LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CU6051NI Artificial Intelligence

Title: NeuroColor

75% Individual Coursework

Submission: Final Submission

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Nabin Bista

London Met ID: 23049053

College ID: NP01CP4A230284

Assignment Due Date: 21/01/2026

Assignment Submission Date: 21/01/2026

Submitted To: Mr. Binod Bhattarai

GitHub Link	*https://github.com/Nabinbista12/neuro-color*

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Acknowledgement

I would like to express thanks to my class teacher and lecturer for valuable feedback and direction throughout the development process. This project was also completed with the help of various learning resources and documentation. Appreciation is given to the developers and open-source community, especially the scikit-learn library team, whose tools and documentation made it possible to build, train, and evaluate the machine learning models efficiently. Finally, sincere thanks are offered to classmates and peers for their suggestions, discussions, and motivation during the project work.

Abstract

This project aims to predict RGB color values from mood-based text using machine learning. The dataset contains mood words or short phrases as input and corresponding RGB values as output. The mood text is preprocessed to reduce noise and improve consistency, then converted into numerical features using TF-IDF. Two regression approaches are implemented for comparison: Random Forest Regressor and Support Vector Regression (LinearSVR) with MultiOutputRegressor for handling the three RGB channels. Model performance is evaluated using R^2, MAE, MSE, and RMSE on a held-out test set, and GridSearchCV is used to tune hyperparameters to improve reliability. The final outcome demonstrates that a mood-to-color mapping can be learned from text features, producing reasonable RGB predictions for new mood inputs, while also highlighting the importance of dataset consistency for stable results.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The project is a color generator or predictor, or recommender based on mood text, built to convert simple mood inputs like “calm”, “happy”, (or even in a sentences) into a meaningful RGB color output. The main idea is to help the user to get some color idea from the feeling they describe, so that it can be used in the UI theme generation, mood visualization dashboards, creative design support, or any design where text-based emotion needs to be shown visually as a color. Instead of choosing color manually choosing colors every time, the system learns pattern from the existing mood color example them produce a suitable RGB color for new mood inputs.

1.1. Topic / AI concepts used

1.1.1. Topic Name (Neurocolor)

The name of the project is Neurocolor. “Neuro” represents the brain-like part of the system which in my project is machine learning to learn data from the pattern from the prediction and recommend (similar to how the brain learns from experience). “Color” directly represents the system generates the RGB color from mood text.

1.1.2. Problem type (Supervised learning: Regression)

The system leans from labelled examples as each mood has its own corresponding RGB color target. Because RGB values are continuous numbers, this is a regression task.

1.1.3. Multi output regression (predicting RGB color which has 3 value)

Instead of predicting one output the model predict 3 output (R, G, B) at once:

- **Input(X):** mood text
- **Output(Y):** [R, G, B]

This is called multi-output regression, because the target is a vector of multiple continuous values (geeksforgeeks, 2025).

1.1.4. Categorical to numeric using TF-IDF (NLP feature extraction)

Machine learning cannot directly understand the raw text as the machine learning good at numerical value, so the mood text is converted into numeric vectors using TF-IDF (Term Frequency Inverse Document Frequency)

- TF increases when a word appears more in a mood text
- IDF decreases when a word is common across many moods

So, TF-IDF highlights words from mood that are important for the machine learning model. The TF-IDF produces a matrix (mostly zeros), which is idea for many ML models (Scikit Learn, 2026).

1.1.5. Hyperparameter tuning with GridSearchCV

Hyperparameter tuning in machine learning is used find the best parameter for the given model for the given data. In the project there are two models SVR and Random Forest Regressor these two models have lot of parameters. And hyperparameter algorithm GridSearchCV automatically tries every combination of these models and give the best parameter. It is used to reduces guesswork and usually improves test performance compared to default settings (Scikit Learn, 2026).

1.1.6. SVR (Support vector regressor)

SVR is built on the concept of the SVM (support vector machine). It is one of the popular algorithms in machine learning models that can be used in classification problems or regression or when the data is not linearly separable (Great Learning, 2024). The SVM or SVR has three kernels:

- Linear Kernel
- Polynommial Kernal
- Radial Basis Function (RBF)

1.1.7. Random Forest Regression

Random Forests are combination of may trees which depend on the parameter of the n_iterators. The generalization error of a forest of tree classifiers depends of the

strength of the individual trees in the forest and the correlation between them. Each tree learns slightly differently due to randomness in data sampling and feature selection. Their averaged prediction becomes the final output (Breiman, 2001).

1.2. Domain/topic

1.2.1. Problem domain overview

NeuroColor falls under affective computing / emotion-aware systems, where human feelings or moods are captured and represented in a form a computer can use. Affective computing group creates and evaluates the emotionAI and other technologies like this for the betterment of people lives.

1.2.2. Why this domain matters

Many digital products need need a quick, intuitive way to visualize emotion. For example: music mood theme, photo editing, creative design palettes, or interface that adapt their theme based on user feeling. NeuroColor addresses this by turning good text into a simple, universal visual signal that can be directly used in UI/graphics.

1.2.3. Key challenge in this domain

Mood to color mapping is not a fixed formula because emotional associations with color can be differ by culture, context and personal experience. Because of this subjectivity, NeuroColor treats the mapping as a data driven learning problem: it learns mood color patterns from examples rather than hard coded rules.

1.3. Aim and Objective

Aims: The aim of this project is to build a AI based model to generate or recommend the color according to the mood text as input and it gives a suitable RGB color output, using machine learning models and hyperparameter turning to improve prediction accuracy and generalization.

