Machine Learning Project

Legal Outcome Predictor: Analyzing Court Cases with Machine Learning

Sourav Kumar Singh

Legal Outcome Predictor: Analyzing Court Cases with Machine Learning.

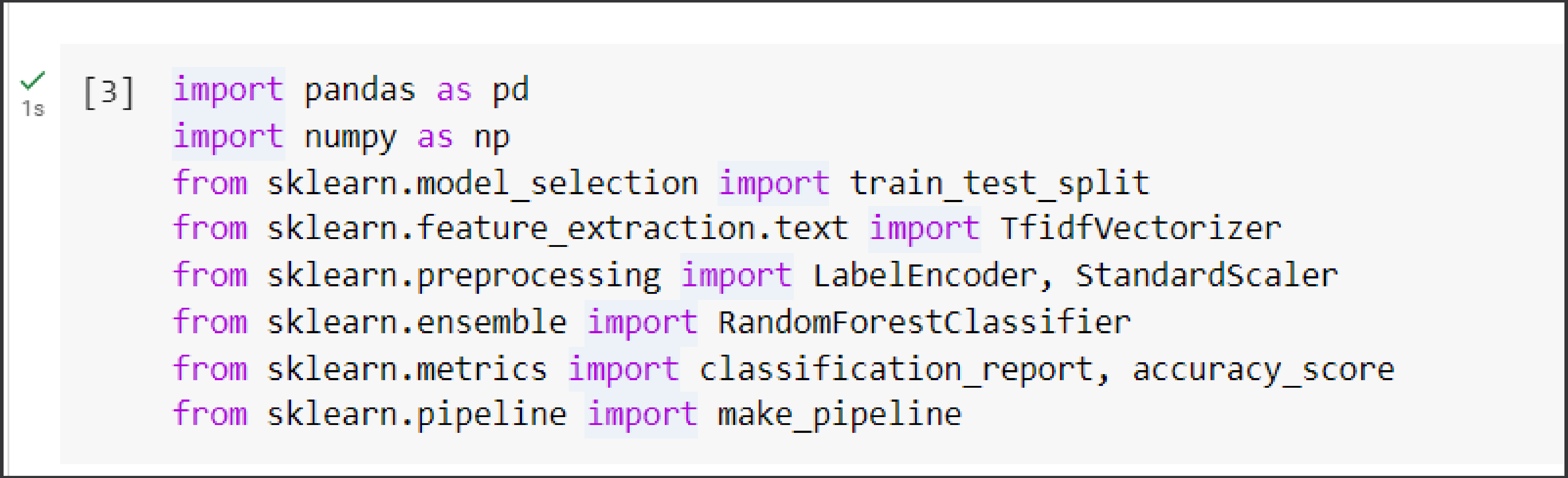
Introduction:

This project aims to predict the outcomes of legal cases using various features extracted from case data. The code demonstrates how to preprocess and analyze a dataset containing information about legal cases, including facts of the case, issue areas, decision types, and dispositions. It involves steps such as reading the data, encoding categorical variables, handling missing values, and preparing the data for model training.

The project employs a Random-Forest-Classifier, a powerful machine learning model, to predict whether the first party or the second party wins in each case. Additionally, it includes text processing with TF-IDF vectorization for the 'facts' column of the dataset and evaluates the model's performance using accuracy metrics and feature importance analysis.

This project showcases how machine learning can be applied in the legal domain to assist in understanding and predicting case outcomes.

Important Libraries to import:

