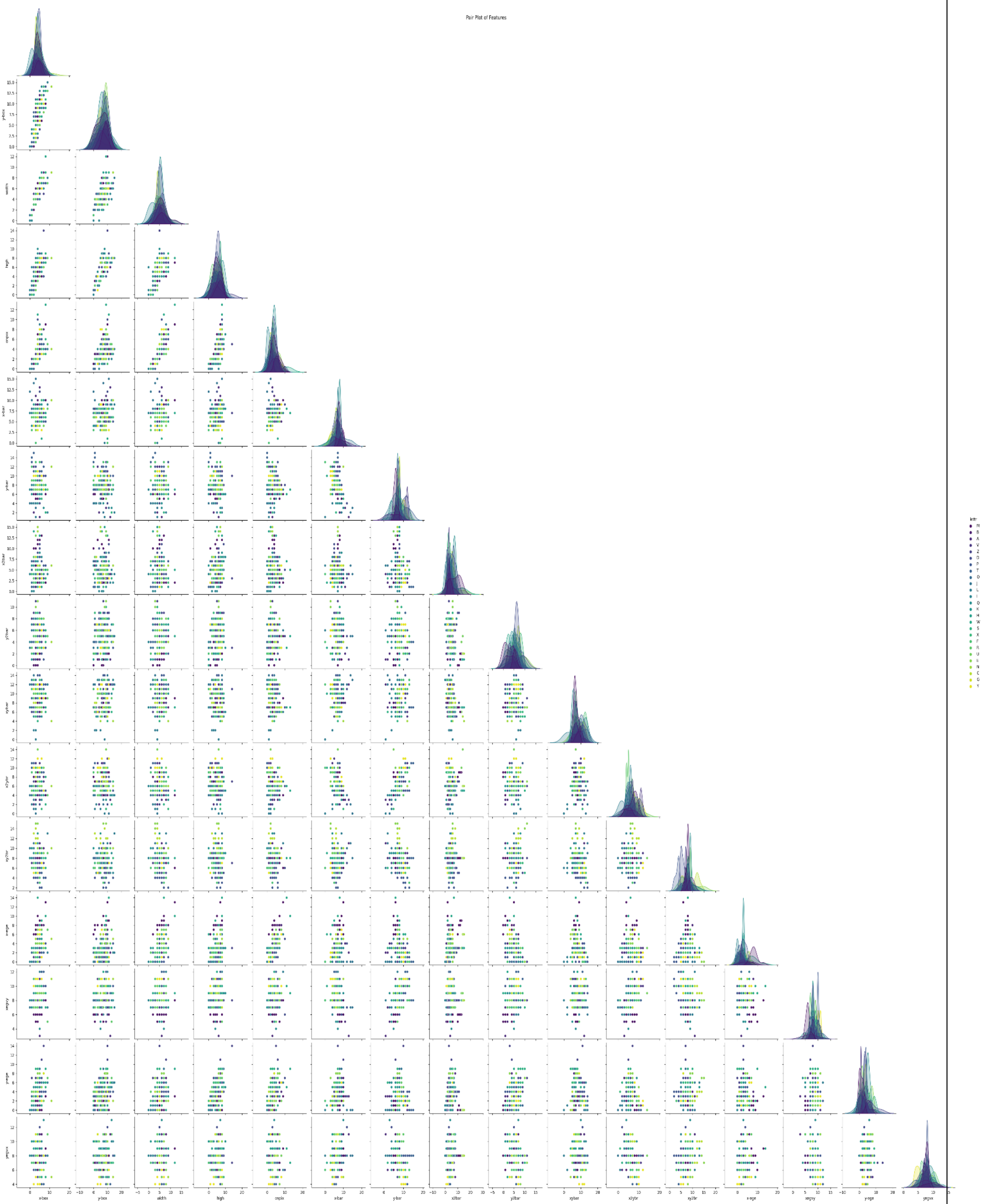
We can see that all the values of target variable is adequately represented and there is no huge disparity in number of points for a particular value.

Pair Plot (or scatterplot matrix): -

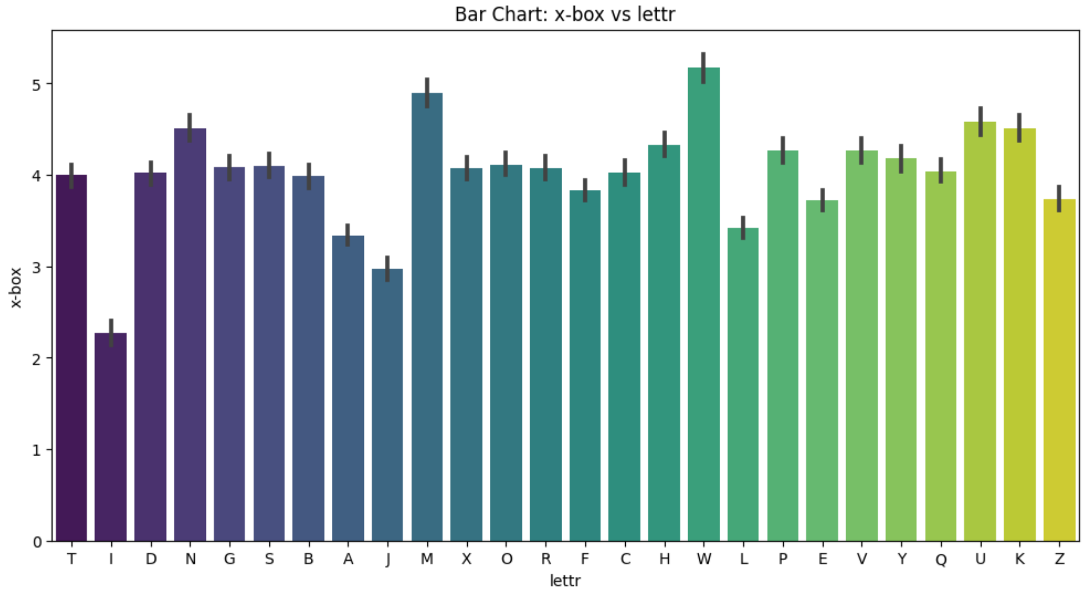
1. **Main Diagonal Histograms:**
    - Along the diagonal of the pair plot, instead of scatterplots, histograms are plotted for each variable in the dataset. Each histogram shows the distribution of values for a specific variable.
2. **Interpretation:**
    - The main diagonal histograms provide insights into the shape, central tendency, and spread of each variable in isolation. You can observe whether a variable

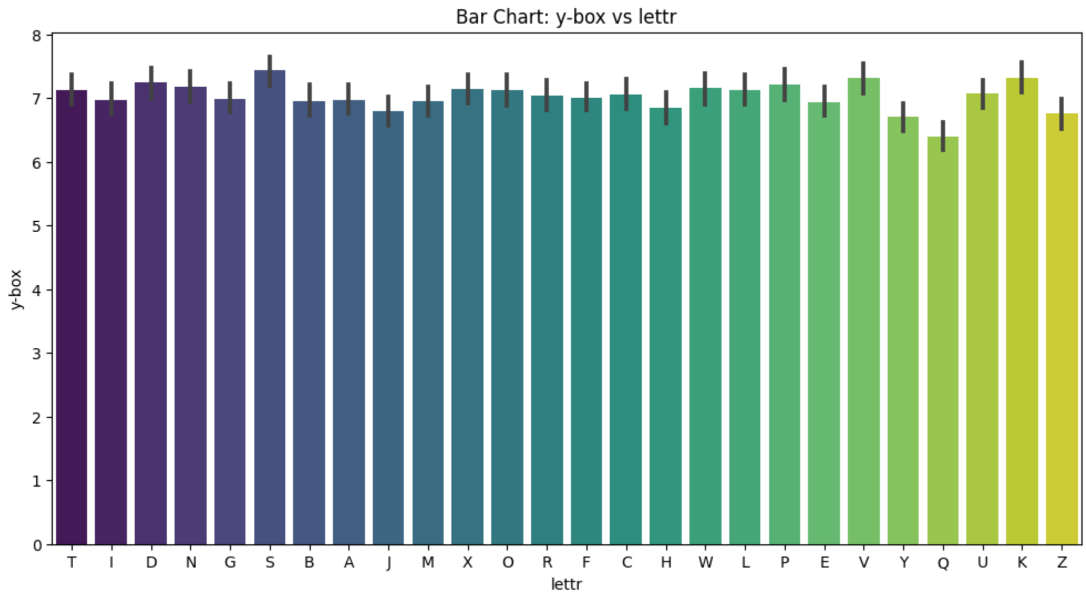
follows a normal distribution, has a skewed distribution, or exhibits any other characteristic patterns.