Objective:

The objective of this project are as follows:

- Collect the dataset that contains mood text and corresponding RGB values for training and testing.
- Preprocess the dataset by cleanring mood text and ensuring RGB labels are valid numeric values.
- Convert mood text into numerical features using TF-IDF vectorization so it can be used by machine learning algorithms.
- Train and evaluate machine learning algorithms.
- Apply GridSearchCV for better accuracy and better performance of the SVR and RandomForest.
- Evaluate the final tuned models using appropriate regression metrices and compare results to identify the best approach.
- Implement a working application or script where a user inputs a mood word/sentence and the system outputs the predicted RGB color.

2. Background

Human naturally connect color with lot of things. It is said that almost every big company's logo and their product use associate color according to the human psychology of the color. And with the mood human naturally connect their mood with the color. Studies such as Kaya & Epps (2004) show measurable links between color choices and reported emotional responses in participants (Kaya, 2004). Research also suggests that color preference is related to people's average emotional experiences with objects and meaning associated with those color.

2.1. Research review and literature

Colors are widely used as a way to communicate feeling and meaning, not only for decoration. In many design and psychology related works, it is commonly seen that people connect colors with emotion and mood, like calm feeling with softer tones or energetic feeling with brighter tones. Because of that, building a mood-to-color system has a real basis, since the output color can act like a quick emotional signal. Also in daily language, words and descriptions often already suggest a "color vibe", so mood text is not random text, it carries some pattern that can be learned.

2.2. Machine learning concept used

This project fits into supervised learning because the dataset already contains the input (mood related text) and the output (RGB values). Since RGB has three numeric targets, it becomes multi-output regression, meaning the system is not predicting one value but three values together. Another important concept is that text itself is not usable directly in ML, so it needs to be converted into numbers first. That is why feature extraction is needed, so the model can actually learn from the text in a mathematical way.

2.3. Approach that are used in the research

The overall approach follows a simple pipeline where the text is cleaned first, then converted into vectors using TF-IDF, and after that a regression model is trained to predict RGB values. A train-test split is used so the result is checked on unseen data, not only on training samples, otherwise it can give fake good score. Linear SVR is used because it generally works well with sparse TF-IDF features, and a multi-

output wrapper is used so separate prediction can be done for R, G, and B channels. For improving the performance, GridSearchCV is applied to test different parameter settings, and finally the trained vectorizer and model is saved so it can be reused when new mood text is given.

2.4. Research in this project

In this project domain was explored as a text-based mood-to-color prediction problem, where user mood descriptions are transformed into feature vectors and mapped into RGB values. Prior research shows that color attributes such as brightness and saturation can influence emotional responses, and people also tends to associate particular hues with certain emotions in a fairly consistent way. In addition, NLP research indicates many words have stable color associations across users, which support the idea that mood text can be translated into meaningful colors rather than being random output.

Based on this, project investigated a pipeline using TF-IDF for representing the mood text and then applying classical machine learning models (e.g., Random Forest and SVR) for predicting the RGB channels. Since the output is numeric and has multiple channels, the problem was treated as a multi-output regression, and the evaluation plan focused on MAE, MSE, RMSE, and R^2 to check both error size and overall fit. Initial comparison was done to understand which model behaves better on this kind of text features, while keeping the approach simple and explainable for the prototype. After that, GridSearchCV was planned for optimization so that the model can be tuned properly (like changing key parameters) and to make the final results more reliable instead of depending on default settings.

3. Solution

3.1. Used AI algorithm

3.1.1. Linear SVR

In this project, SVM is used as Support Vector Regression (SVR) in its linear form (LinearSVR). It learns a linear relationship between the TF-IDF text features and

the target values by optimizing a margin-based objective with an epsilon-insensitive loss (errors inside the epsilon “tube” are not penalized), and it’s implemented with liblinear, which generally scales better for large, sparse feature spaces like TF-IDF. Because RGB is a 3-output target, MultiOutputRegressor is used to extend LinearSVR by training one separate regressor per target (R, G, and B) (Scikit Learn, 2026).

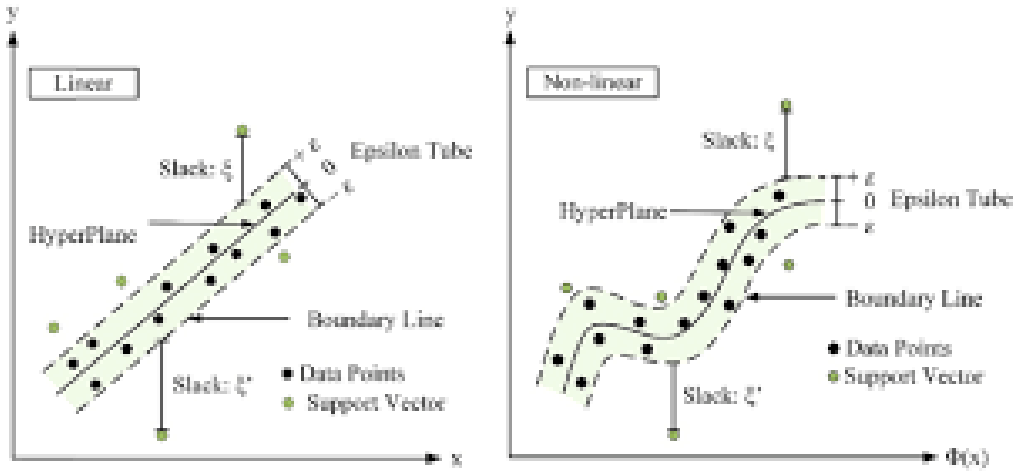

Figure 1: Linear vs non-linear svm (Ditthakit, 2022)

3.1.2. Random Forest Regressor

Random Forest is an ensemble (meta-estimator) that trains many decision tree regressors on different random sub-samples of the dataset and then averages their predictions. This “perturb-and-combine” approach reduces variance compared to a single tree, which helps control overfitting and often improves generalization. In scikit-learn, RandomForestRegressor performs this averaging across its trees, and it naturally supports regression outputs like your RGB targets (Scikit Learn, 2026).

