Università degli Studi di Salerno

Progetto di Fondamenti di Intelligenza Artificiale

FeelTrack

Analizzare il sentiment dei post sui diversi social media e confrontarli tra loro

2025

Nome	Matricola
Castiello Mattia	*0512120487*
Paolillo Valentina	*0512114820*

Link Github: FeelTrack

Indice

Introduzione:

Nell'era digitale, i social media giocano un ruolo centrale nella comunicazione e nella diffusione di contenuti. Ogni giorno, milioni di utenti interagiscono attraverso post, commenti e condivisioni, generando un'enorme quantità di dati testuali. Comprendere il sentiment e le emozioni espresse nei contenuti social è fondamentale per aziende, analisti e ricercatori, poiché permette di interpretare il coinvolgimento degli utenti e le dinamiche di interazione online.

Il progetto FeelTrack si propone di sviluppare un sistema automatizzato per l'analisi delle emozioni nei testi pubblicati sui social media. Attraverso l'uso di tecniche di machine learning e analisi del linguaggio naturale (NLP), il sistema sarà in grado di classificare il sentiment dei post e analizzare il loro impatto in termini di interazioni, come like, commenti e condivisioni.

1. Definizione del problema

1.1. Obiettivo

L'obiettivo principale di **FeelTrack** è sviluppare un sistema automatizzato in grado di analizzare il sentiment e le emozioni espresse nei contenuti testuali dei social media. Attraverso l'impiego di modelli di machine learning e tecniche di analisi del linguaggio naturale, il progetto mira a classificare le emozioni presenti nei post e a valutare il loro impatto in base alle interazioni degli utenti, come like, commenti, condivisioni e numero di follower.

In particolare, **FeelTrack** si propone di:

- Implementare un modello di classificazione del sentiment basato su machine learning.
- Analizzare e visualizzare le emozioni espresse nei testi attraverso grafici e rappresentazioni statistiche.
- Studiare la correlazione tra il sentiment dei post e il livello di engagement degli utenti.
- Fornire insight utili per strategie di marketing, analisi di trend e comprensione del comportamento degli utenti sui social media.

2. Raccolta, analisi e preprocessing dei dati

2.1. La scelta del dataset

Venendo al dataset, necessario per la creazione del modello di machine learning, le possibili strade da seguire sono due:

1. Creare un dataset da zero, formulando un questionario da far riempire ad un campione di persone, per poterne carpire le preferenze;
2. Cercare sulla rete un dataset già formato, e adeguarlo alle nostre esigenze;

La prima opzione era soggetta a limitazioni non trascurabili:

- Scarsità di dati
- Possibile inconsistenza dei dati, dovuta alle risposte degli utenti non sempre veritiere.

Si è deciso, pertanto, di procedere con la seconda soluzione, cercando in rete un dataset già creato.

Per lo sviluppo di questo progetto, sono stati utilizzati tre dataset dalla piattaforma Kaggle:

Train *(⚠️Per visualizzare il train bisogna andare nella cartella Sentiment Analysis e trovare il file Train.csv)*
Twitter Sentiment Dataset
Social Media Sentiment Analysis Dataset

Il primo e il secondo dataset sono stati combinati per effettuare analisi di sentiment applicata al trading. Questa unione ha consentito di ottenere un dataset più ampio e variegato, migliorando la capacità del modello di individuare pattern
utili per il trading.

Il terzo dataset è stato invece impiegato esclusivamente per la fase di test del modello. Questo approccio ha permesso di valutare le performance del sistema su dati non visti in fase di addestramento, garantendo una verifica oggettiva dell'accuratezza della classificazione del sentiment.

L'utilizzo di questi dataset ha contribuito alla creazione di un modello solido ed efficace nell'analisi del sentiment, con possibili applicazioni nell'ambito del trading basato sulle emozioni degli utenti.

2.2. Analisi del dataset

2.2.1. Social_Media_Sentiment_Analysis

CAMPO	**SIGNIFICATO**
User	L'identificativo univoco dell'utente che ha pubblicato il contenuto.
Platform	La piattaforma di social media su cui è stato pubblicato il post.
Post	Il contenuto testuale del post sui social media, che rappresenta opinioni, reazioni o feedback degli utenti.
Hashtag	Gli hashtag associati al post, che riflettono i temi di discussione.
Sentiment	Il sentiment del post, classificato come Positivo, Negativo o Neutro.

Likes	Il numero di "mi piace" ricevuti dal post, che rappresentano il livello di engagement.
Shares	Il numero di volte che il post è stato condiviso, mostrando quanto ha risuonato con il pubblico.
Comments	Il numero di commenti ricevuti dal post, che indicano il livello di interazione.

2.2.2. Train

CAMPO	**SIGNIFICATO**
Id	A unique identifier for each entry.
Body	The text content or main body of the entry.
Sentiment Type	The sentiment classification of the text (e.g., positive, negative, neutral).

2.2.3. Twitter Sentiment Dataset

CAMPO	**SIGNIFICATO**
clean_text	The text content or main body of the entry.
category	The sentiment classification of the text

2.3. Formattazione dei dati

La formattazione dei dati è stata effettuata su più livelli ed in più fasi, al fine di eliminare dati rumorosi.

2.3.1. Caricamento e pulizia dei dati

I dataset utilizzati per l'addestramento e il test sono stati caricati utilizzando **pandas**.

2.3.1.1. Rimozione delle colonne non necessarie

Il dataset comprendeva dati che si riferivano anche a informazioni di cui non interessava farne l'analisi. A fronte di ciò, si è avuta una fase di tagli verticali.

```
# Drop not used columns
train_pt1 = train_pt1.drop(columns=['Id'], axis=1)
```

2.3.1.2. Gestione dei valori mancanti

Eventuali valori nulli presenti nei dataset sono stati rimossi.

```
[ ]  # Drop rows where any column has missing values
     train = train.dropna()
     test = test.dropna()
```

2.3.2. Pre-processing del testo

Poiché l'analisi del sentiment si basa su dati testuali, il contenuto dei post è stato elaborato con tecniche di **Natural Language Processing (NLP)**.