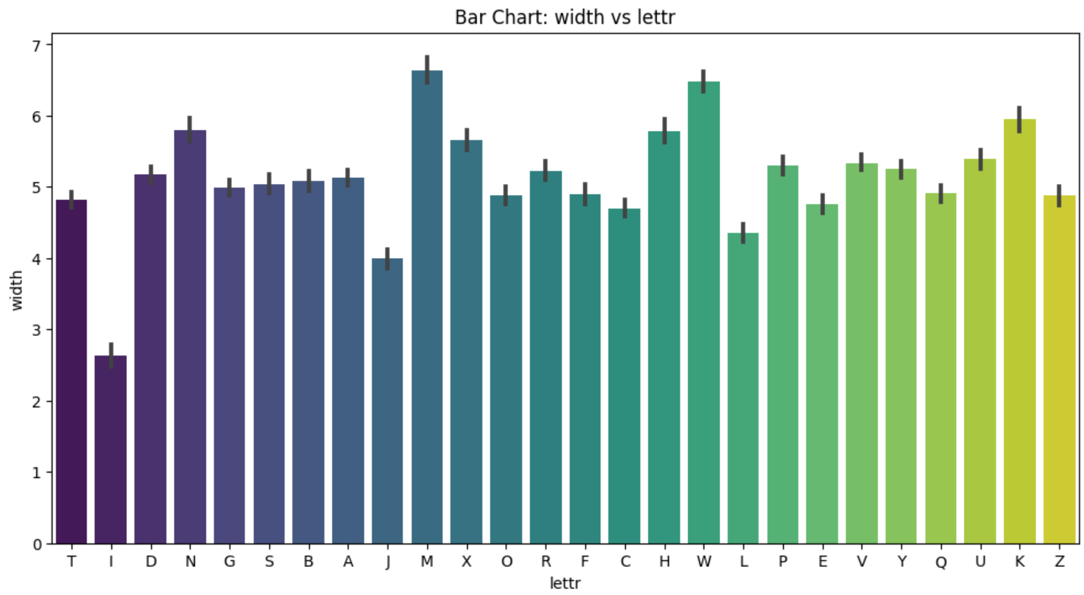
3. **Frequency Distribution:**
    - The height of the bars in each histogram represents the frequency or count of observations falling into different bins. This visual representation helps you understand how values are distributed across the range of each variable.
4. **Identifying Outliers:**
    - Outliers or unusual patterns in the distribution of a variable can be detected by examining the shape of the histogram. For example, a long tail on one side might indicate the presence of outliers.
5. **Visualization of Univariate Distributions:**
    - While the scatterplots in the rest of the pair plot show bivariate relationships between pairs of variables, the histograms on the main diagonal provide a complementary view by focusing on the univariate distribution of each variable.

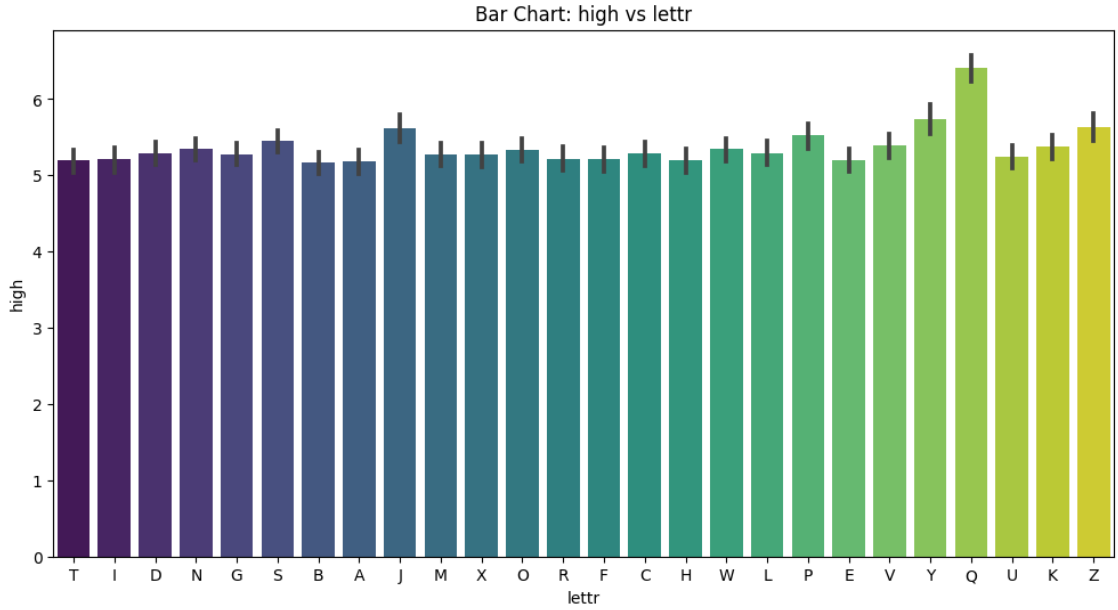

Pair Plot of Features

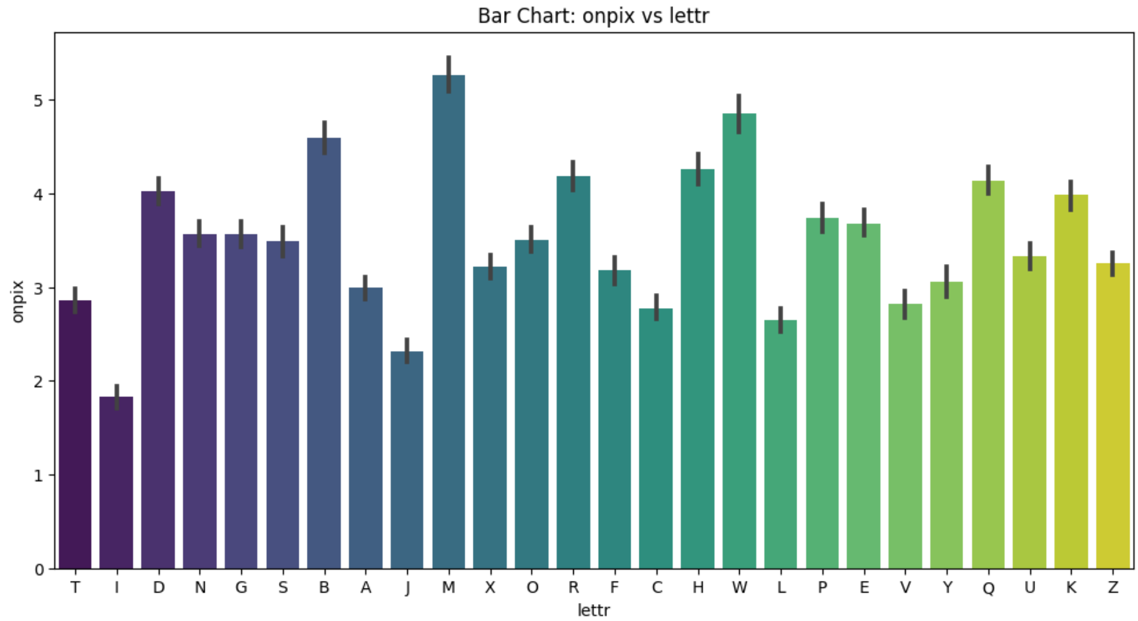
Bar chart between features and target variable:-