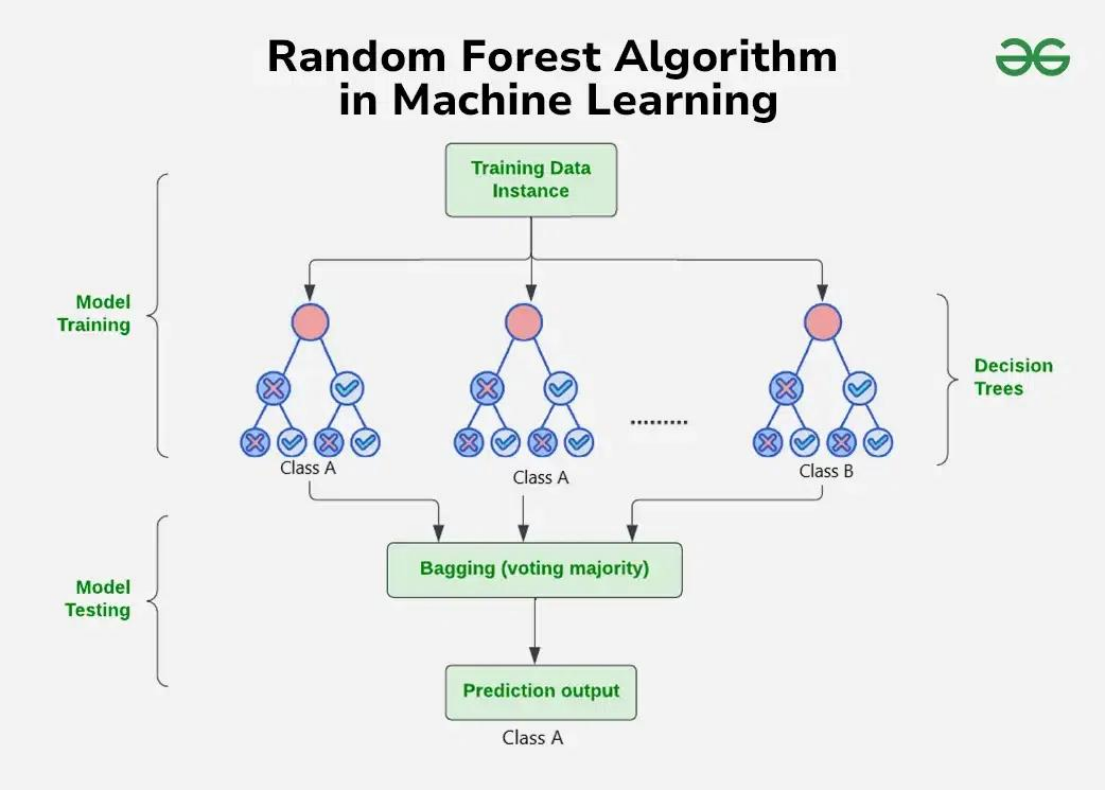

Figure 2: Random Forest Tree (geeksforgeeks, 2025)

3.1.3. GridSearchCV

GridSearchCV is used to systematically tune models by performing an exhaustive search over all combinations in a specified parameter grid, using cross-validation to estimate performance for each combination. After evaluating all candidates, it selects the best setting (e.g., best C, epsilon, etc.) and exposes key results such as best_params_ (the best hyperparameters) and best_score_ (the best cross-validated score) (Scikit Learn, 2026).

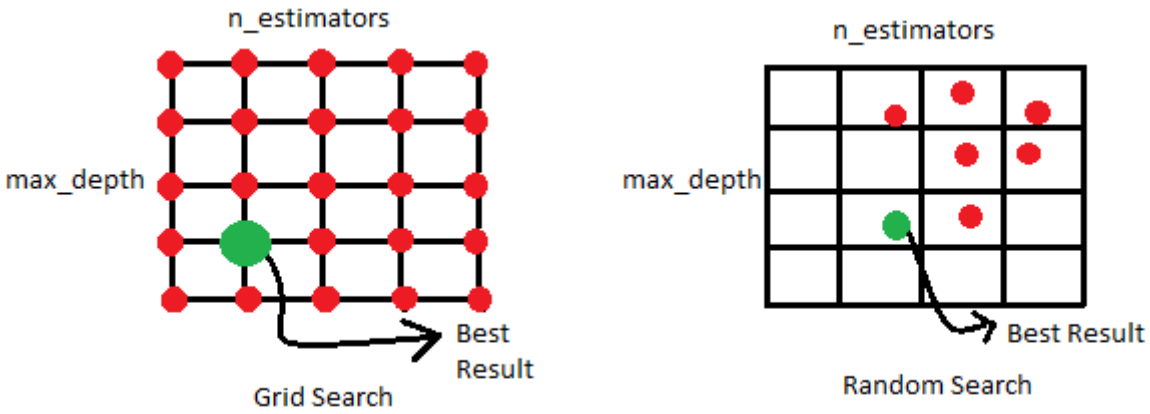

Figure 3: Grid Search CV (Chaurasiya, 2022)

