Professors :

Simona Campisi

Gennaro Trtarisco

Made by:

Mujibullah Rabin

Università degli Studi di Messina

TABLE OF CONTENTS

Emotional State Analysis in Consumer Behavior

INTRODUCTION

This project delves into the dataset "Study 4," focusing on the impact of anxiety on consumer product evaluations among 83 healthy subjects. Through data mining and analytics, we aim to predict emotional states based on psychophysiological responses. The script employs feature selection, PCA, and classification, showcasing a systematic approach from data loading to model evaluation. This analysis aims to be a predictive tool, offering insights into the relationship between anxiety and consumer choices. The constructed model holds implications for understanding and predicting emotional responses in consumer behavior.

CODE EXPLANATION

1. Import necessary libraries

This segment involves importing essential libraries for data analysis and machine learning. These include pandas for data manipulation, matplotlib and seaborn for data visualization, and scikit-learn for machine learning tasks such as model building and evaluation.

```
[1] import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.preprocessing import StandardScaler
    from sklearn.feature_selection import SelectKBest, f_classif
    from sklearn.decomposition import PCA
    from sklearn.cluster import KMeans
    from sklearn.svm import SVC
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

2. Load the dataset

Here, the code reads the dataset from a CSV file into a Pandas DataFrame.

```
[2] file_path = "Dataset_Study4.csv"
    dataset = pd.read_csv(file_path)
```

3. Display first few rows of the dataset

The dataset.head() function is used to display the first few rows of the loaded dataset. It's a quick way to get an overview of the structure and content of the data.

```
[3] dataset.head()
```

	HRV_MeanNN	HRV_SDNN	HRV_RMSSD	HRV_Prc20NN	HRV_Prc80NN	HRV_pNN50	HRV_HTI	HRV_VLF	HRV_LF	HRV_HF	HRV_TP	HRV_LFHF	HRV_SD1	HRV_
0	833.314763	106.424590	88.217436	733.8	932.0	50.277778	24.000000	62.654127	713.021695	4228.126423	5161.003600	0.168638	62.465836	137.040
1	766.966581	92.330308	76.581684	682.8	856.0	49.487179	18.571429	73.007851	1245.114713	3010.533164	4423.639565	0.413586	54.220273	118.651
2	896.408408	61.685886	51.577499	851.2	948.0	36.526946	14.521739	36.950885	1201.268110	1179.137264	2458.917474	1.018769	36.525411	79.282
3	833.256267	71.637093	46.592095	789.8	889.0	26.666667	12.857143	156.099084	1241.711995	1299.760891	2731.734549	0.955339	32.991403	95.896
4	782.793194	45.483921	21.992054	753.4	820.0	1.566580	9.820513	28.439521	396.440159	156.126782	591.933499	2.539219	15.571013	62.476

4.Get the shape of the dataset

When you run dataset.shape, it will return a tuple where the first element is the number of rows, and the second element is the number of columns.

```
0s
dataset.shape

(166, 20)
```

5.Check for duplicated rows

The dataset.duplicated().sum() code is checking for and counting the number of duplicated rows in your dataset.

```
0s
dataset.duplicated().sum()

0
```

6.Check for missing values

This part checks for missing values in the dataset using isnull().sum() and prints the count of missing values for each column.

```
[3] print(dataset.isnull().sum())