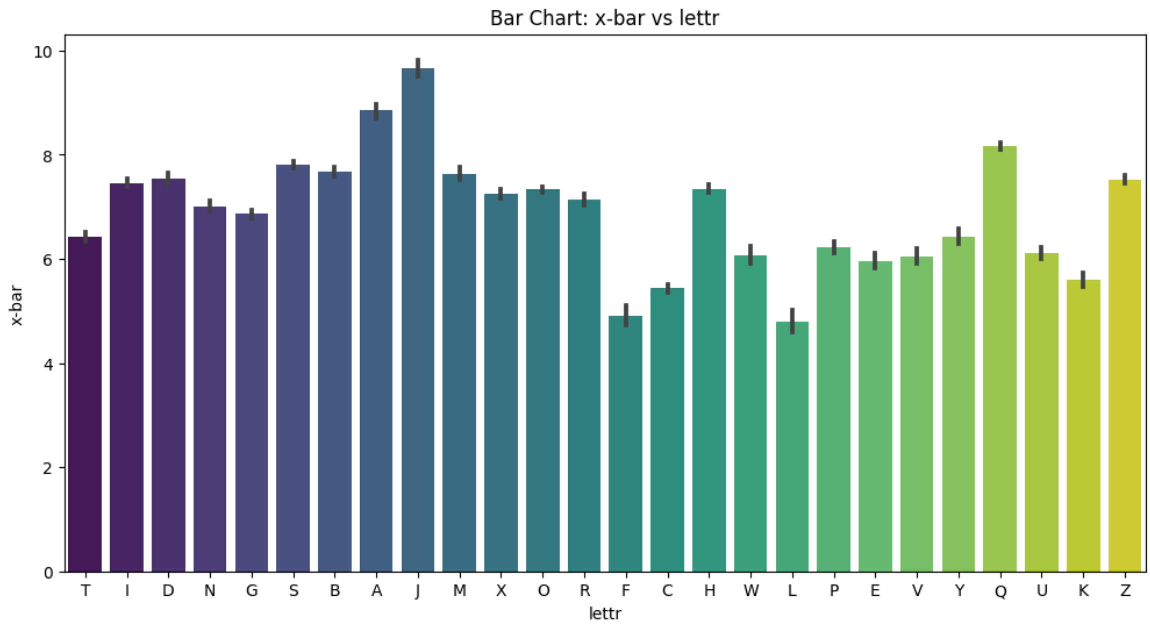


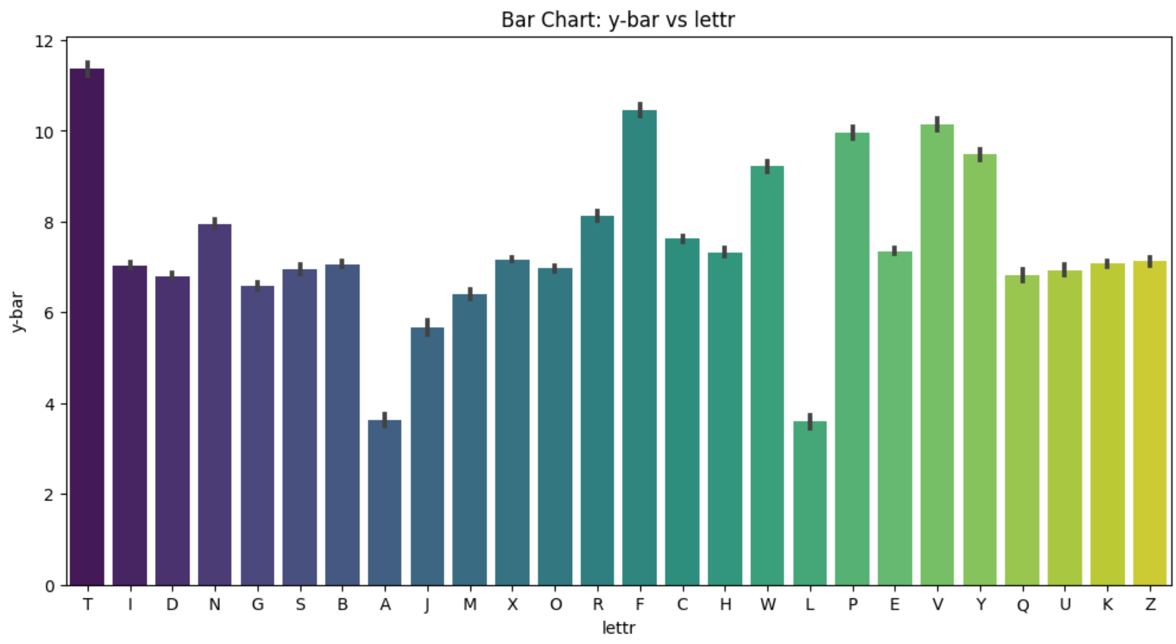




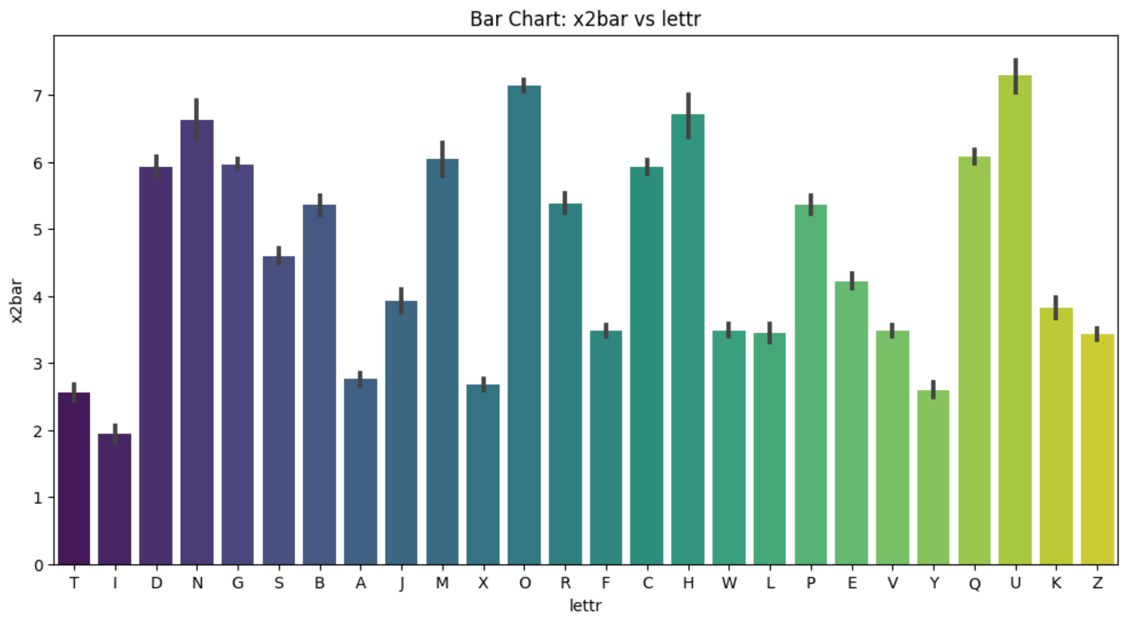

Bar Chart: high vs lettr
high
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z


Bar Chart: onpix vs lettr
onpix
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z

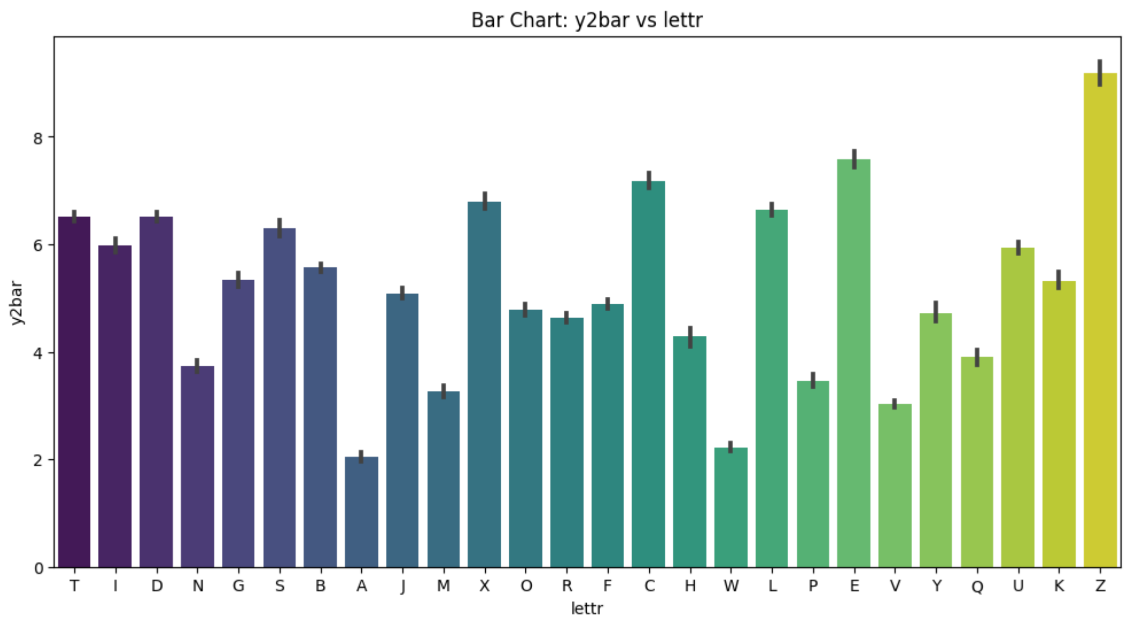

Bar Chart: x-bar vs lettr
x-bar
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z



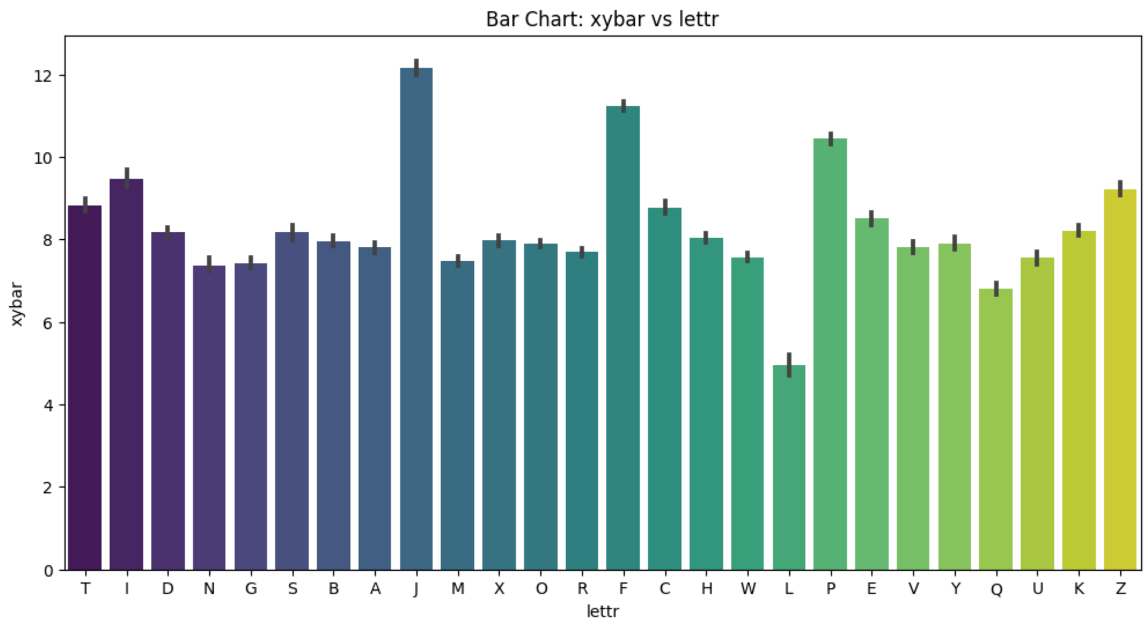
Bar Chart: y-bar vs lettr
y-bar
0
2
4
6
8
10
12
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr

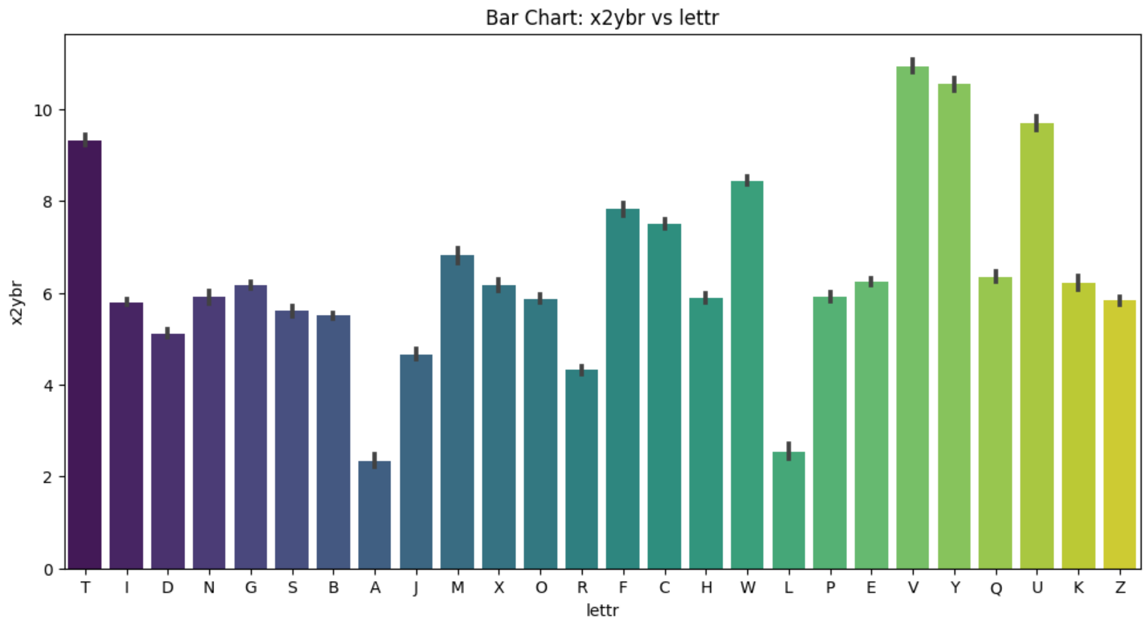


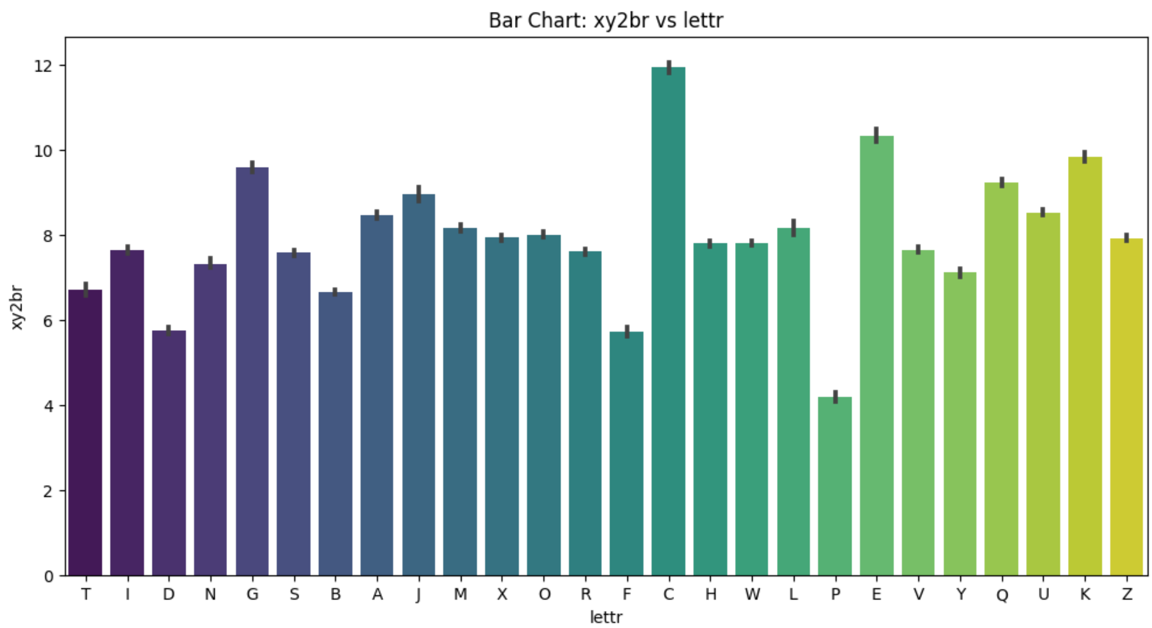
Bar Chart: x2bar vs lettr
x2bar
0
1
2
3
4
5
6
7
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr

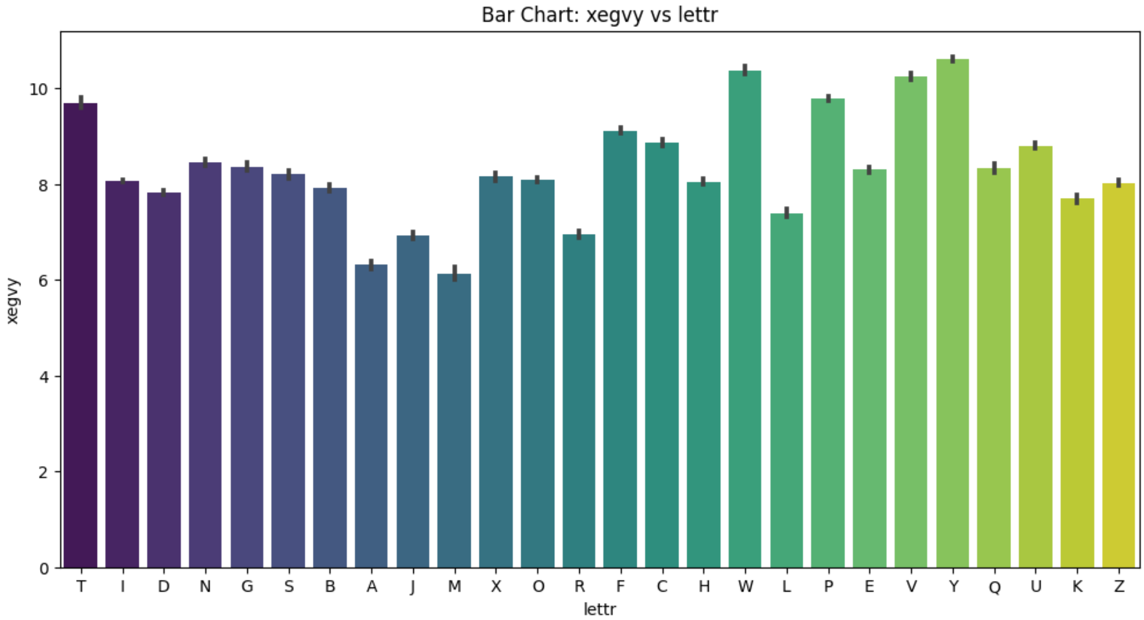


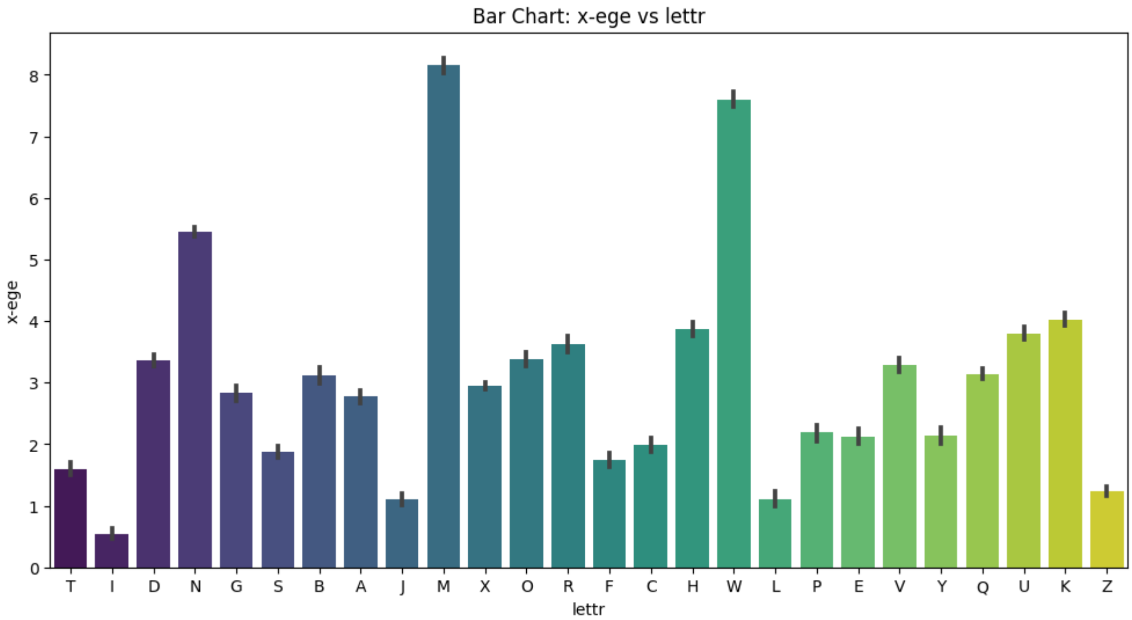
Bar Chart: y2bar vs lettr
y2bar
0
2
4
6
8
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr

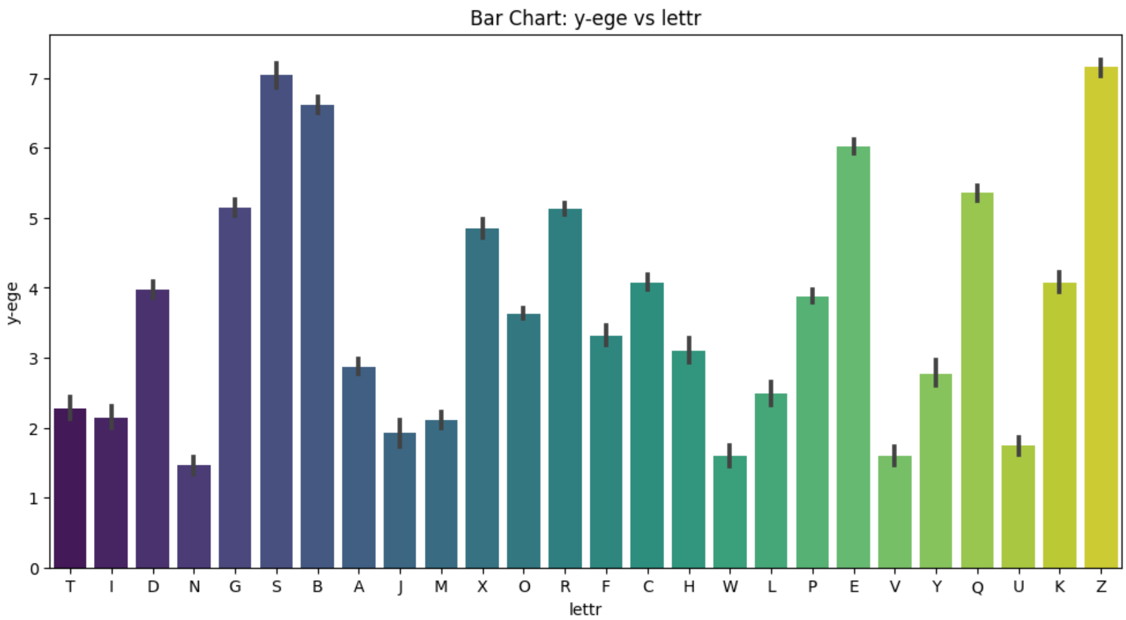

Bar Chart: xybar vs lettr
xybar
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
0 2 4 6 8 10 12