3.2. Pseudocode

3.2.1. Pseudocode for SVR

```
DATA ← read_parquet("color_pedia.parquet")

IF columns {Mood, R, G, B} are not present THEN

    STOP (invalid dataset)

 ENDIF

Remove invalid rows:

 DATA ← rows where Mood is not null AND R,G,B are not null


Define CLEAN_MOOD(text):

    text ← string(text)

    text ← lowercase(text)

    text ← trim(text)

    text ← replace punctuation/symbols with space

    text ← normalize multiple spaces to single space

    RETURN text


DATA.Mood ← apply CLEAN_MOOD to Mood column


X_text ← DATA.Mood

Y_rgb  ← DATA[[R, G, B]]


(X_train_text, X_test_text, Y_train, Y_test) ← train_test_split(
```

```
    X_text, Y_rgb, test_size=0.20, random_state=42
  )

VEC ← TfidfVectorizer(ngram_range=(1,2))
X_train_vec ← VEC.fit_transform(X_train_text)
X_test_vec  ← VEC.transform(X_test_text)

BASE_SVR ← LinearSVR(random_state=42)
SVM_MODEL ← MultiOutputRegressor(BASE_SVR)

SVM_MODEL.fit(X_train_vec, Y_train)

Baseline prediction:
  Y_pred_base ← SVM_MODEL.predict(X_test_vec)

Baseline evaluation:
  Compute R2, MAE, MSE, RMSE (overall + optional per-channel)

Define PARAM_GRID for the wrapped estimator:
  PARAM_GRID = {
    "estimator__C":       [ ... ],
    "estimator__epsilon": [ ... ],
    "estimator__tol":     [ ... ],
```

```
    "estimator__loss":    [ ... ]
  }

GRID ← GridSearchCV(
    estimator = SVM_MODEL,
    param_grid = PARAM_GRID,
    cv = K,
    verbose = 1
  )

GRID.fit(X_train_vec, Y_train)
```

Extract best configuration:

```
  BEST_PARAMS ← GRID.best_params_
  BEST_SCORE  ← GRID.best_score_
  BEST_MODEL  ← GRID.best_estimator_

Y_pred_best ← BEST_MODEL.predict(X_test_vec)
```

Compute final test metrics:

```
  R2_overall, MAE_overall, MSE_overall, RMSE_overall
  (Optional) per-channel: raw_values
```

INPUT mood_text

mood_text ← CLEAN_MOOD(mood_text)

mood_vec ← VEC.transform([mood_text])

rgb_pred ← BEST_MODEL.predict(mood_vec)

END

3.2.2. Pseudocode for Random Forest

```
DATA ← read_parquet("color_pedia.parquet")

IF columns {Mood, R, G, B} are not present in DATA THEN
    STOP (invalid dataset)
ENDIF

Remove invalid rows:
    DATA ← rows where Mood is not null AND R,G,B are not null

Define CLEAN_MOOD(text):
    text ← string(text)
    text ← lowercase(text)
    text ← trim(text)
    text ← replace punctuation/symbols with space
    text ← replace multiple spaces with single space
    RETURN text

DATA.Mood ← apply CLEAN_MOOD to every Mood value

X_text ← DATA.Mood
Y_rgb  ← DATA[[R, G, B]]   (multi-output target)
```

```
(X_train_text, X_test_text, Y_train, Y_test) ← train_test_split(
      X_text, Y_rgb, test_size=0.20, random_state=42
   )

VEC ← TfidfVectorizer(ngram_range=(1,2))

X_train_vec ← VEC.fit_transform(X_train_text)
X_test_vec  ← VEC.transform(X_test_text)

RF ← RandomForestRegressor(parameters...)
RF.fit(X_train_vec, Y_train)

Y_pred ← RF.predict(X_test_vec)

Overall metrics (average over RGB):
   R2   ← r2_score(Y_test, Y_pred, multioutput="uniform_average")
   MAE  ← mean_absolute_error(Y_test, Y_pred, multioutput="uniform_average")
   MSE  ← mean_squared_error(Y_test, Y_pred, multioutput="uniform_average")
   RMSE ← sqrt(MSE)
Per-channel metrics (R, G, B separately):
   R2_rgb   ← r2_score(Y_test, Y_pred, multioutput="raw_values")
   MAE_rgb  ← mean_absolute_error(Y_test, Y_pred, multioutput="raw_values")
   MSE_rgb  ← mean_squared_error(Y_test, Y_pred, multioutput="raw_values")
```

```
    RMSE_rgb ← sqrt(MSE_rgb)


INPUT mood_text

mood_text ← CLEAN_MOOD(mood_text)

mood_vec  ← VEC.transform([mood_text])

rgb_pred  ← RF.predict(mood_vec)


END
```