2.3.2.1. Tokenizzazione dei dati

Suddivisione del testo in parole o frasi per facilitarne l'analisi.
Dividiamo il testo in parole con `nltk.word tokenize` e rimuoviamo i caratteri non alfabetici.

```
import nltk
nltk.download('punkt')

train['Body'] = train['Body'].apply(lambda x: " ".join(
    [re.sub('[^A-Za-z]+', '', token) for token in nltk.word_tokenize(x)]
))
test['Post Content'] = test['Post Content'].apply(lambda x: " ".join(
    [re.sub('[^A-Za-z]+', '', token) for token in nltk.word_tokenize(x)]
))
```

2.3.2.2. Rimozione delle stopwords

Eliminazione delle parole comuni (es. "il", "e", "ma") che non aggiungono valore semantico all'analisi.

```
from nltk.corpus import stopwords
nltk.download('stopwords')

stop = stopwords.words('english')
train['Body'] = train['Body'].apply(lambda x: " ".join(
    [word for word in x.split() if word not in stop]
))
test['Post Content'] = test['Post Content'].apply(lambda x: " ".join(
    [word for word in x.split() if word not in stop]
))
```

2.3.2.3. Lemmatizzazione

Conversione delle parole alla loro radice per ridurre la variabilità lessicale (es. "correndo" → "correre").

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
train['Body'] = train['Body'].apply(lambda x: " ".join(
    [lemmatizer.lemmatize(word) for word in nltk.word_tokenize(x)]
))
test['Post Content'] = test['Post Content'].apply(lambda x: " ".join(
    [lemmatizer.lemmatize(word) for word in nltk.word_tokenize(x)]
))
```

2.3.2.4. Pulizia del testo

Rimozione di caratteri speciali, numeri e punteggiatura per migliorare la qualità dei dati in ingresso.

```
import re

def contractions(s):
    s = re.sub(r"won't", "will not", s, flags=re.IGNORECASE)
    s = re.sub(r"can't", "can not", s, flags=re.IGNORECASE)
    s = re.sub(r"n't", " not", s, flags=re.IGNORECASE)
    s = re.sub(r"'re", " are", s, flags=re.IGNORECASE)
    s = re.sub(r"'s", " is", s, flags=re.IGNORECASE)
    s = re.sub(r"'ll", " will", s, flags=re.IGNORECASE)
    s = re.sub(r"'t", " not", s, flags=re.IGNORECASE)
    s = re.sub(r"'ve", " have", s, flags=re.IGNORECASE)
    s = re.sub(r"'m", " am", s, flags=re.IGNORECASE)
    return s

train['Body'] = train['Body'].apply(lambda x: contractions(x))
test['Post Content'] = test['Post Content'].apply(lambda x: contractions(x))
```

2.3.3. Normalizzazione e codifica

Per rendere i dati adatti ai modelli di machine learning, sono state applicate operazioni di trasformazione. Grazie a queste operazioni, il dataset è stato reso coerente e strutturato, permettendo ai modelli di machine learning di apprendere in modo efficace le caratteristiche dei dati e migliorare la capacità predittiva del sistema.

2.3.3.1. Conversione in formato numerico

Le variabili testuali sono state convertite in rappresentazioni numeriche utilizzando tecniche di vettorizzazione come **TF-IDF** o **word embeddings**.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Inizializzazione del vettorizzatore TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=6000, ngram_range=(1, 2))

# Trasformazione del testo in rappresentazione numerica
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print("Shape of X_train_tfidf:", X_train_tfidf.shape)
print("Shape of X_test_tfidf:", X_test_tfidf.shape)
```

2.3.3.2. Bilanciamento del dataset

Per evitare squilibri nelle classi di sentiment, sono state applicate tecniche di oversampling o undersampling, se necessario.

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

# Mostra la distribuzione delle classi prima del bilanciamento
print("Distribuzione delle classi prima:", Counter(y_train))

# Scegli tra oversampling o undersampling
apply_oversampling = True  # Imposta a False se vuoi usare l'undersampling

if apply_oversampling:
    smote = SMOTE(random_state=42)
    X_train_balanced, y_train_balanced = smote.fit_resample(X_train_tfidf, y_train)
else:
    undersampler = RandomUnderSampler(random_state=42)
    X_train_balanced, y_train_balanced = undersampler.fit_resample(X_train_tfidf, y_train)

# Mostra la distribuzione delle classi dopo il bilanciamento
print("Distribuzione delle classi dopo:", Counter(y_train_balanced))
```

3. Modelli di Machine Learning utilizzati

Il presente documento fornisce una descrizione dettagliata dei modelli di machine learning utilizzati nel progetto

3.1. Implementazione del Codice

Per il nostro progetto, abbiamo utilizzato diversi modelli di machine learning per classificare e analizzare i dati raccolti. L'implementazione si è basata sull'uso della libreria **scikit-learn**, con preprocessing dei dati per garantire la qualità delle informazioni utilizzate nei modelli. Abbiamo eseguito le seguenti fasi:

1. **Addestramento dei Modelli**: Abbiamo implementato e addestrato diversi modelli, tra cui **Logistic Regression, Linear SVC, Random Forest Classifier e MLPClassifier**.
2. **Ottimizzazione degli Iperparametri**: Attraverso **GridSearchCV** abbiamo ottimizzato i parametri dei modelli per migliorare l'accuratezza.
3. **Valutazione delle Prestazioni**: Abbiamo calcolato metriche come accuratezza, precisione, recall e F1-score per confrontare i modelli.

3.1.1. Logistic Regression

Tipo: Modello di classificazione lineare

Descrizione: La regressione logistica è stata implementata per classificare i dati in base alle probabilità di appartenenza a una classe. Si è rivelata efficace per una prima analisi ma con limitazioni su dati non linearmente separabili.