HRV_MeanNN       0
HRV_SDNN         0
HRV_RMSSD        0
HRV_Prc20NN      0
HRV_Prc80NN      0
HRV_pNN50        0
HRV_HTI          0
HRV_VLF          0
HRV_LF           0
HRV_HF           0
HRV_TP           0
HRV_LFHF         0
HRV_SD1          0
HRV_SD2          0
HRV_SD1SD2       0
HRV_DFA_alpha1   0
HRV_DFA_alpha2   0
HRV_ApEn         0
HRV_SampEn       0
Label            0
dtype: int64
```

7.Explore the dataset

dataset.describe() provides a concise summary of the dataset, including the number of non-null values and data types for each column and total memory usage of the DataFrame. It helps you understand the distribution and variability of your numerical features.

dataset.info() provides summary statistics for numerical columns in your dataset, such as mean, standard deviation, minimum, 25th percentile, 50th percentile(median), 75th percentile, and maximum values. It helps you understand the distribution and variability of your numerical features.

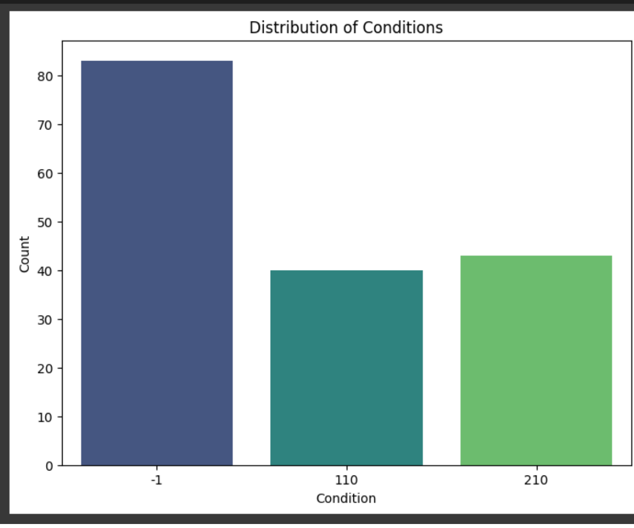
```python
print(dataset.info())
print(dataset.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 166 entries, 0 to 165
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   HRV_MeanNN      166 non-null    float64
 1   HRV_SDNN        166 non-null    float64
 2   HRV_RMSSD       166 non-null    float64
 3   HRV_Prc20NN     166 non-null    float64
 4   HRV_Prc80NN     166 non-null    float64
 5   HRV_pNN50       166 non-null    float64
 6   HRV_HTI         166 non-null    float64
 7   HRV_VLF         166 non-null    float64
 8   HRV_LF          166 non-null    float64
 9   HRV_HF          166 non-null    float64
 10  HRV_TP          166 non-null    float64
 11  HRV_LFHF        166 non-null    float64
 12  HRV_SD1         166 non-null    float64
 13  HRV_SD2         166 non-null    float64
 14  HRV_SD1SD2      166 non-null    float64
 15  HRV_DFA_alpha1  166 non-null    float64
 16  HRV_DFA_alpha2  166 non-null    float64
 17  HRV_ApEn        166 non-null    float64
 18  HRV_SampEn      166 non-null    float64
 19  Label           166 non-null    int64
dtypes: float64(19), int64(1)
memory usage: 26.1 KB
None
         HRV_MeanNN    HRV_SDNN   HRV_RMSSD  HRV_Prc20NN  HRV_Prc80NN  \
count    166.000000  166.000000  166.000000   166.000000   166.000000
mean     753.929001   63.892229   44.198295   703.165060   805.502410
std      103.047627   31.535479   36.852678    92.960201   124.015723
min      523.712785   16.910649    5.098153   501.000000   536.000000
25%      683.971330   44.235922   25.088206   639.600000   717.500000
50%      746.050891   55.993083   34.934981   694.200000   794.800000
75%      823.638007   75.135998   47.316602   756.500000   874.950000
max     1084.850909  213.710837  269.996901   999.000000  1229.000000

        HRV_pNN50     HRV_HTI     HRV_VLF        HRV_LF        HRV_HF  \
count  166.000000  166.000000  166.000000    166.000000    166.000000
mean    15.846306   13.684865   73.215664   1456.757545   1378.864096
std     16.965290    4.757301   70.899578   2278.424999   4110.417245
min      0.000000    4.539683    9.244941     65.526316     19.238621
25%      3.489703   10.096970   27.162383    564.384887    262.664392
50%     10.215322   13.030835   49.034185    966.579897    511.237107
75%     21.179379   16.062500   89.702415   1494.851470    946.561098
max     78.816199   29.000000  501.321504  22554.374344  37130.282959

             HRV_TP    HRV_LFHF     HRV_SD1     HRV_SD2  HRV_SD1SD2  \
count    166.000000  166.000000  166.000000  166.000000  166.000000
mean    2969.829008    2.246604   31.294365   84.042702    0.350801
std     5467.775565    1.608922   26.098812   37.806632    0.127198
min       98.023392    0.065682    3.608012   23.647461    0.152575
25%      928.849917    1.167909   17.759990   59.166917    0.274794
50%     1648.780152    1.886142   24.732381   74.570948    0.337210
75%     2662.233813    2.972089   33.502981   97.500962    0.395311
max    40087.331053    9.748947  191.214715  244.158020    0.867390