3.3. Diagrammatical representation

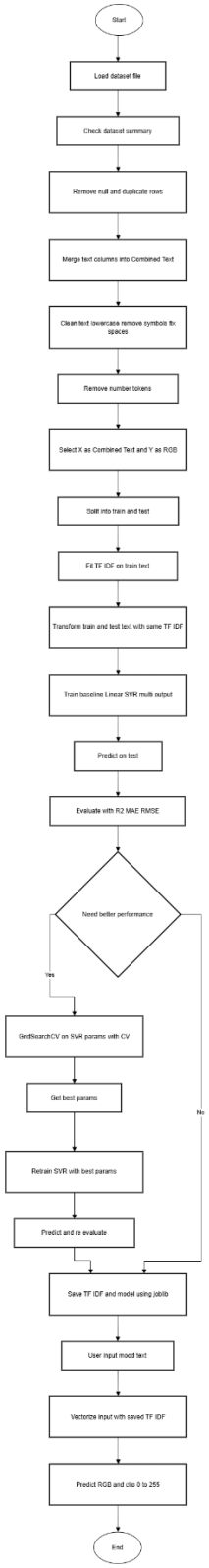

Figure 4: SVR flowchart

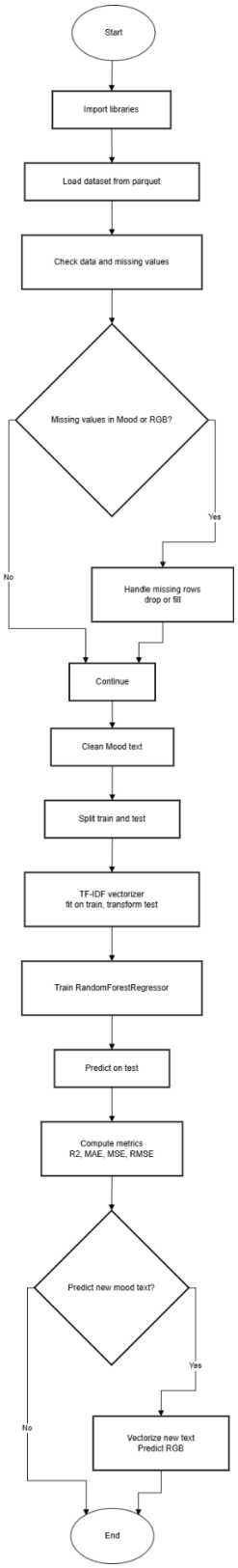

Figure 5: Random Forest Tree Flowchart

3.4. Explanation of the development process

3.4.1. Development process

3.4.1.1. Development process of LinearSVR with GridSearchCV

a. Notebook setup

Required Python libraries are installed and imported, so the notebook can read the dataset, process the text, train the models and calculate the evaluation metrics without any missing library issue.

b. Dataset preparation

The dataset is loaded from a parquet file, so the table data (text fields and RGB columns) becomes available for model training. Then quick checking is done using preview and dataset summary (shape/size and description), mainly to confirm dataset is not empty and RGB values is in valid range. Missing values are checked and removed because null text can break vectorization and null RGB makes the supervised learning not possible, so training can become unstable.

c. Text building and preprocessing

Multiple text columns like Description, Emotion, Mood, Symbolism and Keywords are merged into one “Combined Text”, because using only one column may lose lot of information and combining gives more context. After that, the combined text is cleaned by converting into lowercase and removing punctuation/special symbols using regex, then extra spaces are fixed. This helps because “Calm”, “calm!” and “ calm ” should not be treated as different tokens. Another cleaning step removes number-based tokens since numbers normally don’t add meaning for mood and it just creates noisy features.

d. Train-test split

The combined text is taken as input (X) and RGB columns are taken as output (Y). The dataset is split into training and testing (like 80/20) so evaluation

can be done on unseen data. A fixed random state is used so each run gives same split, otherwise the score may change randomly and make confusion.

e. Feature extraction (TF-IDF)

TF-IDF is used to convert the cleaned text into numerical vectors, because ML models can't learn from raw string directly. Vectorizer settings are chosen to reduce noise, like removing stopwords, using unigram and bigram to capture small phrases, using sublinear term frequency, filtering too rare and too common words, and limiting maximum features so training don't become too slow. The vectorizer is fit on training text only and then applied on test text, because fitting on whole dataset can leak information and gives fake better result.

f. Model development (baseline)

LinearSVR is used as the regression model because TF-IDF creates high dimensional sparse features and linear model usually works good on that. Since RGB has 3 outputs, a multi-output wrapper is used so the system trains separate internal SVR model for R, G and B.

g. Evaluation (baseline results)

Train and test R^2 score is checked to understand if model is overfitting or generalizing. Predictions are generated on test set, and evaluated using MAE, MSE, RMSE and R^2. Both overall average metric and per-channel metrics are calculated so it shows performance for each RGB channel also.

h. Prediction output step

A prediction step is made where a sample input text is vectorized and passed into the trained model to get RGB output. The predicted values are rounded and clipped into 0–255 because regression can output float values or sometimes go outside valid RGB range.

i. Hyperparameter tuning (GridSearchCV)

GridSearchCV is applied to test different parameter combinations (like C, epsilon, loss type and intercept). Verbose is enabled so progress can be seen during fitting. After getting best parameters, the model is retrained using those settings and evaluated again to compare with baseline result.

j. Saving the trained pipeline

Finally, the trained TF-IDF vectorizer and the trained model is saved using joblib, so the same vector space and learned weights can be reused later during application prediction.

3.4.1.2. Development process of Random Forest

a. Notebook setup

The notebook first installs and imports the required Python libraries, so the whole work can run properly like reading the dataset, cleaning the text, training the models and calculating the evaluation metrics without facing missing package problem.

b. Dataset preparation

c. The dataset is loaded from a parquet file, so all the text fields and RGB columns becomes available for training. After loading, a quick preview and dataset summary (shape/size and description) is checked just to make sure the dataset is not empty and the RGB values are looking in the correct range. Then missing values are checked and removed, because if the text is null it can break vectorization, and if RGB is null then supervised learning can't work, so training

Text building and preprocessing

result may become unstable or misleading.

Multiple text columns like Description, Emotion, Personality, Mood, Symbolism, Use Case and Keywords are merged into one "Combined Text", because using only one column can lose many useful information and the combined text gives more context for learning. After that, the combined text is cleaned by making it lowercase and removing punctuations/special symbols using regex, and extra spaces are fixed. This is done because words like "Calm", "calm!" and " calm " should not act like different tokens. Another cleaning step removes number based tokens, since numbers normally don't carry mood meaning and it just adds noisy features in TF-IDF.

d. Train-test split

The combined text is taken as input (X) and RGB columns are taken as output (Y). The dataset is split into training and testing (around 80/20) so evaluation can be done on unseen data and not only on the training set. A fixed

random state is used so the split stays same each time, otherwise the score can change randomly and it makes confusion while comparing results.

e. Feature extraction (TF-IDF)

TF-IDF is used to transform the cleaned text into numerical vectors, because ML model can't understand raw string directly. Some vectorizer settings are used to reduce noise, like removing stopwords, using unigram and bigram to catch small phrases, using sublinear term frequency so repeated words don't dominate too much, filtering too rare/common words, and limiting maximum features so training does not become too slow. The vectorizer is fitted only on training data and then applied to test data, because fitting on the full dataset can cause data leakage and give fake better performance.