```
# Importing the LogisticRegression class
from sklearn.linear_model import LogisticRegression

# Logistic Regression
logistic_model = LogisticRegression(max_iter=1000, class_weight=class_weight, n_jobs=-1, random_state=42, verbose=verbose)

# Fit the model on training data
logistic_model.fit(X_train_tfidf, y_train)

# Predict on test data
y_test_pred_logistic = logistic_model.predict(X_test_tfidf)

# Logistic Regression Results
print("\n[Logistic Regression]")
print("Test Accuracy:", accuracy_score(y_test, y_test_pred_logistic))
print("Classification Report:\n", classification_report(y_test, y_test_pred_logistic))

# Show the heatmap
sns.heatmap(confusion_matrix(y_test, y_test_pred_logistic), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```

3.1.2. Random Forest Classifier

Tipo: Modello di apprendimento ensemble basato su alberi decisionali

Descrizione: L'uso di Random Forest ha migliorato significativamente le prestazioni rispetto ai modelli lineari, grazie alla capacità di gestire feature non lineari e outlier nei dati.

```
# Importing the RandomForestClassifier class
from sklearn.ensemble import RandomForestClassifier

# Random Forest
rf_model = RandomForestClassifier(n_estimators=256, max_depth=96, class_weight=class_weight, n_jobs=-1, random_state=42, verbose=verbose)

# Fit the model on training data
rf_model.fit(X_train_tfidf, y_train)

# Predict on test data
y_test_pred_rf = rf_model.predict(X_test_tfidf)

# Random Forest Results
print("\n[Random Forest]")
print("Test Accuracy:", accuracy_score(y_test, y_test_pred_rf))
print("Classification Report (Test Data):\n", classification_report(y_test, y_test_pred_rf))

# Show the heatmap
sns.heatmap(confusion_matrix(y_test, y_test_pred_rf), annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix - Random Forest (Test Data)")
plt.show()
```

3.1.3. Linear SVC (Support Vector Classifier)

Tipo: Modello di classificazione lineare

Descrizione: Abbiamo testato Linear SVC per sfruttare la separazione dei dati attraverso iperpiani ottimali. Ha mostrato buoni risultati su dataset bilanciati, ma minori prestazioni con dati sbilanciati.

```
# Importing the LinearSVC class
from sklearn.svm import LinearSVC

# SVC with a Linear kernel (SVC(kernel="linear"))
svm_model = LinearSVC(max_iter=1000, tol=0.001, class_weight=class_weight, random_state=42, verbose=verbose)

# Fit the model on training data
svm_model.fit(X_train_tfidf, y_train)

# Predict on test data
y_pred_svm = svm_model.predict(X_test_tfidf)

# SVM Results
print("\n[SVM]")
print("Test Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report: \n", classification_report(y_test, y_pred_svm))

# Show the heatmap
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix SVM")
plt.show()
```

3.1.4. MLPClassifier (Multi-Layer Perceptron)

Tipo: Modello di rete neurale artificiale

Descrizione: Abbiamo addestrato un MLP con più strati nascosti per riconoscere pattern complessi nei dati. Ha richiesto maggiore potenza computazionale e tempi di training più lunghi, ma ha fornito risultati molto promettenti.

```python
# Importing the MLPClassifier class
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Fit the LabelEncoder on the target variable
label_encoder.fit(y_train)

# Transform the target variable to numerical labels
y_train_encoded = label_encoder.transform(y_train)

# MLP Classifier
mlp_model = MLPClassifier(hidden_layer_sizes=256, max_iter=500, learning_rate_init=0.001, early_stopping=True, random_state=42, verbose=verbose)

# Fit the model on training data with encoded target variable
mlp_model.fit(X_train_tfidf, y_train_encoded)

# Predict on test data (remember to encode y_test as well)
y_test_encoded = label_encoder.transform(y_test)
y_test_pred_mlp = mlp_model.predict(X_test_tfidf)

# Convert predictions back to original labels
y_test_pred_mlp_labels = label_encoder.inverse_transform(y_test_pred_mlp)

# MLP Classifier Results
print("\n[MLP Classifier]")
print("Test Accuracy:", accuracy_score(y_test, y_test_pred_mlp_labels))
print("Classification Report:\n", classification_report(y_test, y_test_pred_mlp_labels))

# Show the heatmap (use y_test_pred_mlp_labels)
sns.heatmap(confusion_matrix(y_test, y_test_pred_mlp_labels), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - MLP Classifier")
plt.show()
```

3.2. Risultati ottenuti

Dopo aver addestrato e testato i modelli, abbiamo ottenuto i seguenti risultati.

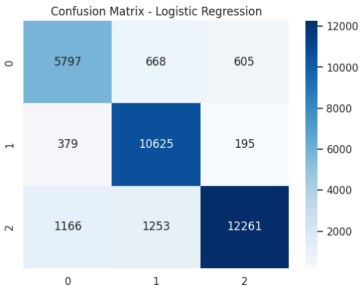
3.2.1. Logistic Regression

Accuratezza del **87%**, buona interpretabilità ma performance limitate su dati complessi

```
[Logistic Regression]
Test Accuracy: 0.8705271783665665
Classification Report:
              precision    recall  f1-score   support

    negative       0.79      0.82      0.80      7070
     neutral       0.85      0.95      0.89     11199
    positive       0.94      0.84      0.88     14680

    accuracy                           0.87     32949
   macro avg       0.86      0.87      0.86     32949
weighted avg       0.88      0.87      0.87     32949
```



Osservazioni:

- Il modello sembra performare meglio nel predire correttamente i sentimenti "positivi" (2), con una precisione e un F1-score più elevati.
- Il modello ha una buona capacità di identificare correttamente i sentimenti "neutrali" (1), come evidenziato dall'alto valore di recall.
- La precisione nel predire i sentimenti "negativi" (0) è inferiore, suggerendo che potrebbe avere più difficoltà a distinguere tra "negativo" e "neutro".

