

```

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler

import pandas as pd

# Load the dataset
df = pd.read_csv("/content/sample_data/student-dataset.csv") # Replace with your file
print(df.head())

↗ school;sex;age;address;famsize;Pstatus;Medu;Fedu;Mjob;Fjob;reason;guardian;trave
0 GP;"F";18;"U";"GT3";"A";4;4;"at_home";"teacher...
1 GP;"F";17;"U";"GT3";"T";1;1;"at_home";"other";...
2 GP;"F";15;"U";"LE3";"T";1;1;"at_home";"other";...
3 GP;"F";15;"U";"GT3";"T";4;2;"health";"services...
4 GP;"F";16;"U";"GT3";"T";3;3;"other";"other";"h...

```

Start coding or [generate](#) with AI.

```
print(df.info())
```

```

↗ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 1 columns):
#   Column
---  ---
0   school;sex;age;address;famsize;Pstatus;Medu;Fedu;Mjob;Fjob;reason;guardian;tr
dtypes: object(1)
memory usage: 3.2+ KB
None

```

```

# Not applicable unless you have extra categorical columns.
# Example:
# df['source'] = LabelEncoder().fit_transform(df['source'])

```

Start coding or [generate](#) with AI.

```

# Only needed if you have additional categorical fields.
# Example:
# df = pd.get_dummies(df, columns=['category'], drop_first=True)

```

```
from sklearn.metrics import accuracy_score, classification_report
```

```

def predict_fake_news(text):
    vector = tfidf.transform([text]).toarray()
    scaled = scaler.transform(vector)
    pred = model.predict(scaled)[0][0]
    return "Fake News" if pred > 0.5 else "Real News"

def encode_text_input(text):
    df_input = pd.DataFrame({'text': [text]})
    tfidf_vector = tfidf.transform(df_input['text']).toarray()
    scaled_vector = scaler.transform(tfidf_vector)
    return scaled_vector

```

```

def final_grade(text):
    result = predict_fake_news(text)
    return f"The article is classified as: {result}"

```

```
from sklearn.metrics import accuracy_score, classification_report
```

[Explain code](#)
[KeyError](#)
[Explain code](#) × ...

Please explain [this code](#).

✦ Okay, let's break down this part of the code:

Splitting Data for Training and Testing

This section of code is crucial for evaluating how well a machine learning model will perform on unseen data. It uses a function called `train_test_split` from the `sklearn.model_selection` module.

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

```

[Use code with caution](#)



Here's a breakdown:

- 1. `from sklearn.model_selection import train_test_split`**: This line imports the `train_test_split` function, which is specifically designed for splitting data into training and testing sets.
- 2. `X_train, X_test, y_train, y_test = train_test_split(...)`**: This line is where the actual splitting happens. The function is called with some important parameters, and it returns four objects:
 - `X_train`**: This will contain the features (input data) used to **train** the machine learning model.
 - `X_test`**: This will hold the features that are held back and used to **test** the trained model's performance.
 - `y_train`**: This contains the corresponding **target values** (what you're trying to predict) for the training data.
 - `y_test`**: This contains the target values for the testing data, used to evaluate the model's predictions.
- 3. `X_scaled, y`**: These are the inputs to the `train_test_split` function:
 - `X_scaled`**: This likely represents your feature data after it has been scaled using the `StandardScaler`. Scaling is often important to ensure that features with different ranges don't disproportionately influence the model.
 - `y`**: This represents your target variable—what you're trying to predict. For example, in a fake news detection system, this would be a variable indicating whether an article is real or fake.
- 4. `test_size=0.2`**: This parameter controls the proportion of the data that is allocated to the testing set. In this case, 20% (0.2) of the data will be used for testing, and the remaining 80% will be used for training.
- 5. `random_state=42`**: This ensures that the data is split in the same way every time the code is run.

This is important for reproducibility so you can compare results across different runs consistently.

In simpler terms, imagine you have a deck of cards and want to teach someone a card game. You'd first split the deck into two piles: one for practice (training) and one to see how well they learned (testing). `train_test_split` is doing something similar with your data. It's creating two sets of data to train and test the accuracy of your machine-learning model.

► Sources

Rate this answer  

Enter a prompt here 

0 / 2000

Gemini can make mistakes so double-check responses and use code with caution. [Learn more](#)