f. Model development

LinearSVR is used as the baseline regression model since TF-IDF produces high dimensional sparse features and linear models usually works good on this kind of input. Because RGB has 3 outputs, a multi-output wrapper is used so the system trains separate internal models for R, G and B channels, but it still behaves like one pipeline for prediction.

g. Evaluation

Train and test R^2 score is checked first to see if the model is overfitting or generalizing well. Then predictions are generated on the test set and evaluated using MAE, MSE, RMSE and R^2. Both overall average metrics and per-channel metrics are calculated, so it becomes clear if one channel (like B) is performing worse than others.

h. Prediction output step

A prediction step is created where a sample mood text is vectorized using the same trained TF-IDF and passed into the trained model to get RGB output. The predicted values are rounded and clipped into 0–255 range, because

regression can output float values and sometimes it can go outside the valid RGB limit.

i. Saving the trained pipeline

Finally, the trained TF-IDF vectorizer and the trained model is saved using joblib, so the same vector space and learned weights can be reused later when doing prediction in the application, without needing to train again.

3.4.2. Tools and Technologies/libraries used

Tools used in this project:

Tools	Description
Python	Python is primarily used as a main programming language for this project as many ml libraries are built using python.
Jupyter Notebook/ VS code	To build this project VS code IDE is used with the help of the Jupyter Notebook extension in the VS code.
Draw.io	Draw.io is used to make the diagram for this project.
Word	For the documentation of this project word is used.
Google	Goole is used to do research in this project.
MySecondTeacher	Mysecondteacher is used for the submission and detail of the project.

Table 1: Tools used in this project

Technologies used in this project are:

Technologies	Description
Numpy	Numpy as numerical python is used for handling the mathematical part of the program
Pandas	Pandas is used to handling the data and inserting data in the project.
Matplotlib	Matplotlib is for the diagram of the project
Seaborn	Seaborn used matplotlib but it is used for higher diagram of the project.
Scikit Learn	Scikit Learn or sklearn provide all the required models for the project.

Table 2: Technologies used in this project

3.5. Achieved result

3.5.1. Achieved result of the LinearSVR

```
svr.score(x_test, y_test), svr.score(x_train, y_train)
✓ 0.0s
(0.7829060966580975, 0.7899108267592426)
```

Figure 6: Score of linear svr

The first result is the Linear SVR score (R^2) on test and train set. The test R^2 is **0.7829** and the train R^2 is 0.7899, so both are almost same. This means the model is learning properly and it is not heavily overfitting, because if train score was very high but test was low then it would show memorizing. Here the small gap shows the prediction performance is stable on unseen data also.

```
r2_overall = r2_score(y_test, y_pred, multioutput="uniform_average")
mae_overall = mean_absolute_error(y_test, y_pred, multioutput="uniform_average")
mse_overall = mean_squared_error(y_test, y_pred, multioutput="uniform_average")
rmse_overall = np.sqrt(mse_overall)

print("R2  :", r2_overall)
print("MAE :", mae_overall)
print("MSE :", mse_overall)
print("RMSE:", rmse_overall)
✓ 0.0s
R2  : 0.7829060966580975
MAE : 27.186723751160354
MSE : 1181.797535989337
RMSE: 34.37728226589962
```

Figure 7: Metrics of the LinearSVR

The second output shows the overall (uniform average) metrics for Linear SVR, meaning it gives one single value by averaging R, G, B equally. The overall R^2 is 0.7829, which means around 78% of the variation in RGB values is explained compared to baseline mean prediction. The MAE is 27.1867, so on average the prediction is off by about 27 RGB levels, and the RMSE is 34.3773, which is a bit higher because it punish bigger

errors more. The MSE is 1181.7975, but it is in squared unit so it looks large and not so easy to understand directly.

```
r2_rgb = r2_score(y_test, y_pred, multioutput="raw_values")
mae_rgb = mean_absolute_error(y_test, y_pred, multioutput="raw_values")
mse_rgb = mean_squared_error(y_test, y_pred, multioutput="raw_values")
rmse_rgb = np.sqrt(mse_rgb)

print("R2   [R,G,B]:", r2_rgb)
print("MAE  [R,G,B]:", mae_rgb)
print("MSE  [R,G,B]:", mse_rgb)
print("RMSE [R,G,B]:", rmse_rgb)
```

✓ 0.0s

```
R2   [R,G,B]: [0.74730953 0.83121261 0.77019615]
MAE  [R,G,B]: [29.14387743 24.10481163 28.31148219]
MSE  [R,G,B]: [1377.34224758  917.29583949 1250.7545209 ]
RMSE [R,G,B]: [37.11256186 30.28689221 35.36600799]
```

Figure 8: Metrics of Individual values

The third output is the single (per-channel) metrics for Linear SVR, where results are shown separately for R, G, B. The R^2 values are [0.7473, 0.8312, 0.7702], so the Green channel is performing best (highest R^2), and Red is slightly weaker than others. For error values, MAE is [29.1439, 24.1048, 28.3115] and RMSE is [37.1126, 30.2869, 35.3660], which again shows Green has lower error compare to Red and Blue. This per-channel view is useful because even if overall score looks good, it still shows which RGB channel is more difficult for the model.