       HRV_DFA_alpha1  HRV_DFA_alpha2    HRV_ApEn  HRV_SampEn       Label
count      166.000000      166.000000  166.000000  166.000000  166.000000
mean         1.234471        0.770297    1.074733    1.311300   80.403614
std          0.254357        0.190655    0.126677    0.291660   89.009225
min          0.334951        0.328423    0.539257    0.460738   -1.000000
25%          1.093744        0.642667    1.021211    1.138868   -1.000000
50%          1.264021        0.762627    1.097244    1.332827   54.500000
75%          1.412508        0.903358    1.152502    1.521149  210.000000
max          1.739736        1.294787    1.322549    1.959751  210.000000
```

8. Visualize the distribution of the 'Label' column

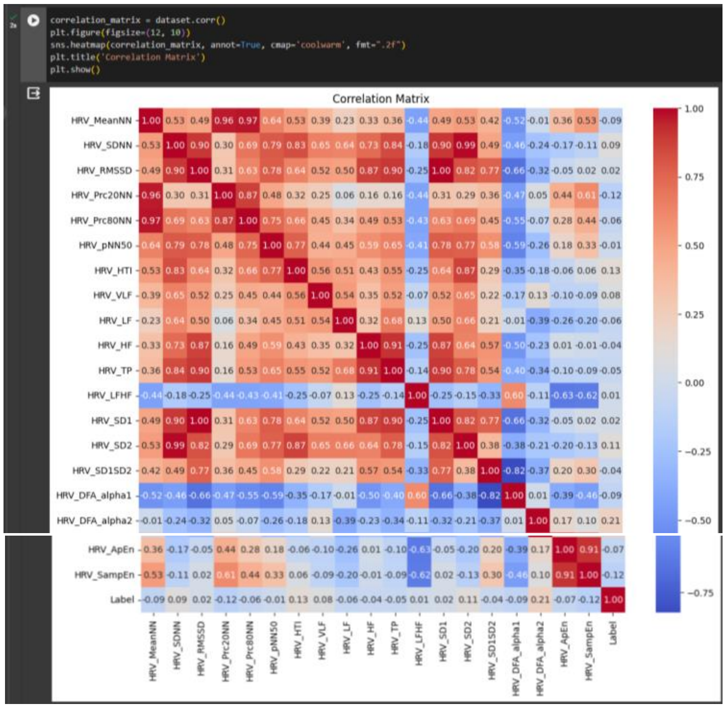
This code snippet is using the Seaborn library to create a histogram plot to visualize the distribution of emotional states in your dataset.

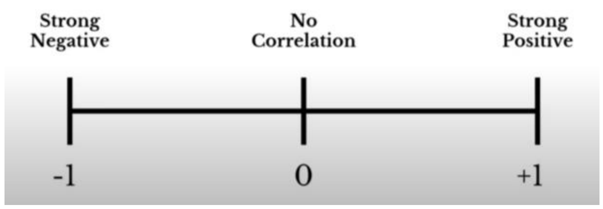
```
plt.figure(figsize=(8, 6))
sns.countplot(x='Label', data=dataset, palette='viridis')
plt.title('Distribution of Conditions')
plt.xlabel('Condition')
plt.ylabel('Count')
plt.show()
```



9. Explore correlations between features

This section generates a heatmap to visualize the correlation matrix of features in the dataset.

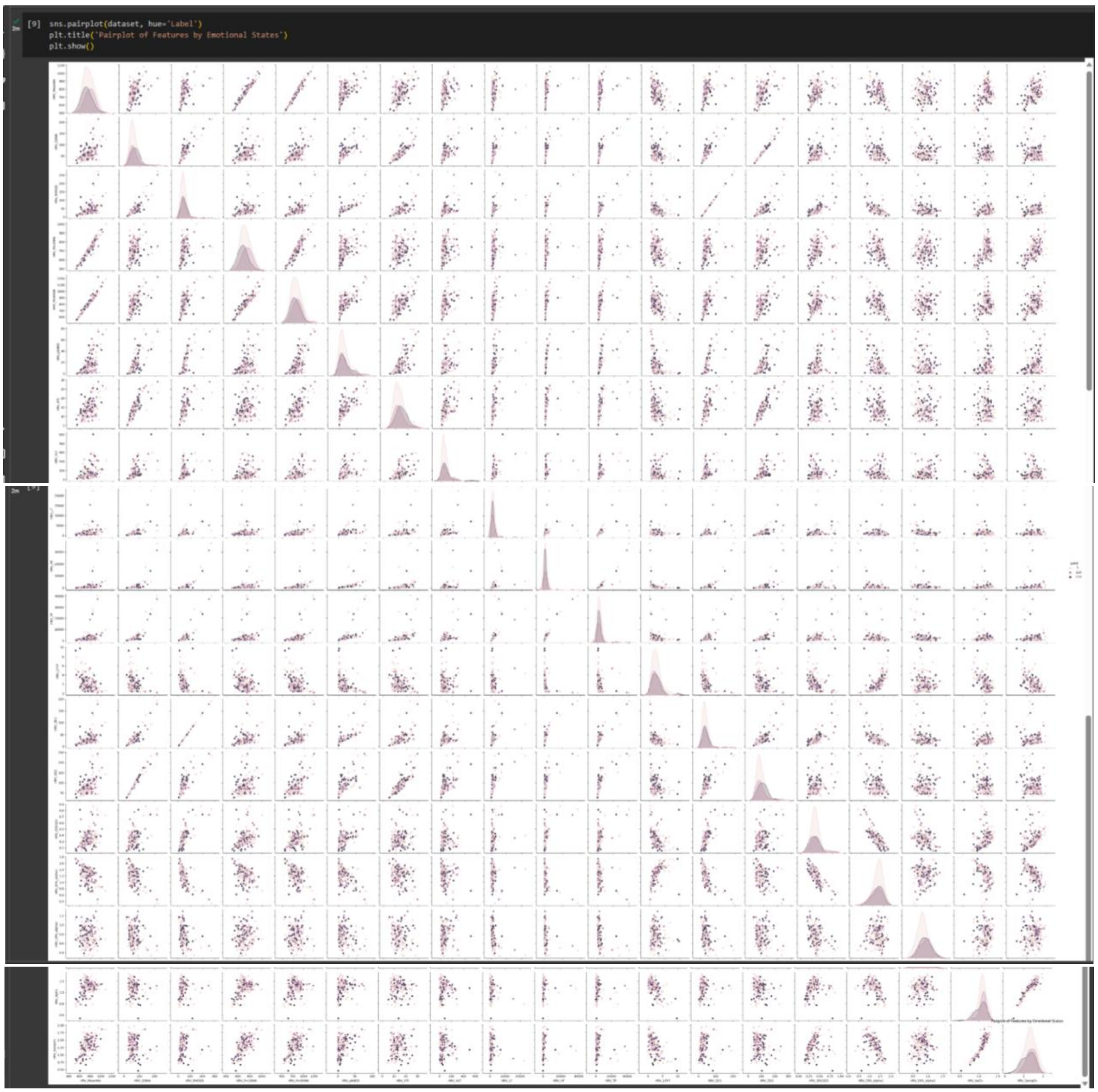




Positive correlations suggest that the variables tend to move together, while negative correlations suggest an inverse relationship between the variables.

10. Visualize relationships between features and emotional states

Here, a pair plot is generated using seaborn to visualize relationships between features in the dataset, with different emotional states distinguished by color.



11. Separate features and labels

This part separates features (X) and labels (y) from the dataset. The target variable is assumed to be named 'Label.'