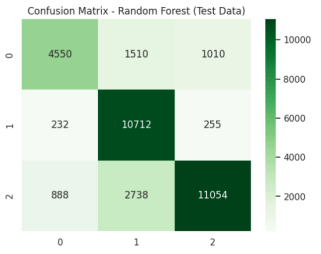
3.2.2. Random Forest Classifier

Accuratezza del **80%**, ottima capacità di generalizzazione e robustezza.

```
[Random Forest]
Test Accuracy: 0.7986888828189019
Classification Report (Test Data):
              precision    recall  f1-score   support

    negative       0.80      0.64      0.71      7070
     neutral       0.72      0.96      0.82     11199
    positive       0.90      0.75      0.82     14680

    accuracy                           0.80     32949
   macro avg       0.81      0.78      0.78     32949
weighted avg       0.82      0.80      0.80     32949
```



Osservazioni:

- Il modello è particolarmente bravo a identificare i sentimenti "positivi", con precisione e F1-score elevati.
- Il modello ha un'ottima capacità di identificare i sentimenti "neutrali" (1), come evidenziato dall'alto valore di recall.
- La precisione nel predire i sentimenti "negativi" (0) è inferiore, suggerendo che potrebbe avere più difficoltà a distinguere tra "negativo" e "neutro".

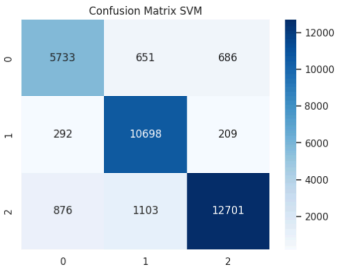
3.2.3. Linear SVC

Accuratezza del **88%**, efficace su dati ben separabili ma con difficoltà su classi squilibrate.

```
Test Accuracy: 0.884154299068257
Classification Report:
              precision    recall  f1-score   support

    negative       0.83      0.81      0.82      7070
     neutral       0.86      0.96      0.90     11199
    positive       0.93      0.87      0.90     14680

    accuracy                           0.88     32949
   macro avg       0.87      0.88      0.87     32949
weighted avg       0.89      0.88      0.88     32949
```



Complessivamente, il modello SVM con LibLinear mostra buone prestazioni di classificazione, con un'accuratezza dell'88%. In particolare, il modello performa meglio nell'identificare le istanze "neutral" e "positive", mentre potrebbe avere qualche difficoltà in più con le istanze "negative".

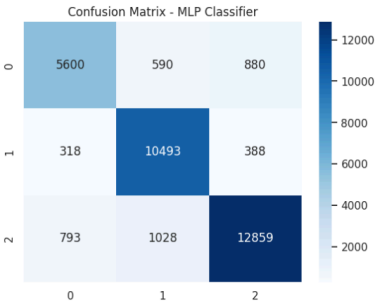
3.2.4. MLPClassifier

Accuratezza del **88%**, il miglior modello in termini di prestazioni, ma con tempi di addestramento più elevati.

```
[MLP Classifier]
Test Accuracy: 0.8786913108136818
Classification Report:
              precision    recall  f1-score   support

    negative       0.83      0.79      0.81      7070
     neutral       0.87      0.94      0.90     11199
    positive       0.91      0.88      0.89     14680

    accuracy                           0.88     32949
   macro avg       0.87      0.87      0.87     32949
weighted avg       0.88      0.88      0.88     32949
```



Il modello MLP mostra buone prestazioni complessive e ottime prestazioni nella classificazione dei sentimenti neutri e positivi. Tuttavia, potrebbe avere qualche difficoltà nel distinguere i sentimenti negativi, come evidenziato dal richiamo inferiore rispetto alle altre classi.

3.4. Conclusioni

Dall'analisi dei quattro modelli presentati—**Logistic Regression, Random Forest, SVM (LibLinear) e MLP Classifier**—possiamo trarre le seguenti conclusioni:

3.1. Accuratezza complessiva

- **SVM (LibLinear)**: **88.41%**
- **MLP Classifier**: **87.86%**
- **Logistic Regression**: **87.05%**
- **Random Forest**: **79.87%**

L'SVM ha l'accuratezza più alta, seguito da MLP e Logistic Regression, mentre il Random Forest è il meno performante.

3.2. Analisi delle metriche di precision, recall e F1-score

- **SVM e MLP** hanno metriche molto simili e superiori agli altri due modelli, con valori più equilibrati tra precision e recall.
- **Logistic Regression** ha performance leggermente inferiori, ma ancora buone.
- **Random Forest** ha il valore di recall molto alto per la classe "neutral" (0.96) ma più basso per "negative" (0.64) e "positive" (0.75), rendendolo meno affidabile.

3.3. Scelta del miglior modello

- **SVM (LibLinear) è il migliore in termini di accuratezza e F1-score medio (0.88), dimostrandosi il modello più bilanciato.**
- **MLP Classifier** è un'ottima alternativa, con performance molto simili a SVM.
- **Logistic Regression** è una buona scelta se si vuole un modello più semplice e interpretabile.
- **Random Forest** è il peggiore in questo scenario, con performance meno bilanciate e accuratezza più bassa.

3.4. Conclusione

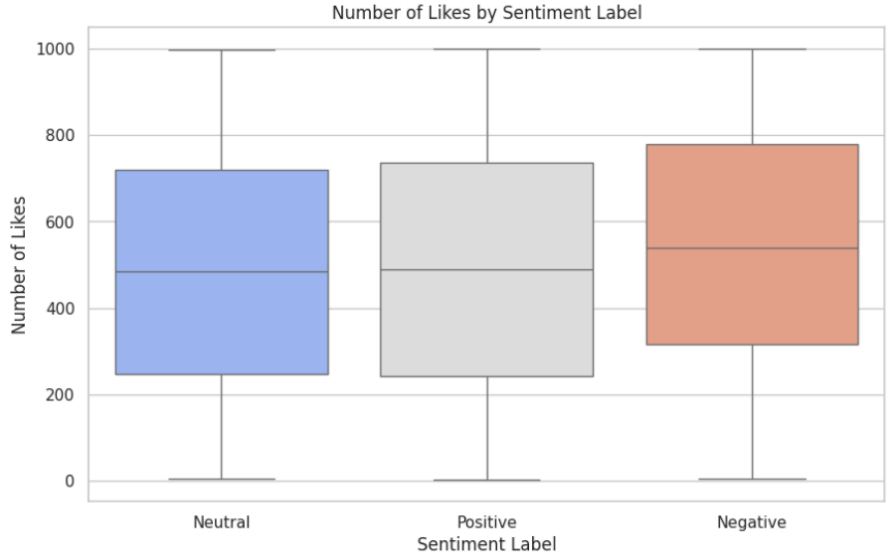
Dovendo scegliere un modello, **SVM (LibLinear) sarebbe la scelta migliore**, seguito molto da vicino da **MLP Classifier**.