3.5.2. Achieved Result of LinearSVR with GridSearchCV

```
grid_svr.score(x_train, y_train), grid_svr.score(x_test, y_test)
✓ 0.0s
(0.8292104657115917, 0.8031982649279836)
```

Figure 9: Score of the linearSVR in GridSearchCV

The first output is the tuned Linear SVR (GridSearchCV) score in terms of R^2. The train R^2 is 0.8292 and the test R^2 is 0.8032, so after tuning the model improved compare to baseline. The gap is still not huge, so it is not like heavy overfitting, but train is a bit higher which is normal because model learns from train data more.

```
r2_overall = r2_score(y_test, grid_predict, multioutput="uniform_average")
mae_overall = mean_absolute_error(y_test, grid_predict, multioutput="uniform_average")
mse_overall = mean_squared_error(y (variable) y_test: Any ioutput="uniform_average")
rmse_overall = np.sqrt(mse_overall,

print("R2  :", r2_overall)
print("MAE :", mae_overall)
print("MSE :", mse_overall)
print("RMSE:", rmse_overall)
✓ 0.0s
R2  : 0.8031982649279836
MAE : 25.816371300857128
MSE : 1071.3250464749365
RMSE: 32.73110212740989
```

Figure 10: Metrics of the LinearSVR in GridSearchCV

The second result shows the overall (uniform average) metrics for the tuned Linear SVR, means it average the performance of R, G and B into one value. The overall R^2 is 0.8032, which is better than before, so the model explains more variance in RGB prediction. The MAE is 25.8164, so average error drop a bit (now around 25 RGB units), and the RMSE is 32.7311, which also decreased showing fewer big mistakes. The MSE is 1071.3250, and it also become smaller because errors are reduced overall.

```
r2_rgb = r2_score(y_test, grid_predict, multioutput="raw_values")
mae_rgb = mean_absolute_error(y_test, grid_predict, multioutput="raw_values")
mse_rgb = mean_squared_error(y_test, grid_predict, multioutput="raw_values")
rmse_rgb = np.sqrt(mse_rgb)

print("R2   [R,G,B]:", r2_rgb)
print("MAE  [R,G,B]:", mae_rgb)
print("MSE  [R,G,B]:", mse_rgb)
print("RMSE [R,G,B]:", rmse_rgb)
```

✓ 0.0s

```
R2   [R,G,B]: [0.77303991 0.84622552 0.79032936]
MAE  [R,G,B]: [27.53327713 22.99032847 26.9255083 ]
MSE  [R,G,B]: [1237.0934105   835.70630383 1141.1754251 ]
RMSE [R,G,B]: [35.17233871 28.9085853  33.78128809]
```

Figure 11: Metrics of the Individual Value

The third output is the per-channel (single) metrics for tuned Linear SVR, shown separately for R, G, B. The R^2 values are [0.7730, 0.8462, 0.7903], so again the Green channel is best and the model predict G more accurately than R and B. The MAE values are [27.5333, 22.9903, 26.9255] and RMSE values are [35.1723, 28.9086, 33.7813], which shows errors reduced in all channels compare to baseline, specially for Green it is lowest. This single metrics view is helpful because it clearly shows which color channel is performing strong and which one still need improvement.

3.5.3. Achieved Result of the RandomForest

```
rf.score(x_test, y_test), rf.score(x_train, y_train)
✓ 2.7s

[Parallel(n_jobs=1)]: Done  40 tasks      | elapsed:    0.1s
[Parallel(n_jobs=1)]: Done  40 tasks      | elapsed:    0.5s

(0.7416899144059528, 0.7695721066128205)
```

Figure 12: Score for the Random forest

The first output is the Random Forest regressor score (R^2) for test and train. The test R^2 is 0.7417 and the train R^2 is 0.7696, so train is higher than test but not too extreme. This shows the model is learning patterns from training data but it still generalize in test set in an okay way. The “Parallel done 40 tasks” line is just showing the model is running trees in background, it is not an error.

```
r2_overall = r2_score(y_test, y_pred, multioutput="uniform_average")
mae_overall = mean_absolute_error(y_test, y_pred, multioutput="uniform_average")
mse_overall = mean_squared_error(y_test, y_pred, multioutput="uniform_average")
rmse_overall = np.sqrt(mse_overall)

print("R2  :", r2_overall)
print("MAE :", mae_overall)
print("MSE :", mse_overall)
print("RMSE:", rmse_overall)
✓ 0.0s

R2  : 0.7416899144059528
MAE : 30.648920946602363
MSE : 1406.162033434616
RMSE: 37.498827094118774
```

Figure 13: Metrics for the random forest

The second output shows the overall (uniform average) metrics for Random Forest, which gives one combined value for all RGB channels. The overall R^2 is 0.7417, meaning the model explains around 74% of the RGB variation compared to baseline mean prediction. The MAE is 30.6489, so on average it is off by around 30 RGB levels, and the RMSE is

37.4988, which is higher because it punish large mistakes more. The MSE is 1406.1620, and this value looks bigger mainly because it is squared error unit.

```
r2_rgb = r2_score(y_test, y_pred, multioutput="raw_values")
mae_rgb = mean_absolute_error(y_test, y_pred, multioutput="raw_values")
mse_rgb = mean_squared_error(y_test, y_pred, multioutput="raw_values")
rmse_rgb = np.sqrt(mse_rgb)

print("R2   [R,G,B]:", r2_rgb)
print("MAE  [R,G,B]:", mae_rgb)
print("MSE  [R,G,B]:", mse_rgb)
print("RMSE [R,G,B]:", rmse_rgb)
```

✓ 0.0s

```
R2   [R,G,B]: [0.70693447 0.80495312 0.71318216]
MAE  [R,G,B]: [32.67104568 26.71455451 32.56116265]
MSE  [R,G,B]: [1597.4149628  1060.00628679 1561.06485071]
RMSE [R,G,B]: [39.96767397 32.55773774 39.51031322]
```

Figure 14: Metrics of the model

The third output is the per-channel (single) metrics for Random Forest, shown separately for R, G, B. The R^2 values are [0.7069, 0.8050, 0.7132], so again Green channel performs best while Red and Blue are lower. The MAE values are [32.6710, 26.7146, 32.5612] and RMSE values are [39.9677, 32.5577, 39.5103], which shows Red and Blue have larger error compare to Green. This single metrics is useful because overall score might look okay, but it clearly shows which channel is giving more error and where the model is weak.