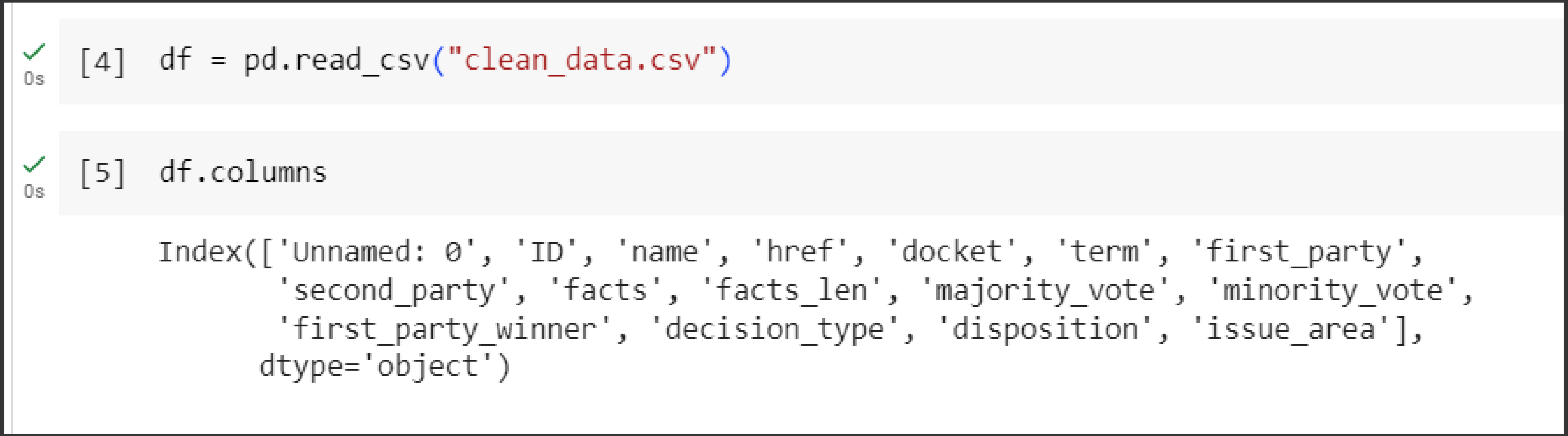


```
[3] import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.preprocessing import LabelEncoder, StandardScaler
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, accuracy_score
    from sklearn.pipeline import make_pipeline
```

Implementing Code:

Reading Data:

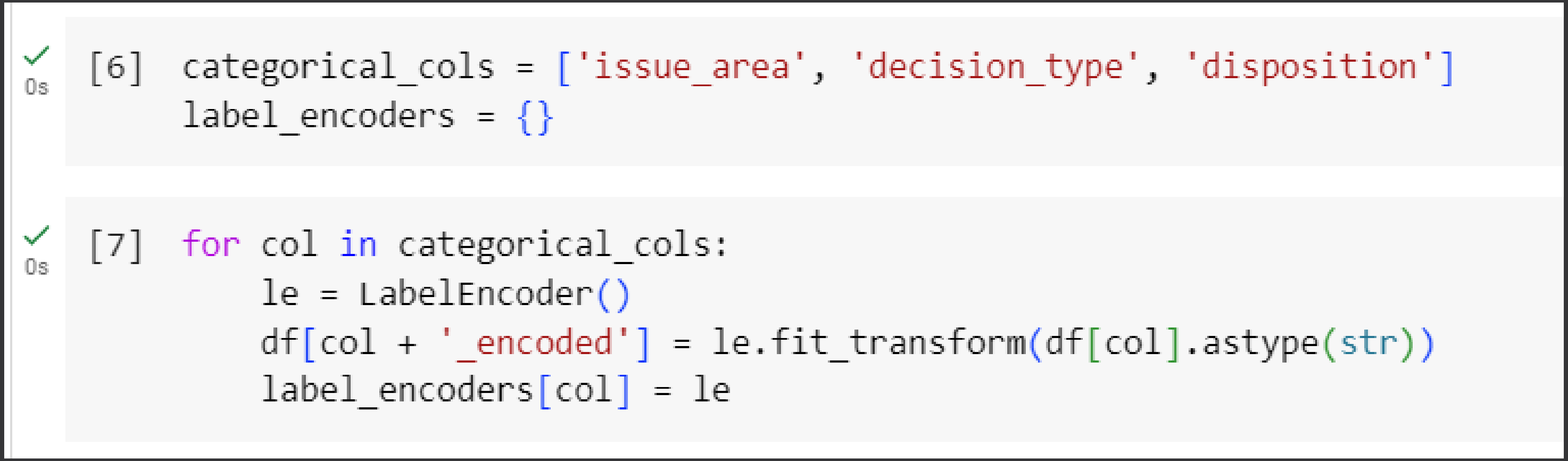
- The code starts by reading a CSV file named "clean_data.csv" into a Data Frame ‘**df**’. This file likely contains data about legal cases, including facts, issues, decision types, dispositions, and information about which party won.



```
0s [4]  df = pd.read_csv("clean_data.csv")

0s [5]  df.columns

        Index(['Unnamed: 0', 'ID', 'name', 'href', 'docket', 'term', 'first_party',
               'second_party', 'facts', 'facts_len', 'majority_vote', 'minority_vote',
               'first_party_winner', 'decision_type', 'disposition', 'issue_area'],
              dtype='object')
```

Encoding Categorical Data:

•The code identifies three categorical columns:
'issue_area', 'decision_type', and 'disposition’.
• We have used **LabelEncoder** for each of these columns.
•This process converts the categorical text data into numerical form, which is easier for machine learning algorithms to process.