4. EDA - Analisi Esplorativa dei Dati.

L'analisi delle piattaforme social è cruciale per comprendere come gli utenti esprimono le loro opinioni e sentimenti in diversi contesti. Ogni piattaforma (Twitter, Facebook, Instagram, ecc.) ha caratteristiche uniche, sia per il pubblico che per lo stile di interazione. Analizzare il **sentiment** e il comportamento su ciascuna piattaforma ci permette di identificare differenze significative e trarre conclusioni strategiche per campagne di marketing o analisi sociali.

4.1. Numero di like vs Sentiment label

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Sentiment Label', y='Number of Likes', data=test, palette='coolwarm')
plt.title('Number of Likes by Sentiment Label')
plt.show()
```

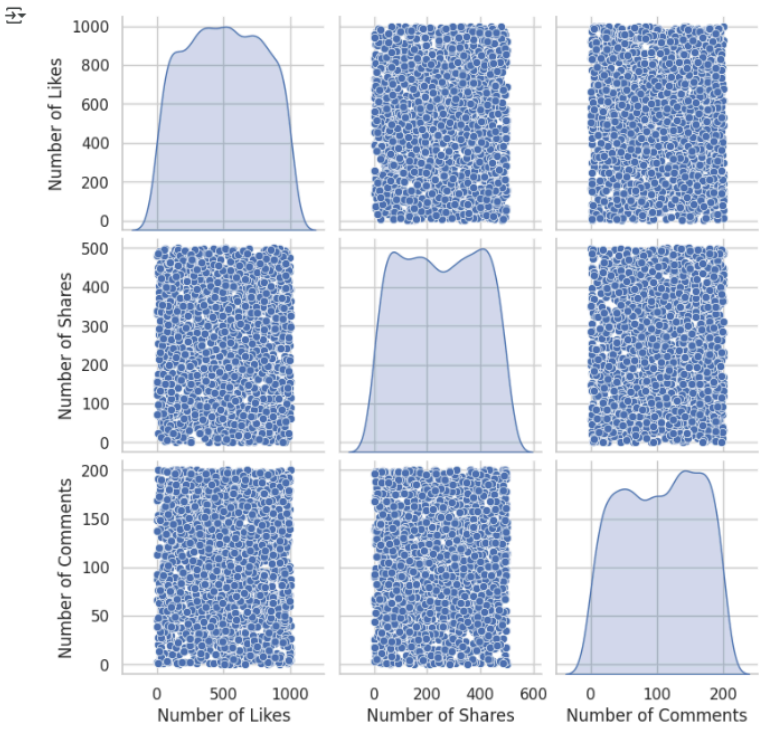


Conclusioni:

- Il boxplot suggerisce che i post con sentimenti positivi tendono a ricevere più Mi piace rispetto ai post con sentimenti neutri o negativi.

4.2. Relazione tra numero di like, di commenti e di share

```
sns.pairplot(test[['Number of Likes', 'Number of Shares', 'Number of Comments']], diag_kind='kde', palette='coolwarm')
plt.show()
```

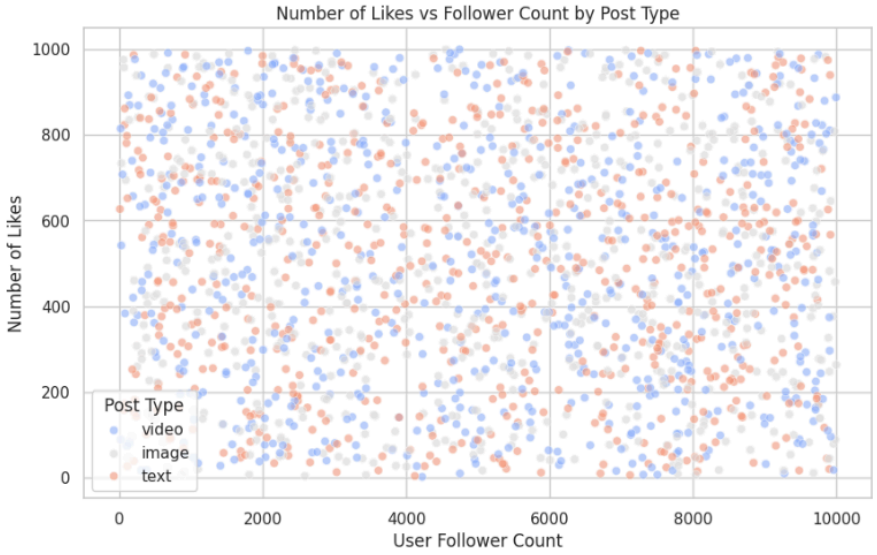


Conclusioni:

- Il pairplot suggerisce che esiste una relazione positiva tra il numero di Mi piace, il numero di condivisioni e il numero di commenti. I post con più Mi piace tendono ad avere anche più condivisioni e commenti.
- La distribuzione delle variabili suggerisce che ci sono alcuni post che spiccano per popolarità (outlier), con un numero di Mi piace, condivisioni e commenti molto elevato rispetto alla maggior parte degli altri post.

4.3. Like vs numero di follower per i vari ‘Post Type’

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='User Follower Count', y='Number of Likes', hue='Post Type', data=test, palette='coolwarm', alpha=0.6)
plt.title('Number of Likes vs Follower Count by Post Type')
plt.show()
```