3.5.4. Comparison of the model

Model	Train R^2	Test R^2	Overall MAE	Overall RMSE	Best?
Linear SVR (baseline)	0.7899	0.7829	27.1867	34.3773	No
Linear SVR (tuned – GridSearchCV)	0.8292	0.8032	25.8164	32.7311	Yes (Best)
Random Forest	0.7696	0.7417	30.6489	37.4988	No

Table 3: Comparison table

The comparison table shows that tuned Linear SVR (GridSearchCV) did the best overall. It has the highest test R^2 (0.8032), which means it explains the RGB variation better on unseen data, and it also has the lowest MAE (25.8164) and lowest RMSE (32.7311), so the prediction error is smaller compare to other models. The baseline Linear SVR is second best, and Random Forest performed the weakest here because it has lower test R^2 (0.7417) and higher error values.

4. Conclusion

4.1. Analysis of work done

The analysis of the work done shows that the whole pipeline worked properly from text input to RGB prediction, and the results also proves the approach is valid for this problem. The text preprocessing and combining multiple text columns helped to give more context, and TF-IDF was able to convert the mood text into useful numeric features for training. Since the outputs are RGB values, treating it as multi-output regression was the correct choice, because it allows predicting the three channels together in one workflow.

From the model performance side, Linear SVR performed better than Random Forest for this TF-IDF based text features, which make sense because linear models often handle sparse high-dimensional vectors more effectively. Hyperparameter tuning with GridSearchCV improved the SVR results further, where test R^2 increased and MAE/RMSE decreased, meaning the predictions became more accurate and more stable. Also, the train and test scores were relatively close, which indicates the model is not overfitting too much and it can generalize reasonably. However, the per-channel evaluation showed that some channels (like Red/Blue) still had higher error than Green, so there is still room to improve feature quality or model capacity for more consistent RGB prediction.

4.2. Project real world applications

This project solves a real world problem by converting mood text into a color output, which makes emotion and feeling more easier to visualize. In many cases people can’t describe their mood clearly, but a color can represent the vibe quickly and it can be used as a simple communication tool. For example, in UI/UX design, a system like this can recommend theme colors based on user mood (calm, energetic, sad, confident) which helps in making more user-personalized interfaces and better experience.

It can also be useful in creative fields like posters, branding, digital art, and content creation where color selection takes time and depends on emotion. Instead of manually choosing palette, the model gives a starting color suggestion from text, so it saves time and reduce confusion for beginners. In mental health or journaling type apps, it can work like a small mood tracker where users write how they feel and the system generate a color, making patterns easier to notice over days (like if most outputs are dark tones, it may represent low mood). So overall, it gives a practical way to connect natural language mood into visual color, which is meaningful in real use cases.

4.3. Future Work

For further work, the system can be improved by using more advanced NLP models instead of only TF-IDF, like word embeddings or transformer based models (example BERT type), because TF-IDF mainly depends on word frequency and it sometimes miss the deeper meaning of a sentence. Also the dataset can be expanded and balanced more, because some moods may have less samples and that makes prediction less accurate for those type of inputs. Adding more training data with more variety of mood phrases will help the model generalize better.

Another future improvement is to predict not only one RGB color but a full color palette (like 3–5 colors) so it becomes more useful for designers and UI themes. A simple web interface can be built where user types mood and it directly shows the color box, HEX code, and palette suggestions with preview (like background + text color). Also more evaluation can be done with human testing, where users rate if the predicted color actually matches their mood, because regression metrics alone don’t fully capture “feels right” part. Finally, deployment can be improved by making a lightweight API or local app so it can run smoothly without needing notebook environment.

5. References

Breiman, L., 2001. Random Forests. *Random Forests,* p. 33.

Chaurasiya, P., 2022. *Hyperparameter Tuning Using GridSearchCV and RandomSearchCV!.* [Online] Available at: https://medium.com/@priya38/hyperparameter-tuning-using-gridsearchcv-and-randomsearchcv-d43b58f83a81
[Accessed 18 1 2026].

Ditthakit, P., 2022. *Comparative study of machine learning methods and GR2M model for monthly runoff prediction.* [Online] Available at: https://www.researchgate.net/publication/363295192_Comparative_study_of_machine_learning_methods_and_GR2M_model_for_monthly_runoff_prediction
[Accessed 12 1 2026].

geeksforgeeks, 2025. *Multioutput Regression in Machine Learning.* [Online] Available at: https://www.geeksforgeeks.org/machine-learning/multioutput-regression-in-machine-learning/
[Accessed 2 1 2026].

geeksforgeeks, 2025. *Random Forest Algorithm in Machine Learning.* [Online] Available at: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/
[Accessed 14 1 2026].

Great Learning, 2024. *Support Vector Regression in Machine Learning.* [Online] Available at: https://www.mygreatlearning.com/blog/support-vector-regression/
[Accessed 2 1 2026].

Kaya, N., 2004. Relationship between color and emotion: a study of college. *Relationship between color and emotion: a study of college,* p. 12.

Scikit Learn, 2026. *GridSearchCV.* [Online] Available at: https://scikit-

learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[Accessed 2 1 2026].

Scikit Learn, 2026. *GridSearchCV.* [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[Accessed 3 1 2026].

Scikit Learn, 2026. *LinearSVR.* [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html
[Accessed 3 1 2026].

Scikit Learn, 2026. *RandomForestRegressor.* [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[Accessed 3 1 2026].

Scikit Learn, 2026. *TfidfTransformer.* [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html
[Accessed 2 1 2026].