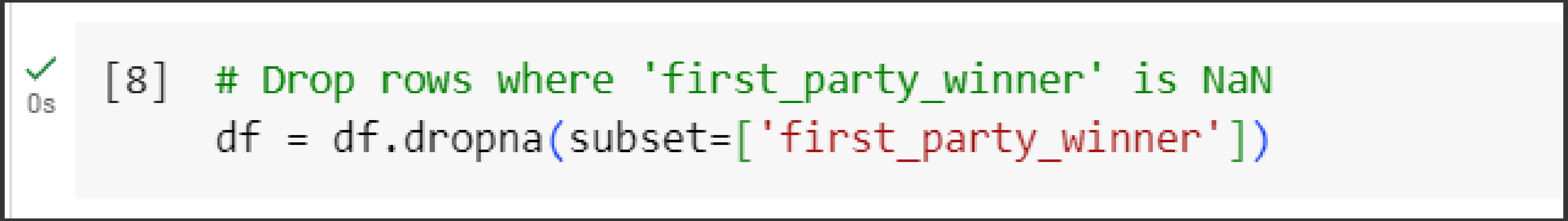


```
[6] categorical_cols = ['issue_area', 'decision_type', 'disposition']
    label_encoders = {}
```

```
[7] for col in categorical_cols:
        le = LabelEncoder()
        df[col + '_encoded'] = le.fit_transform(df[col].astype(str))
        label_encoders[col] = le
```

Data Cleaning:

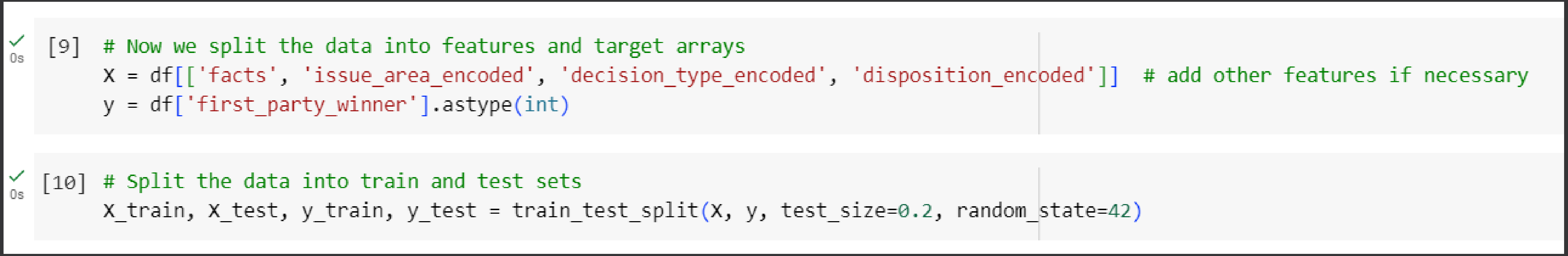
- The code removes rows from ‘**df’** where the 'first_party_winner' column has missing values, ensuring that the analysis only includes complete cases.



```
[8]  # Drop rows where 'first_party_winner' is NaN
     df = df.dropna(subset=['first_party_winner'])
```

Preparing Data for Model:

- The code selects certain columns to be features (**X**) and 'first_party_winner' as the target variable (**y**).
- The features include the text from the 'facts' column and the encoded versions of 'issue_area', 'decision_type', and 'disposition'.