```
[6] X = dataset.drop('Label', axis=1)
    y = dataset['Label']
```

- **X: a DataFrame containing only the feature columns (all columns except 'Label').**
- **y: a Series containing the labels (the 'Label' column).**

This separation is essential for training machine learning models, as the features (X) are used as inputs to the model, while the labels (y) are the target outputs that the model will learn to predict.

12. Standardization

The StandardScaler in scikit-learn is used for standardizing features by removing the mean and scaling them to unit variance.

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

13. Feature selection

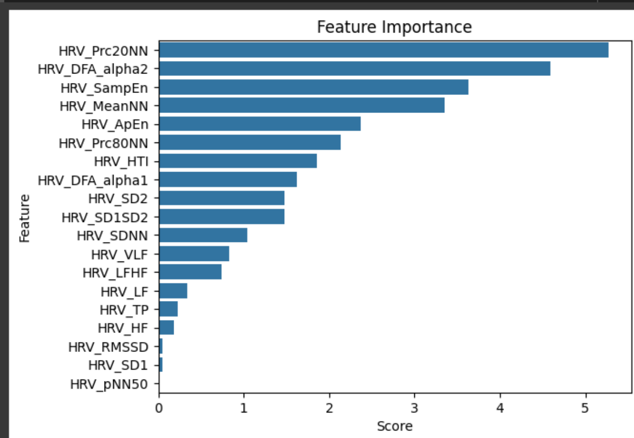
select the top 4 features based on their importance scores obtained from the ANOVA F-value test (f_classif). These selected features are considered the most informative for predicting the target variable (emotional states) in the classification task.
and returns a new array X_selected containing only these selected features.