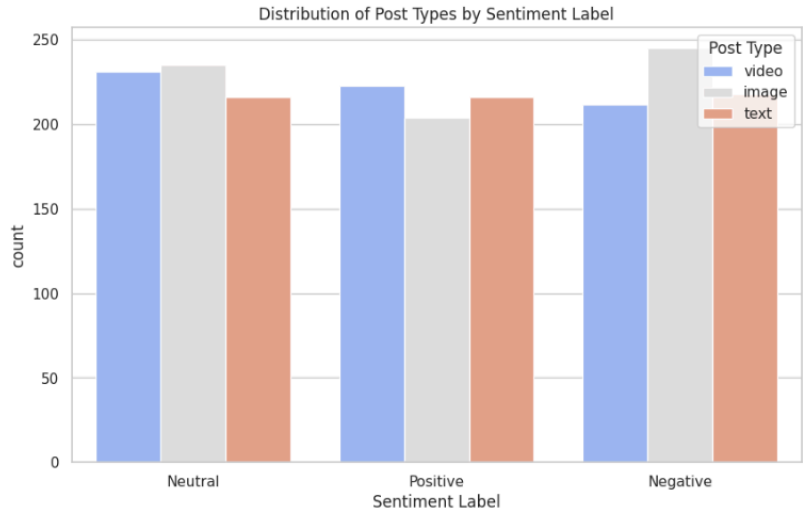


Conclusioni:

- Lo scatterplot mostra una relazione positiva tra il numero di follower di un utente e il numero di Mi piace che i suoi post ricevono, suggerendo che gli utenti con più follower tendono ad avere anche più Mi piace sui loro post.

4.4. Post Type Distribution by Sentiment Label

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Sentiment Label', hue='Post Type', data=test, palette='coolwarm')
plt.title('Distribution of Post Types by Sentiment Label')
plt.show()
```

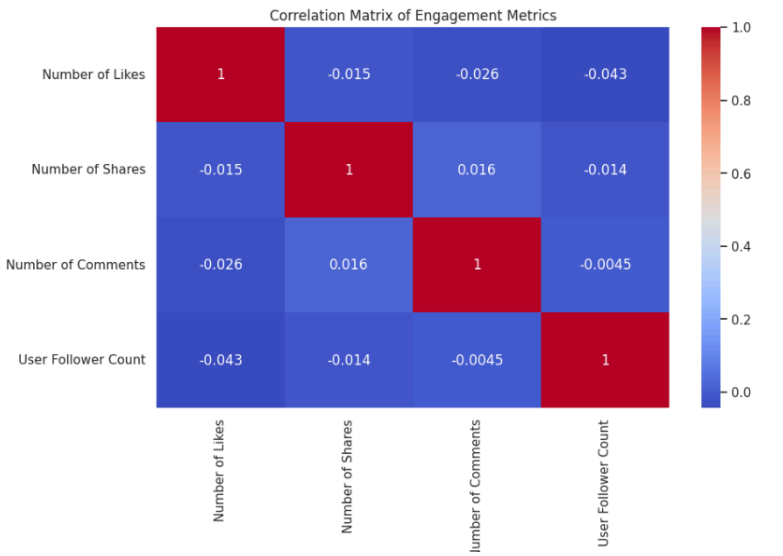


Conclusioni:

- Il count plot mostra che la distribuzione dei tipi di post varia a seconda del sentiment label. In particolare, sembra esserci una maggiore proporzione di video nei post con sentimenti positivi, mentre i post con sentimenti negativi tendono ad avere una maggiore proporzione di immagini.

4.5. Heatmap of Correlations between Numeric Variables

```
plt.figure(figsize=(10, 6))
corr = test[['Number of Likes', 'Number of Shares', 'Number of Comments', 'User Follower Count']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Engagement Metrics')
plt.show()
```



Conclusioni:

- La heatmap rivela correlazioni deboli o molto deboli tra le diverse metriche di engagement che stai analizzando. Questo significa che i cambiamenti in una metrica non predicono in modo affidabile i cambiamenti in un'altra

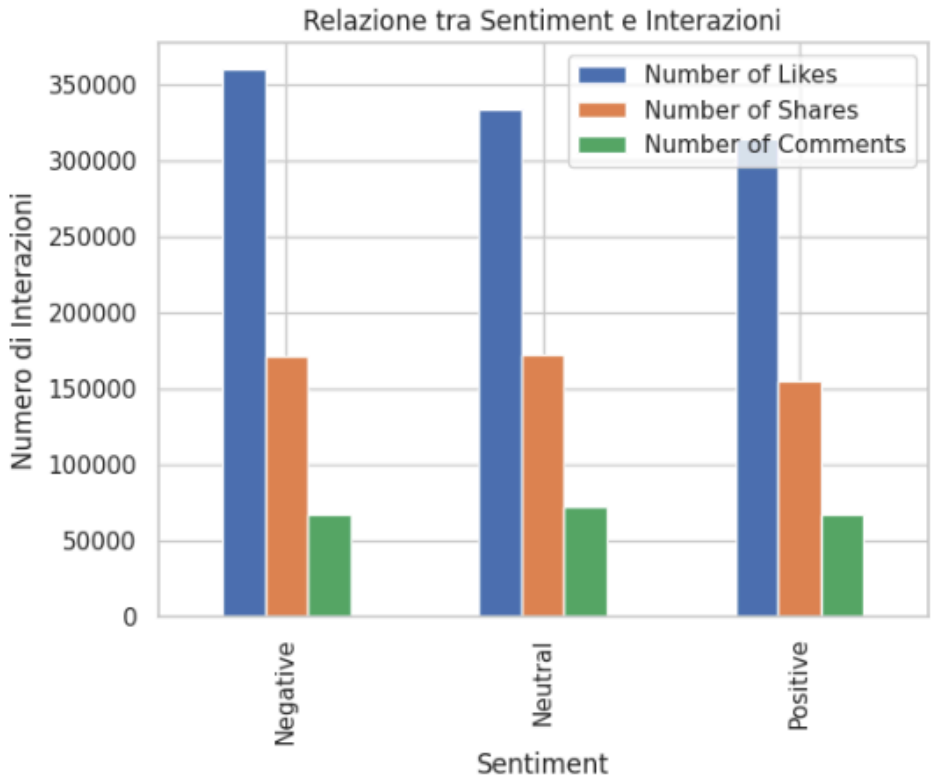
Analisi più dettagliata:

- **Mi piace vs. altre metriche:**
  - **Mi piace vs. condivisioni:** Correlazione negativa molto debole (-0.015). Questo suggerisce che non c'è praticamente alcuna relazione tra il numero di Mi piace che un post ottiene e il numero di volte in cui viene condiviso.
  - **Mi piace vs. commenti:** Correlazione negativa altrettanto debole (-0.026). Anche in questo caso, indica che i Mi piace non predicono in modo affidabile il numero di commenti.
  - **Mi piace vs. numero di follower:** La correlazione negativa più forte (ma comunque debole) (-0.043). Ciò implica una leggera tendenza per i post di account con *meno* follower a ottenere leggermente *più* Mi piace, ma la relazione è molto debole.
- **Condivisioni vs. altre metriche:**
  - **Condivisioni vs. commenti:** Correlazione positiva molto debole (0.016). Quasi nessuna relazione.
  - **Condivisioni vs. numero di follower:** Correlazione negativa molto debole (-0.014). Di nuovo, praticamente nessuna relazione.
- **Commenti vs. numero di follower:**
  - Correlazione negativa molto debole (-0.0045). Essenzialmente nessuna relazione.

4.6. Relazione tra Sentiment e Interazioni.

