Automatic Essay Scoring Model

Loading necessary libraries

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

Loading Data

```
file_path = "/content/training_set_rel3.xls"
data = pd.read_excel(file_path)
```

Data Pre-Processing

```
columns_to_keep = ['essay_id', 'essay_set', 'essay', 'domain1_score']
data = data[columns_to_keep]

data = data.dropna(subset=['essay', 'domain1_score'])
```

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\W', ' ', text)
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)

data['processed_essay'] = data['essay'].apply(preprocess_text)

data['essay_length'] = data['essay'].apply(lambda x: len(x.split()))
```

Feature Extraction Using TF-IDF

```
# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=5000)
X = vectorizer.fit_transform(data['processed_essay'])
```

Preparing the Target Variable

```
# Target variable
scaler = MinMaxScaler()
y = scaler.fit_transform(data[['domain1_score']])
```

Train-Test Split

```
# Train-test split
X_train, X_test, y_train, y_test, train_indices, test_indices = train_test_split(X, y, range(len(data)), test_size=0.2, random_state=42)
```

Model Training with Random Forest

```
# Model training
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Predictions and Evaluation

```
# Predictions
y_pred = model.predict(X_test)
```

Inverse scaling to get actual scores

```
y_pred_inverse = scaler.inverse_transform(y_pred.reshape(-1, 1))
```

Model Evaluation

```
# Evaluation
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MSE: {mse}, MAE: {mae}, R^2 Score: {r2}')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was ex
  return fit_method(estimator, *args, **kwargs)
MSE: 0.0020016455070241018, MAE: 0.0210752482451635, R^2 Score: 0.9134204638753711
```

Displaying the Results

```
predictions_df = pd.DataFrame({
    'essay_id': data.iloc[test_indices]['essay_id'].values,
    'predicted_score': y_pred_inverse.flatten()
})

print(predictions_df)
```

```
      essay_id  predicted_score
0         7398             1.56
1        13472             2.69
2         7004             1.85
3         6790             1.94
4         4599             3.12
...        ...              ...
2591      9600             2.36
2592     15200             3.07
2593      6256             1.11
2594      4509             2.79
```

```
2595        80             9.79

[2596 rows x 2 columns]
```

```
# Interpret the accuracy
if r2 < 0:
    print("The model does not explain any of the variability in the target variable.")
elif r2 < 0.5:
    print("The model has a weak fit.")
elif r2 < 0.75:
    print("The model has a moderate fit.")
else:
    print("The model has a strong fit.")
```

```
The model has a strong fit.
```