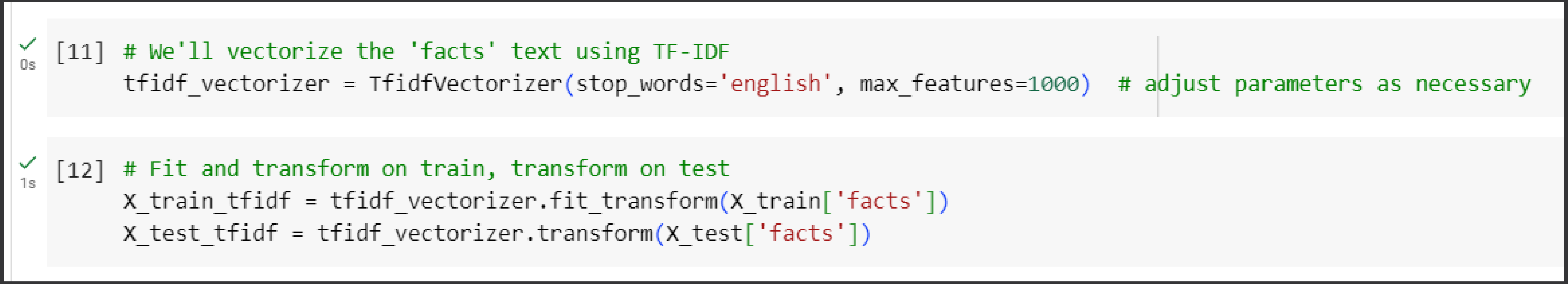


```
[9]  # Now we split the data into features and target arrays
     X = df[['facts', 'issue_area_encoded', 'decision_type_encoded', 'disposition_encoded']]  # add other features if necessary
     y = df['first_party_winner'].astype(int)

[10] # Split the data into train and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Text Vectorization:

- A **TfidfVectorizer** is used to convert the 'facts' text data into a numerical form using the TF-IDF technique. This technique evaluates how important a word is to a document in a collection of documents.

- The TF-IDF values for the training set are calculated using **fit_transform**, and for the test set using **transform**.

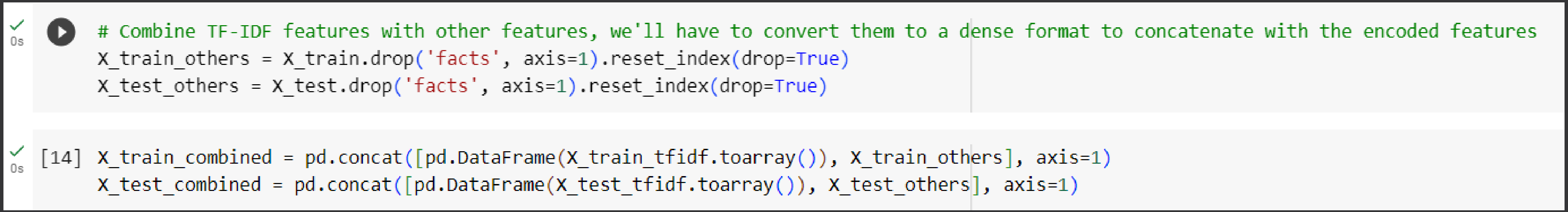


```
[11] # We'll vectorize the 'facts' text using TF-IDF
     tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)  # adjust parameters as necessary
```