```
selector = SelectKBest(f_classif, k=4)
X_selected = selector.fit_transform(x_scaled, y)
```

14. Visualize feature importance

This section creates a bar plot to visualize the importance scores of features obtained from the ANOVA F-statistic.

```python
feature_scores = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
feature_scores = feature_scores.sort_values(by='Score', ascending=False)
sns.barplot(x='Score', y='Feature', data=feature_scores)
plt.title('Feature Importance')
plt.show()
```



15. Standardize features

This part standardizes the selected features using StandardScaler from scikit-learn.

```python
[10] scaler = StandardScaler()
     X_standardized = scaler.fit_transform(X_selected)
```

16. Visualize the selected features

The code segment retrieves the names of the selected features after applying the SelectKBest feature selection method.

```
selected_features = X.columns[selector.get_support()]
print(selected_features)
```

```
Index(['HRV_MeanNN', 'HRV_Prc20NN', 'HRV_DFA_alpha2', 'HRV_SampEn'], dtype='object')
```

17. Apply PCA to reduce dimensionality.

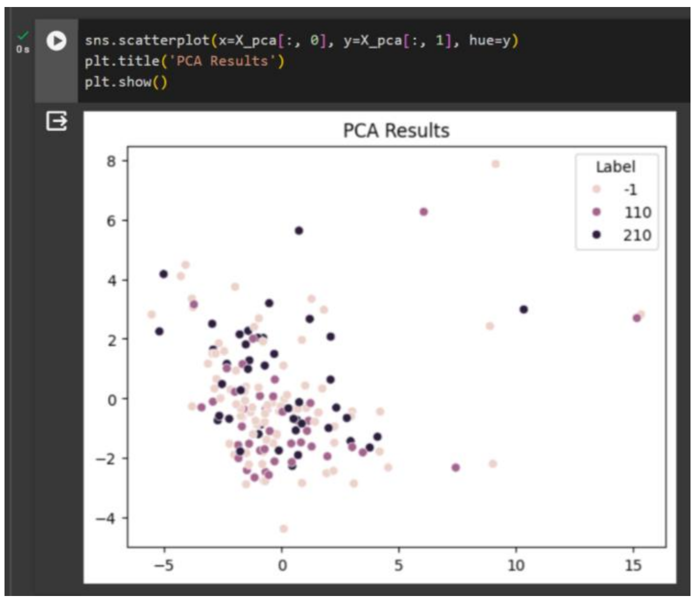
This line of code performs Principal Component Analysis (PCA) on the standardized data X_scaled to reduce its dimensionality to 3 principal components.

```
pca = PCA(n_components=3)
X_pca = pca.fit_transform(x_scaled)
```

X_pca will contain the transformed data with reduced dimensionality, where each sample is represented by three principal components.

18. Visualize PCA results

This segment generates a scatter plot to visualize the results of PCA, where each point is colored based on the corresponding emotional state.



19. Split the dataset into training and testing sets

The dataset is split into training and testing sets using train_test_split from scikit-learn.