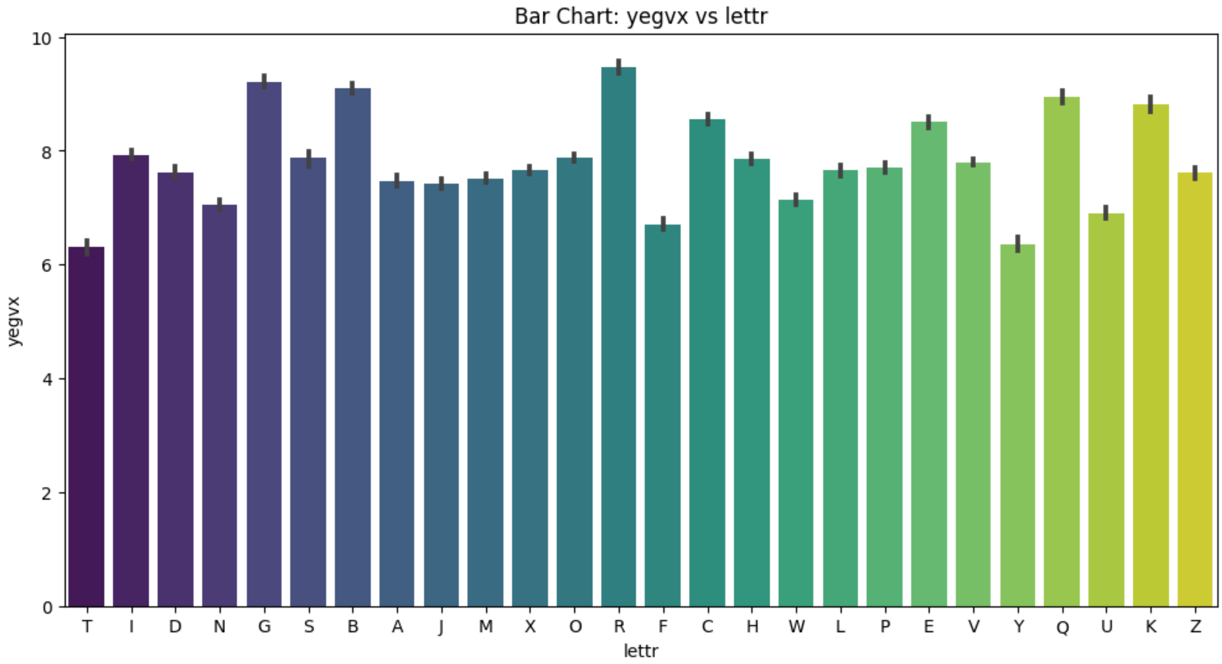

Bar Chart: x2ybr vs lettr
x2ybr
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
0 2 4 6 8 10


Bar Chart: xy2br vs lettr
xy2br
lettr
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
0 2 4 6 8 10 12


Bar Chart: xegvy vs lettr
xegvy
0
2
4
6
8
10
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr


Bar Chart: x-ege vs lettr
x-ege
0
1
2
3
4
5
6
7
8
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr


Bar Chart: y-ege vs lettr
y-ege
0
1
2
3
4
5
6
7
T I D N G S B A J M X O R F C H W L P E V Y Q U K Z
lettr



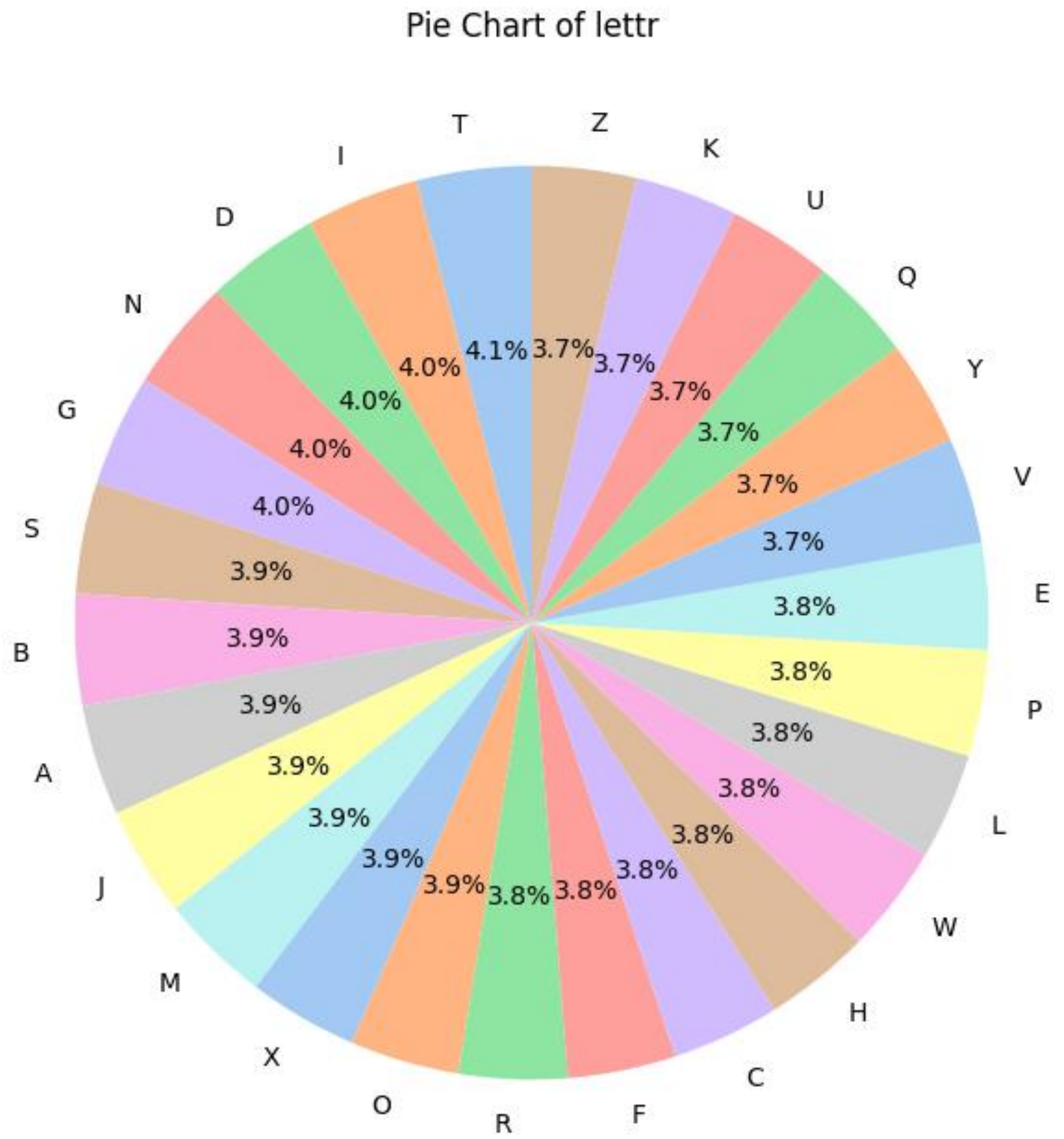
1. **Feature Importance:**
    - The height of each bar represents the frequency or count of observations for different categories or values of a feature. A higher bar indicates that value or category is more prevalent in the dataset, suggesting its importance.
2. **Class Imbalance:**
    - By observing the heights of bars for different classes in the target variable, you can assess whether the dataset is balanced or if there's an imbalance between different classes. Class imbalance can impact model training and evaluation.
3. **Predictive Power:**
    - Features with distinct differences in the distribution of target classes may have strong predictive power. If certain values of a feature are associated with a significantly higher or lower likelihood of a specific target class, it suggests that the feature is informative for predicting the target.
4. **Identifying Patterns:**
    - Bar charts help you identify patterns or trends in the distribution of target classes across different values of a feature. For example, you might observe that for a certain range of values in a feature, one target class is more prevalent than others.
5. **Potential Categorical Encoding:**
    - If the feature is categorical, you can assess whether there's a monotonic relationship between categories and the target variable. This might guide decisions on encoding strategies, such as one-hot encoding or label encoding.

6. **Outliers or Anomalies:**
    - Unusual patterns, spikes, or irregularities in the bar chart could indicate potential outliers or anomalies in the data. Investigating such patterns can be crucial for data quality assessment.
7. **Feature Engineering Insights:**
    - Understanding the relationship between features and the target variable can inform feature engineering decisions. For instance, you might consider creating new features based on observed patterns or transformations to enhance model performance.
8. **Model Interpretation:**
    - When interpreting machine learning models, insights gained from the bar chart can provide context for understanding how individual features contribute to predictions. Some features may have more discriminatory power than others.

Observation: -

From chart we can see that feature high and y -box don't contribute much to classify the target lettr.

Pie Chart: -



Gives same information as the initial bar plot but in pie format.

Preprocessing

Preprocessing refers to the preparation and transformation of raw data before it is fed into a machine learning algorithm. It is a crucial step in the data analysis and modeling pipeline, aimed at improving the quality of the data and making it suitable for analysis or training a model.

Feature vector encoding for models:

We will be encoding the lettr which has character data to classes in integers.

Code: -

```
[23] from sklearn.preprocessing import LabelEncoder
     # Display original target variable
     print("Original 'lettr' values:")
     print(y['lettr'])

     # Encode the target variable using LabelEncoder
     le = LabelEncoder()
     y_encoded = le.fit_transform(y['lettr'])

     # Display the encoded target variable
     print("\nEncoded 'lettr' values:")

     y_encoded = pd.DataFrame({ 'lettr' : y_encoded} )

     print(y_encoded['lettr'])

     # Inverse transform to see the original labels from the encoded values
     decoded_values = le.inverse_transform(y_encoded['lettr'])
     print("\nDecoded values:")
     print(decoded_values)
```

Output: -

```
Original 'lettr' values:
0        T
1        I
2        D
3        N
4        G
        ..
19995    D
19996    C
19997    T
19998    S
19999    A
Name: lettr, Length: 20000, dtype: object

Encoded 'lettr' values:
0        19
1         8
2         3
3        13
4         6
         ..
19995     3
19996     2
19997    19
19998    18
19999     0
Name: lettr, Length: 20000, dtype: int64

Decoded values:
['T' 'I' 'D' ... 'T' 'S' 'A']
```

<u>Dataset Split</u>: -

For evaluating dataset, we will be using holdout and split the data into two sets. One for training (80 %) and other for evaluation/validation/tests (20 %).