```
[12] # Fit and transform on train, transform on test
     X_train_tfidf = tfidf_vectorizer.fit_transform(X_train['facts'])
     X_test_tfidf = tfidf_vectorizer.transform(X_test['facts'])
```

Combining Data:

- The non-text features are separated from the 'facts' column and the indices are reset.
- The TF-IDF data and the other features are combined to form the final training and test datasets.

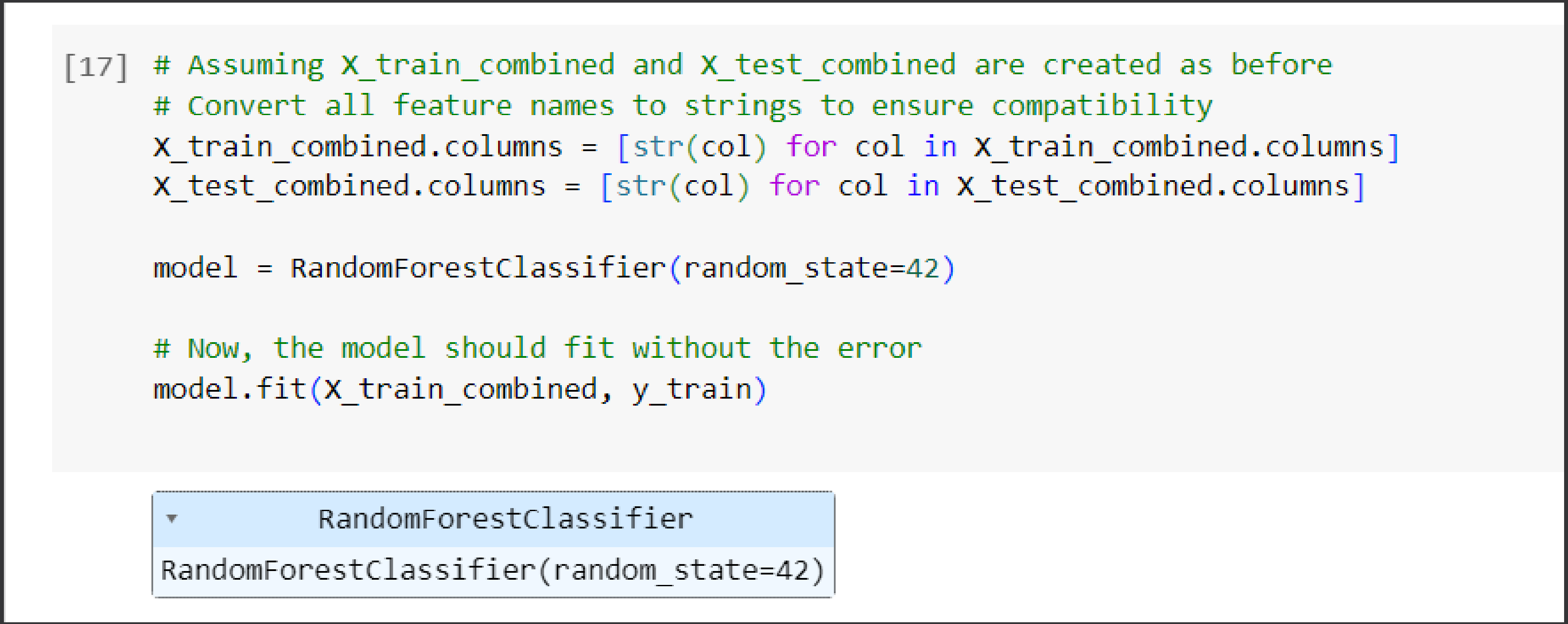


```
# Combine TF-IDF features with other features, we'll have to convert them to a dense format to concatenate with the encoded features
X_train_others = X_train.drop('facts', axis=1).reset_index(drop=True)
X_test_others = X_test.drop('facts', axis=1).reset_index(drop=True)
```