```
[16] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

20. Standardize the features

Features in the training and testing sets are standardized using StandardScaler.

```
[19] scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

21. Build a Random Forest Classifier

A Random Forest Classifier is instantiated, trained on the scaled training data.

```
[20] rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
     rf_classifier.fit(X_train_scaled, y_train)

RandomForestClassifier
RandomForestClassifier(random_state=42)
```

- **X_train_scaled is the feature matrix of the training data, where the features have been scaled using StandardScaler.**
- **y_train is the target variable (labels) corresponding to the training data.**

22. Predict on the test set

This line of code predicts the labels for the test set using the trained Random Forest Classifier model.

```
[21] y_pred = rf_classifier.predict(X_test_scaled)
```

y_pred will contain the predicted labels for the test set based on the input features X_test_scaled

23. Evaluate the model

This segment prints the classification report and accuracy score to evaluate the performance of the Random Forest Classifier on the test set. The classification report provides metrics such as precision, recall, and F1-score for each class.
It compares the true labels (y_test) to the predicted labels (y_pred) and computes these metrics for each class.

```python
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

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

          -1       0.69      0.69      0.69        29
         110       0.45      0.56      0.50         9
         210       0.30      0.25      0.27        12

    accuracy                           0.56        50
   macro avg       0.48      0.50      0.49        50
weighted avg       0.55      0.56      0.56        50

Accuracy: 0.56
```

24. Confusion Matrix

This segment computes the confusion matrix using confusion_matrix from scikit-learn and visualizes it as a heatmap using seaborn. The confusion matrix helps assess the model's performance in terms of true positives, true negatives, false positives, and false negatives for each class.

25. Use SVM with cross-validation

Initialize a Support Vector Machine (SVM) classifier with a linear kernel

```
[28] svm_classifier = SVC(kernel='linear', random_state=42)
```

26. Perform cross-validation using X_standardized

cross_val_score is a function from scikit-learn that performs cross-validation by splitting the dataset into 'cv' (here, 5) consecutive folds. Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

```
cv_scores = cross_val_score(svm_classifier, x_scaled, y, cv=5)

print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy:", cv_scores.mean())
```

```
Cross-Validation Scores: [0.55882353 0.48484848 0.57575758 0.54545455 0.51515152]
Mean Accuracy: 0.5360071301247772
```

27. Fit SVM model on the entire dataset

```
svm_classifier.fit(x_scaled, y)
```

```
SVC
SVC(kernel='linear', random_state=42)
```

28. Split data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.1, random_state=42)
```

29. Predict using the trained model

```
[34] y_pred_svm = svm_classifier.predict(X_test)

     print("\nClassification Report (SVM):\n", classification_report(y_test, y_pred_svm))
     print("Accuracy (SVM):", accuracy_score(y_test, y_pred_svm))
```

```
Classification Report (SVM):
              precision    recall  f1-score   support

          -1       0.71      1.00      0.83        10
         110       0.50      0.25      0.33         4
         210       1.00      0.33      0.50         3

    accuracy                           0.71        17
   macro avg       0.74      0.53      0.56        17
weighted avg       0.71      0.71      0.66        17

Accuracy (SVM): 0.7058823529411765
```

30. Confusion Matrix

CONCLUSION

In summary, the Random Forest classifier achieved an accuracy of 56%, with relatively lower precision, recall, and F1-score values for the emotional states other than the baseline (-1).

The SVM classifier performed slightly better with an accuracy of 71%, showing higher precision, recall, and F1-score values across all classes. However, it still faces challenges in accurately predicting certain emotional states.

Overall, while both models show promise, there's room for improvement. Fine-tuning parameters and exploring additional features could enhance their predictive power.

REFERENCES

1. **Scikit-learn Documentation:**
   - **Link: https://scikit-learn.org/stable/documentation.html**

2. **Seaborn Documentation:**
   - **Link: https://seaborn.pydata.org/tutorial.html**

3. **Matplotlib Documentation:**
   - **Link: https://matplotlib.org/stable/contents.html**

4. **Pandas Documentation:**
   - **Link: https://pandas.pydata.org/docs/**