Code: -

```
[25] import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     from sklearn.preprocessing import LabelEncoder
     import seaborn as sns
     import matplotlib.pyplot as plt

     #encoding of data is needed for classification

     df = pd.concat([X,y_encoded],axis =1)

     y_encoded = df['lettr']
     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

Machine Learning

Models used: -

1. Random Forrest
2. SVM
3. KNN
4. ANN

1. Random Forrest:

A Random Forest is a machine learning method that creates many decision trees during training and combines their outputs for better accuracy. It introduces randomness by using random subsets of data and features, preventing overfitting. This ensemble approach makes Random Forests robust and effective for classification or regression tasks. They're widely used in diverse fields for their versatility and ability to handle complex datasets.

Training: -

```
# Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Prediction:

```
# Predictions
rf_predictions = rf_model.predict(X_test)
rf_accuracy =  accuracy_score(y_test, rf_predictions)

print(f'Sample Input: {X_test.iloc[0]} \n'
      f'Sample Output: {rf_predictions[0]} \n'
      f'Decoded Output: {le.inverse_transform([rf_predictions[0]])[0]} \n'
      f'True Output: {le.inverse_transform([y_test.iloc[0]])[0]}\n')
```

```
Sample Input: x-box    3
y-box    6
width    5
high     6
onpix    4
x-bar    6
y-bar    7
x2bar    3
y2bar    8
xybar    8
x2ybr    6
xy2br    9
x-ege    3
xegvy    7
y-ege    7
yegvx    6
Name: 10650, dtype: int64
Sample Output: 19
Decoded Output: T
True Output: T
```

Evaluation: -

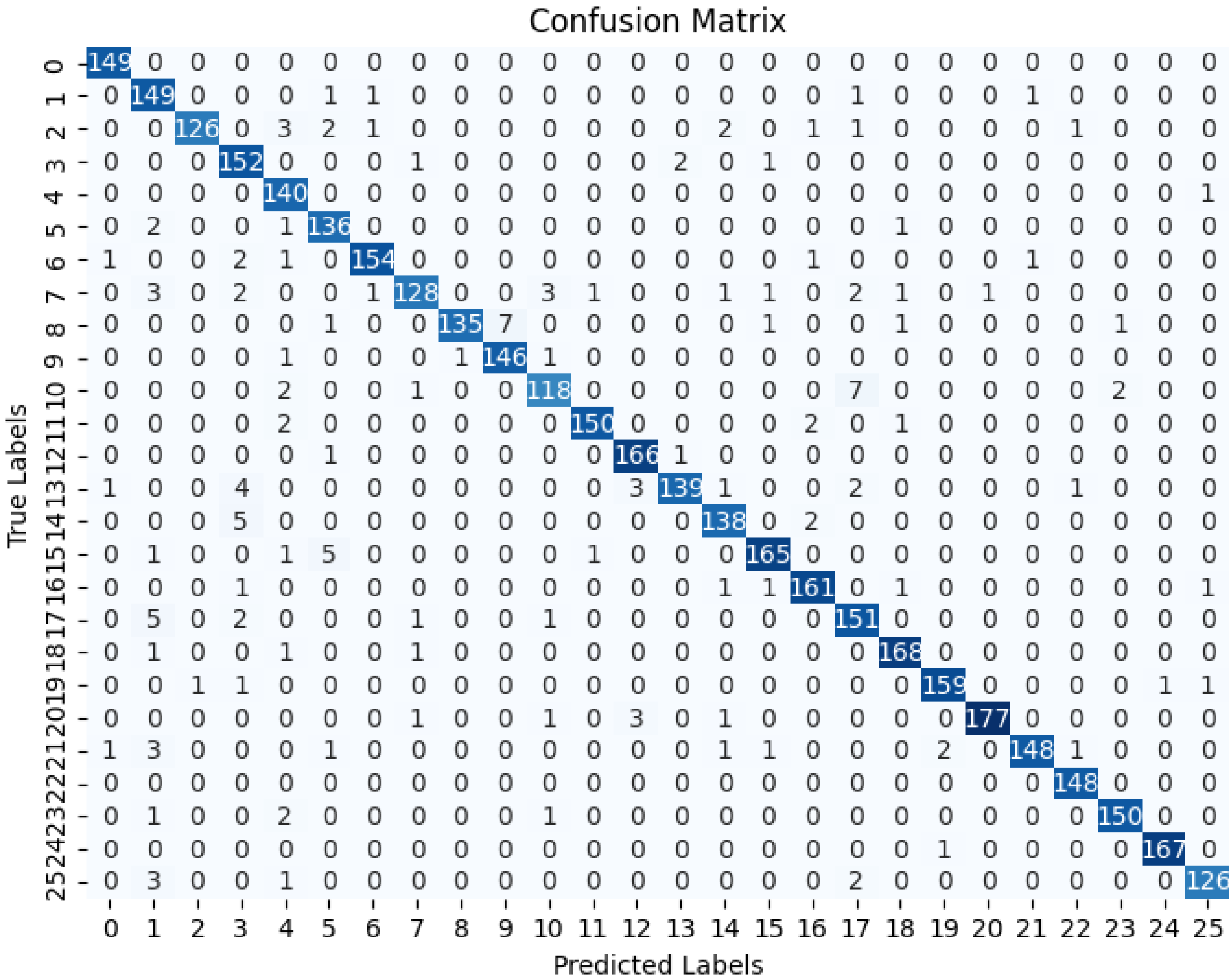
Code:

```
[35] # Evaluate Random Forest Model
     print("Random Forest Classifier:")
     print("Accuracy:", accuracy_score(y_test, rf_predictions))
     print("\nClassification Report:\n", classification_report(y_test, rf_predictions))
     cm = confusion_matrix(y_test, rf_predictions)

     # Create a Seaborn heatmap
     plt.figure(figsize=(8, 6))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.show()
```

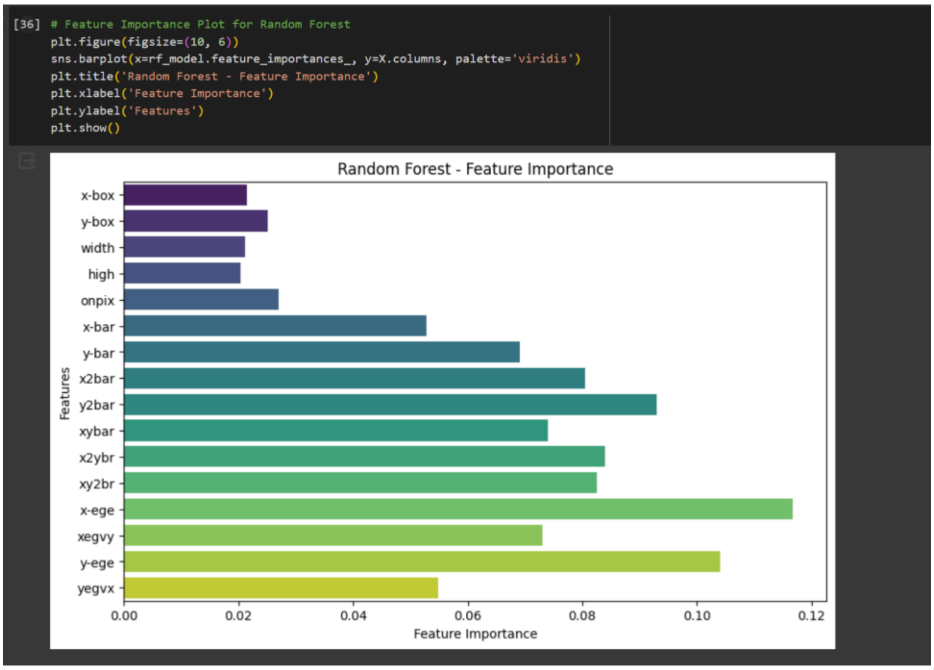
Output:

```
Random Forest Classifier:
Accuracy: 0.9615

Classification Report:
               precision    recall  f1-score   support

           0       0.98      1.00      0.99       149
           1       0.89      0.97      0.93       153
           2       0.99      0.92      0.95       137
           3       0.90      0.97      0.94       156
           4       0.90      0.99      0.95       141
           5       0.93      0.97      0.95       140
           6       0.98      0.96      0.97       160
           7       0.96      0.89      0.92       144
           8       0.99      0.92      0.96       146
           9       0.95      0.98      0.97       149
          10       0.94      0.91      0.93       130
          11       0.99      0.97      0.98       155
          12       0.97      0.99      0.98       168
          13       0.98      0.92      0.95       151
          14       0.95      0.95      0.95       145
          15       0.97      0.95      0.96       173
          16       0.96      0.97      0.97       166
          17       0.91      0.94      0.93       160
          18       0.97      0.98      0.98       171
          19       0.98      0.98      0.98       163
          20       0.99      0.97      0.98       183
          21       0.99      0.94      0.96       158
          22       0.98      1.00      0.99       148
          23       0.98      0.97      0.98       154
          24       0.99      0.99      0.99       168
          25       0.98      0.95      0.97       132

    accuracy                           0.96      4000
   macro avg       0.96      0.96      0.96      4000
weighted avg       0.96      0.96      0.96      4000
```

Confusion Matrix::-



Important features for Random Forest: -



2. SVM

Support Vector Machine (SVM) is a machine learning algorithm used for classification and regression tasks. It works by finding a hyperplane that best separates data into different classes. SVM aims to maximize the margin between classes, enhancing generalization. SVM is effective in high-dimensional spaces and is widely applied in various domains for its robust performance.