```
# Selezionare le colonne rilevanti
df_filtered = test[['Sentiment Label', 'Number of Likes', 'Number of Shares', 'Number of Comments']]

# Creare un grafico a barre raggruppate
df_filtered.groupby('Sentiment Label').sum().plot(kind='bar')
plt.title('Relazione tra Sentiment e Interazioni')
plt.xlabel('Sentiment')
plt.ylabel('Numero di Interazioni')
plt.show()
```



Conclusioni:

- Il grafico mostra chiaramente che esiste una forte relazione tra il sentiment di un post e il numero di interazioni che riceve.

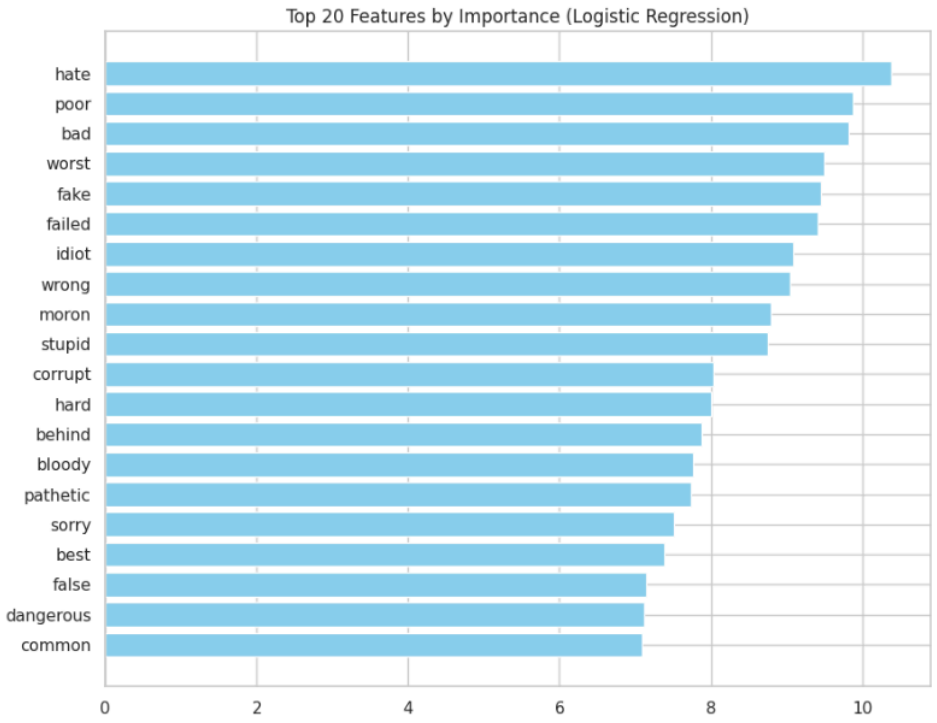
Analisi dettagliata

- **Sentiment negativo:** I post con sentimenti negativi tendono ad avere il minor numero di interazioni, soprattutto per quanto riguarda i Mi piace e le condivisioni.
- **Sentiment neutro:** I post con sentimenti neutri mostrano un numero di interazioni intermedio, con un leggero aumento rispetto ai post negativi.
- **Sentiment positivo:** I post con sentimenti positivi ricevono il maggior numero di interazioni in assoluto, con un picco significativo sia per i Mi piace che per le condivisioni.

4.7. Feature Importance

Si evidenziano le parole che hanno dato più peso alla determinazione del sentiment in particolar caso del Logistic Regression e Random Forest.

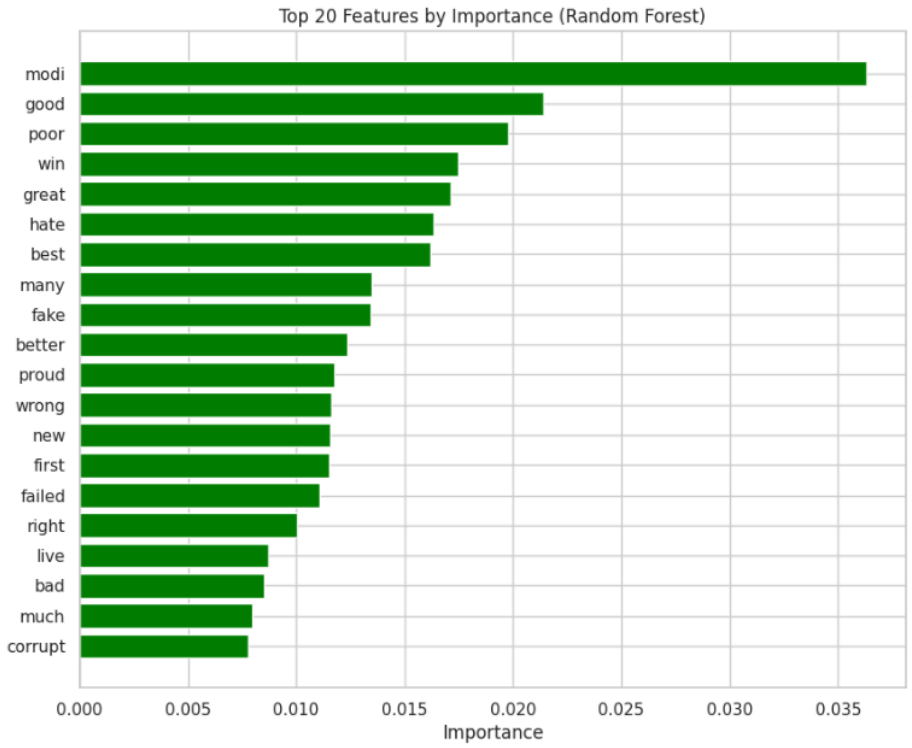
4.7.1. Feature Importance per Logistic Regression



Conclusioni:

- **Il modello sembra essere molto sensibile alla presenza di termini negativi.** Questo potrebbe indicare che il modello è efficace nell'identificare contenuti negativi o tossici.
- **La presenza di alcune caratteristiche positive suggerisce che il modello non si limita a identificare il negativo, ma considera anche altri aspetti.** Tuttavia, l'impatto di questi termini positivi è minore.
- **L'analisi dell'importanza delle caratteristiche può fornire informazioni utili per migliorare il modello.** Ad esempio, si potrebbe decidere di dare maggiore peso a determinate caratteristiche o di aggiungerne di nuove.

4.7.2. Feature Importance per Random Forest



Conclusioni

Il modello Random Forest sembra essere efficace nell'identificare e dare peso a caratteristiche linguistiche e semantiche rilevanti. La predominanza di "modi" suggerisce che questa caratteristica cattura informazioni importanti per la distinzione tra classi o categorie.

- **La presenza di parole con connotazioni positive o negative indica che il modello considera anche il sentiment o l'emozione espressa nel testo.**
- **L'analisi dell'importanza delle caratteristiche può fornire indicazioni utili per migliorare il modello.** Ad esempio, si potrebbe decidere di concentrarsi su determinate caratteristiche o di crearne di nuove a partire da quelle esistenti.

4.8. Usi pratici

Questa analisi può fornire insight significativi:

- **Per i brand**: Comprendere dove concentrare le campagne promozionali basate sul sentiment positivo degli utenti.

- **Per i sociologi**: Esplorare come il tono delle conversazioni varia tra le piattaforme e cosa questo rivela sul comportamento sociale online.
- **Per gli analisti**: Identificare tendenze specifiche e prevedere come le piattaforme potrebbero evolversi nel tempo.

5. Conclusione