Training and Prediction: -

```
# Support Vector Machine (SVM) Classifier
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)

# Predictions
svm_predictions = svm_model.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_predictions)

print(f'Sample Input: {X_test.iloc[0]} \n'
      f'Sample Output: {svm_predictions[0]} \n'
      f'Decoded Output: {le.inverse_transform([svm_predictions[0]])[0]} \n'
      f'True Output: {le.inverse_transform([y_test.iloc[0]])[0]}\n')
```

Sample Input: x-box   3

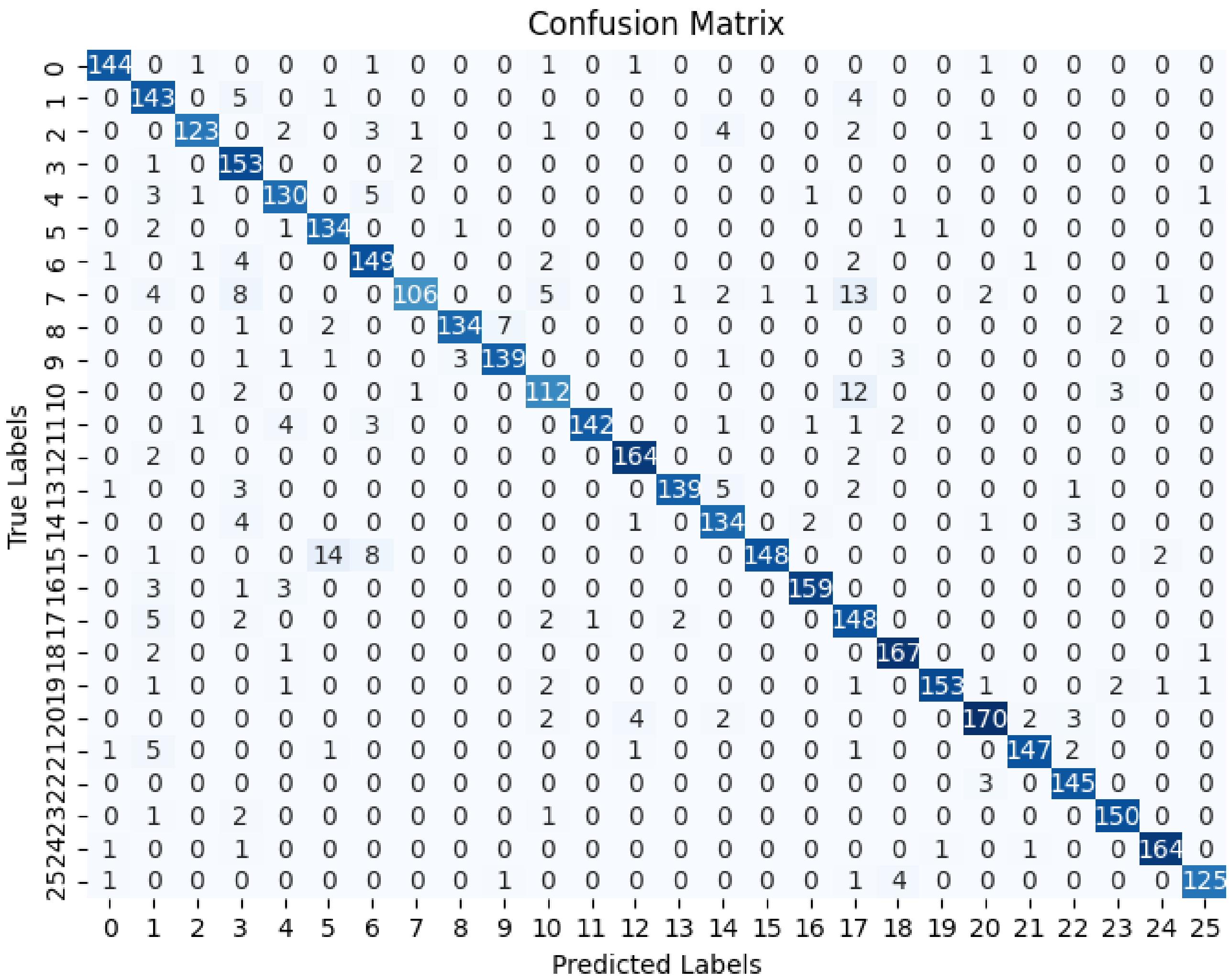
```
Sample Input: x-box    3
y-box    6
width    5
high     6
onpix    4
x-bar    6
y-bar    7
x2bar    3
y2bar    8
xybar    8
x2ybr    6
xy2br    9
x-ege    3
xegvy    7
y-ege    7
yegvx    6
Name: 10650, dtype: int64
Sample Output: 23
Decoded Output: X
True Output: T
```

Evaluation: -

```
[39] # Evaluate SVM Model
     print("\nSupport Vector Machine (SVM) Classifier:")
     print("Accuracy:", accuracy_score(y_test, svm_predictions))
     print("\nClassification Report:\n", classification_report(y_test, svm_predictions))

     cm = confusion_matrix(y_test, svm_predictions)

     # Create a Seaborn heatmap
     plt.figure(figsize=(8, 6))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.show()
```

```
Support Vector Machine (SVM) Classifier:
Accuracy: 0.9305

Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97       149
           1       0.83      0.93      0.88       153
           2       0.97      0.90      0.93       137
           3       0.82      0.98      0.89       156
           4       0.91      0.92      0.92       141
           5       0.88      0.96      0.91       140
           6       0.88      0.93      0.91       160
           7       0.96      0.74      0.83       144
           8       0.97      0.92      0.94       146
           9       0.95      0.93      0.94       149
          10       0.88      0.86      0.87       130
          11       0.99      0.92      0.95       155
          12       0.96      0.98      0.97       168
          13       0.98      0.92      0.95       151
          14       0.90      0.92      0.91       145
          15       0.99      0.86      0.92       173
          16       0.97      0.96      0.96       166
          17       0.78      0.93      0.85       160
          18       0.94      0.98      0.96       171
          19       0.99      0.94      0.96       163
          20       0.95      0.93      0.94       183
          21       0.97      0.93      0.95       158
          22       0.94      0.98      0.96       148
          23       0.96      0.97      0.96       154
          24       0.98      0.98      0.98       168
          25       0.98      0.95      0.96       132

    accuracy                           0.93      4000
   macro avg       0.93      0.93      0.93      4000
weighted avg       0.93      0.93      0.93      4000
```

Confusion matrix: -



3. KNN

K-Nearest Neighbours (KNN) is a simple yet powerful machine learning algorithm used for classification and regression. It makes predictions by considering the majority class or average of the k-nearest data points to a given input

Training and Prediction:

```
[42] import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)


     # Create a KNN classifier (you can adjust the 'n_neighbors' parameter)
     knn_classifier = KNeighborsClassifier(n_neighbors=5)

     # Train the model
     knn_classifier.fit(X_train_scaled, y_train)

     ▾ KNeighborsClassifier
     KNeighborsClassifier()
```

```
# Make predictions on the test set
knn_predictions = knn_classifier.predict(X_test_scaled)
# Print a sample prediction along with its details
print(f'Sample Input: {X_test.iloc[0]} \n'
      f'Sample Output: {knn_predictions[0]} \n'
      f'Decoded Output: {le.inverse_transform([knn_predictions[0]])[0]} \n'
      f'True Output: {y_test.iloc[0]}\n')
```

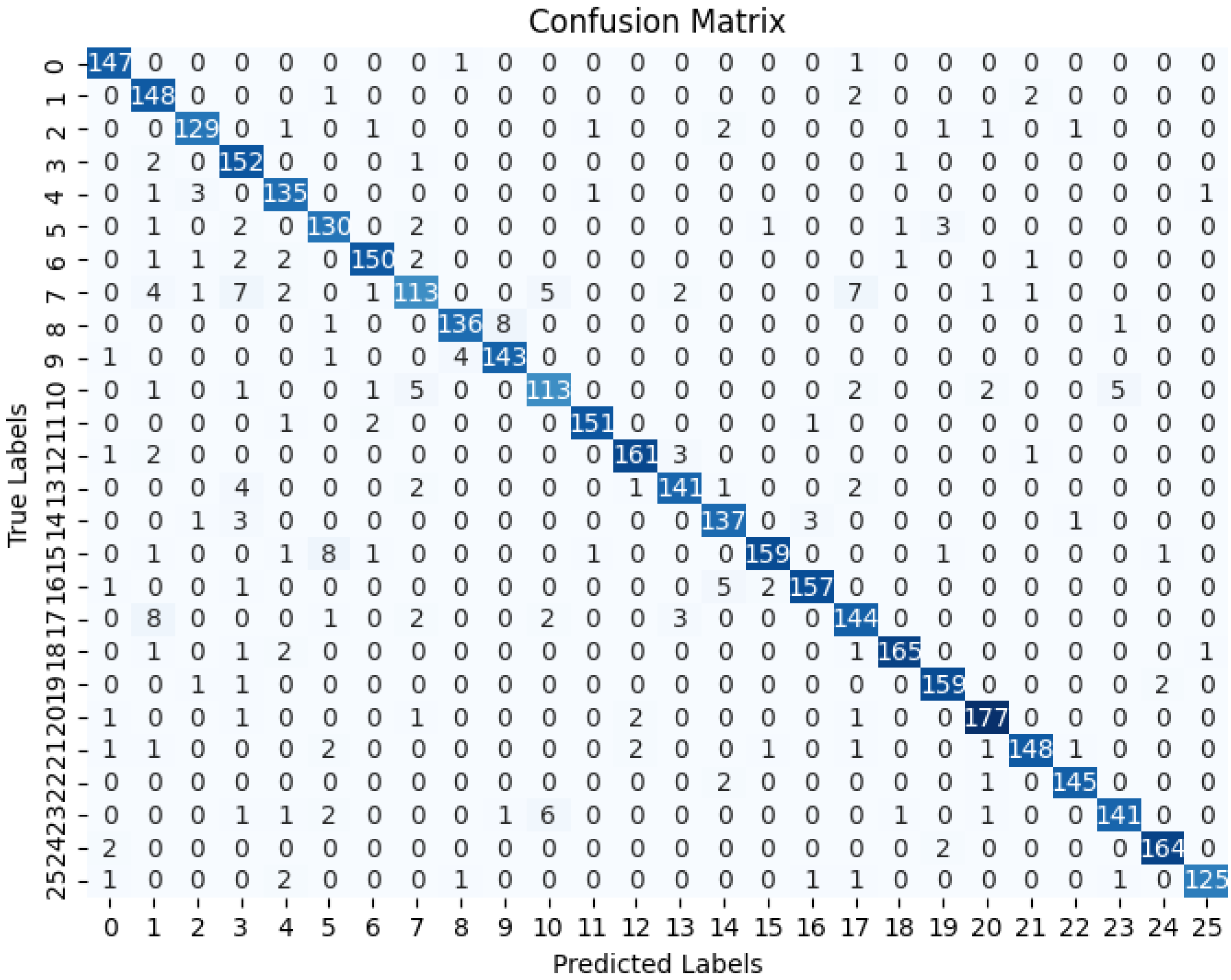
```
Sample Input: x-box    3
y-box     6
width     5
high      6
onpix     4
x-bar     6
y-bar     7
x2bar     3
y2bar     8
xybar     8
x2ybr     6
xy2br     9
x-ege     3
xegvy     7
y-ege     7
yegvx     6
Name: 10650, dtype: int64
Sample Output: 19
Decoded Output: T
True Output: 19
```

Evaluation: -

```
[46] # Evaluate the model
     accuracy = accuracy_score(y_test, knn_predictions)
     classification_report_str = classification_report(y_test, knn_predictions)

     # Print the evaluation results
     print(f'Accuracy: {accuracy:.4f}\n')
     print('\nClassification Report:\n', classification_report_str)
     # Create a confusion matrix
     conf_matrix = confusion_matrix(y_test, knn_predictions)

     # Plot the confusion matrix as a heatmap
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.show()
```

```
Accuracy: 0.9425


Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97       149
           1       0.87      0.97      0.91       153
           2       0.95      0.94      0.95       137
           3       0.86      0.97      0.92       156
           4       0.92      0.96      0.94       141
           5       0.89      0.93      0.91       140
           6       0.96      0.94      0.95       160
           7       0.88      0.78      0.83       144
           8       0.96      0.93      0.94       146
           9       0.94      0.96      0.95       149
          10       0.90      0.87      0.88       130
          11       0.98      0.97      0.98       155
          12       0.97      0.96      0.96       168
          13       0.95      0.93      0.94       151
          14       0.93      0.94      0.94       145
          15       0.98      0.92      0.95       173
          16       0.97      0.95      0.96       166
          17       0.89      0.90      0.89       160
          18       0.98      0.96      0.97       171
          19       0.96      0.98      0.97       163
          20       0.96      0.97      0.96       183
          21       0.97      0.94      0.95       158
          22       0.98      0.98      0.98       148
          23       0.95      0.92      0.93       154
          24       0.98      0.98      0.98       168
          25       0.98      0.95      0.97       132

    accuracy                           0.94      4000
   macro avg       0.94      0.94      0.94      4000
weighted avg       0.94      0.94      0.94      4000
```

Confusion matrix:



4. ANN

Artificial Neural Network (ANN) is a complex machine learning model inspired by the human brain's neural structure. It consists of layers of interconnected nodes, or neurons, and is used for various tasks such as classification, regression, and pattern recognition. ANN learns from data by adjusting weights between neurons during training.

Training:

```
[58] import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     import tensorflow as tf



     # Build a Neural Network Model
     model = tf.keras.Sequential([
         tf.keras.layers.Dense(64, input_dim=X_train.shape[1], activation='relu'),
         tf.keras.layers.Dense(26, activation='softmax')  # Output layer with 26 units for each letter
     ])

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

     # Train the model
     model.fit(X_train, y_train, epochs=50, batch_size=16, validation_split=0.1, verbose=1)
```

Predictions: -

```
# Print Model Comparison
print("Model Comparison:")
print("Random Forest Accuracy:", rf_accuracy)
print("SVM Accuracy:", svm_accuracy)
print("KNN Accuracy:", knn_accuracy)
print("Neural Network Accuracy:", nn_accuracy)
```

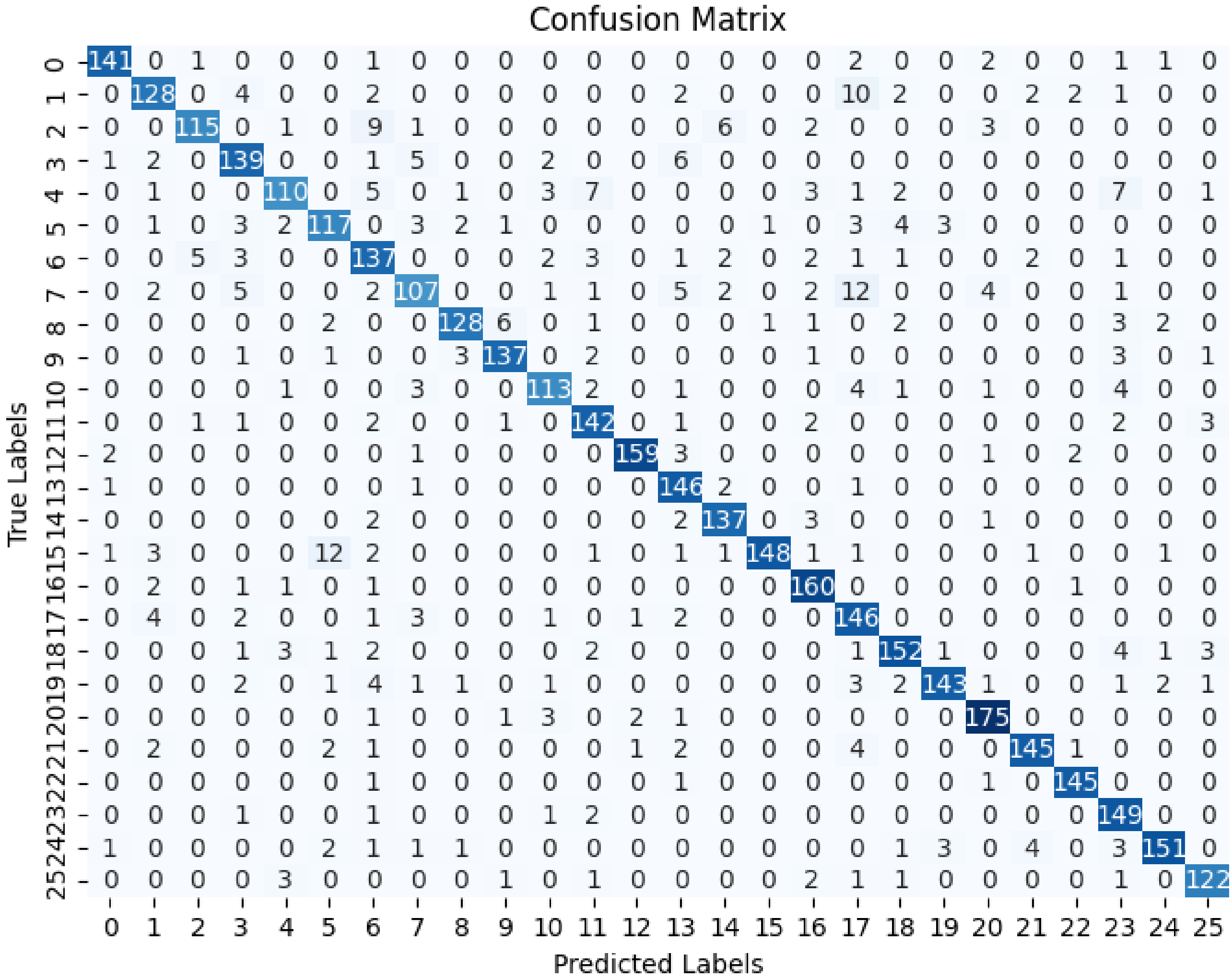
```
Model Comparison:
Random Forest Accuracy: 0.9615
SVM Accuracy: 0.9305
KNN Accuracy: 0.9425
Neural Network Accuracy: 0.8980000019073486
```

```
# Plotting
models = ['Random Forest', 'SVM', 'Neural Network','KNN']
accuracies = [rf_accuracy, svm_accuracy, nn_accuracy, knn_accuracy]

plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=accuracies, palette='viridis')
plt.title('Model Comparison - Accuracy')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()
```

Evaluation: -

```
Neural Network Classifier:
Accuracy: 0.8980000019073486
125/125 [==============================] - 0s 2ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       149
           1       0.88      0.84      0.86       153
           2       0.94      0.84      0.89       137
           3       0.85      0.89      0.87       156
           4       0.91      0.78      0.84       141
           5       0.85      0.84      0.84       140
           6       0.78      0.86      0.82       160
           7       0.85      0.74      0.79       144
           8       0.94      0.88      0.91       146
           9       0.93      0.92      0.93       149
          10       0.89      0.87      0.88       130
          11       0.87      0.92      0.89       155
          12       0.98      0.95      0.96       168
          13       0.84      0.97      0.90       151
          14       0.91      0.94      0.93       145
          15       0.99      0.86      0.92       173
          16       0.89      0.96      0.93       166
          17       0.77      0.91      0.83       160
          18       0.90      0.89      0.90       171
          19       0.95      0.88      0.91       163
          20       0.93      0.96      0.94       183
          21       0.94      0.92      0.93       158
          22       0.96      0.98      0.97       148
          23       0.82      0.97      0.89       154
          24       0.96      0.90      0.93       168
          25       0.93      0.92      0.93       132

    accuracy                           0.90      4000
   macro avg       0.90      0.90      0.90      4000
weighted avg       0.90      0.90      0.90      4000
```

Confusion matrix:-



Comparison of Models