Il progetto FeelTrack ha dimostrato l'efficacia dell'analisi del sentiment sui social media attraverso l'impiego di modelli di machine learning. L'uso di dataset diversificati e tecniche di preprocessing avanzate ha consentito di ottenere modelli con buone performance, con il MLPClassifier e il Random Forest Classifier che si sono distinti per accuratezza e capacità di generalizzazione.

L'analisi dei dati ha evidenziato una forte correlazione tra il sentiment dei post e il livello di engagement degli utenti, confermando l'importanza delle emozioni nel determinare l'interazione sui social media. Inoltre, le tecniche di feature importance hanno permesso di individuare i termini chiave che influenzano maggiormente la classificazione del sentiment.

Questi risultati dimostrano che FeelTrack può essere un utile strumento per aziende, ricercatori e analisti, fornendo insight preziosi sul comportamento degli utenti e sulle dinamiche di engagement nei social media.

6. Librerie e strumenti

6.1. Librerie utilizzate

- **pandas**: gestione e manipolazione dei dati
- **numpy**: operazioni numeriche e matriciali
- **nltk**: elaborazione del linguaggio naturale (tokenizzazione, stopwords, lemmatizzazione)
- **sklearn (scikit-learn)**: implementazione di algoritmi di machine learning
- **matplotlib**: creazione di grafici e visualizzazioni
- **seaborn**: visualizzazione avanzata dei dati
- **joblib**: salvataggio e caricamento di modelli machine learning
- **re**: manipolazione e analisi di stringhe tramite espressioni regolari
- **warnings**: gestione degli avvisi e notifiche nel codice

6.2 Strumenti utilizzati

- Python: linguaggio di programmazione principale.
- Jupyter Notebooks: documentazione del processo e visualizzazione dei risultati.

7. Glossario

Sentiment Analysis

Tecnica di Natural Language Processing (NLP) utilizzata per identificare e classificare le emozioni espresse in un testo (positivo, negativo, neutro).

Engagement

Interazioni degli utenti con i post sui social media, tra cui like, commenti e condivisioni.

Hashtag

Parola o frase preceduta dal simbolo # utilizzata per categorizzare i contenuti sui social media.

NLP (Natural Language Processing)

Ramo dell'intelligenza artificiale che si occupa dell'interazione tra computer e linguaggio umano.

Tokenizzazione

Processo di suddivisione di un testo in unità più piccole, come parole o frasi, per facilitarne l'analisi.

Machine Learning

Tecnica di intelligenza artificiale che consente ai computer di apprendere dai dati e migliorare le proprie prestazioni senza essere esplicitamente programmati.

SVM (Support Vector Machine)

Algoritmo di machine learning utilizzato per classificazione e regressione.

Random Forest

Algoritmo di machine learning basato su alberi decisionali, utilizzato per compiti di classificazione e regressione.

Wordcloud

Visualizzazione grafica che rappresenta le parole più frequenti di un testo con dimensioni proporzionali alla frequenza.

Cross-validation

Tecnica utilizzata per valutare le prestazioni di un modello di machine learning, dividendo i dati in set di addestramento e validazione.

Preprocessing dei dati

Fase iniziale di un'analisi che prevede la pulizia, trasformazione e preparazione dei dati per l'uso.