```
[14] X_train_combined = pd.concat([pd.DataFrame(X_train_tfidf.toarray()), X_train_others], axis=1)
     X_test_combined = pd.concat([pd.DataFrame(X_test_tfidf.toarray()), X_test_others], axis=1)
```

Model Training:

- A **RandomForestClassifier** model is initialized and trained using the combined training data.

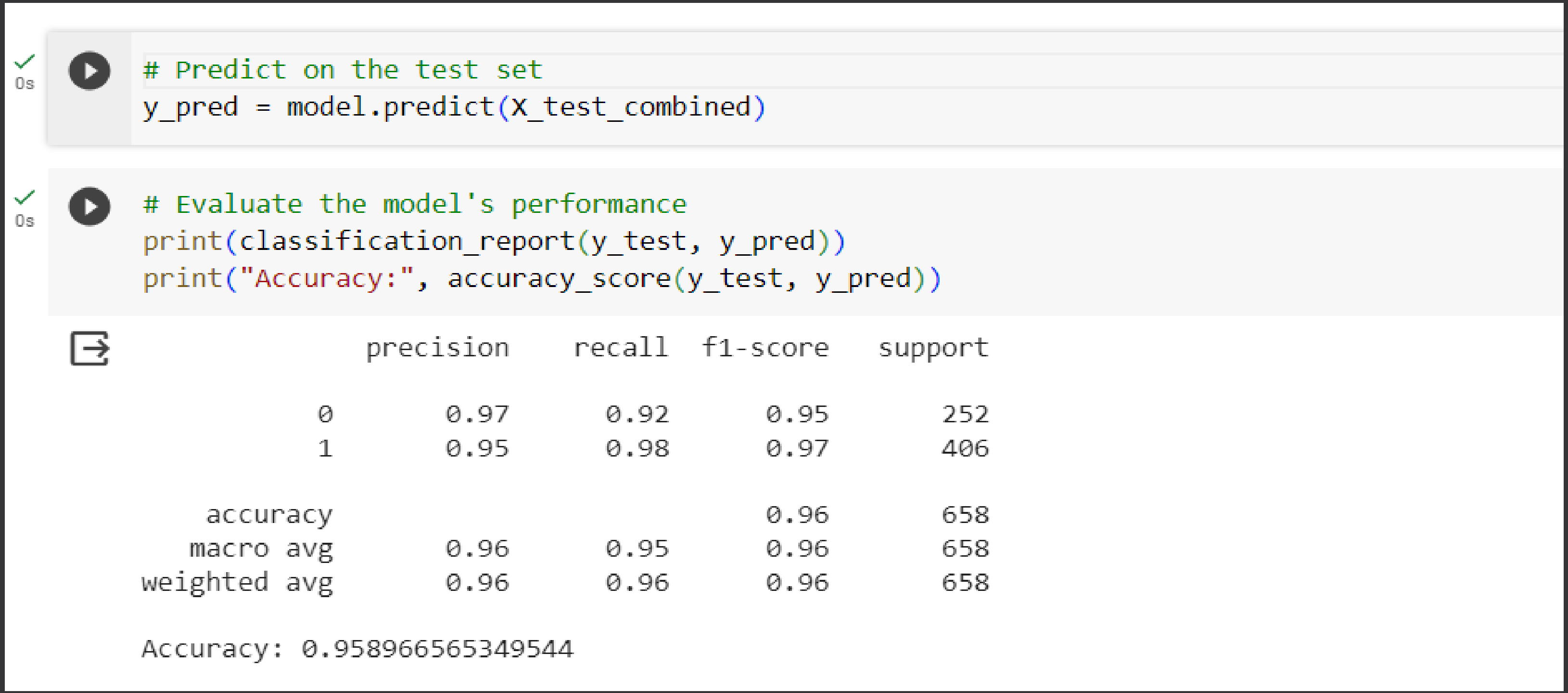


```
[17] # Assuming X_train_combined and X_test_combined are created as before
     # Convert all feature names to strings to ensure compatibility
     X_train_combined.columns = [str(col) for col in X_train_combined.columns]
     X_test_combined.columns = [str(col) for col in X_test_combined.columns]

     model = RandomForestClassifier(random_state=42)

     # Now, the model should fit without the error
     model.fit(X_train_combined, y_train)
```

```
▾          RandomForestClassifier
     RandomForestClassifier(random_state=42)
```

Model Evaluation:

- The trained model is used to predict outcomes on the test set.
- The performance of the model is evaluated using a classification report and accuracy score.



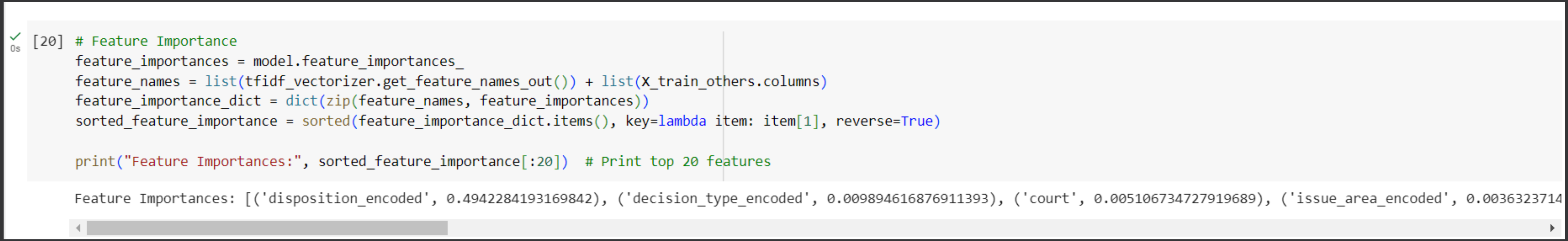
```
# Predict on the test set
y_pred = model.predict(X_test_combined)
```

```
# Evaluate the model's performance
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.92      0.95       252
           1       0.95      0.98      0.97       406

    accuracy                           0.96       658
   macro avg       0.96      0.95      0.96       658
weighted avg       0.96      0.96      0.96       658

Accuracy: 0.958966565349544
```

Feature Importance:

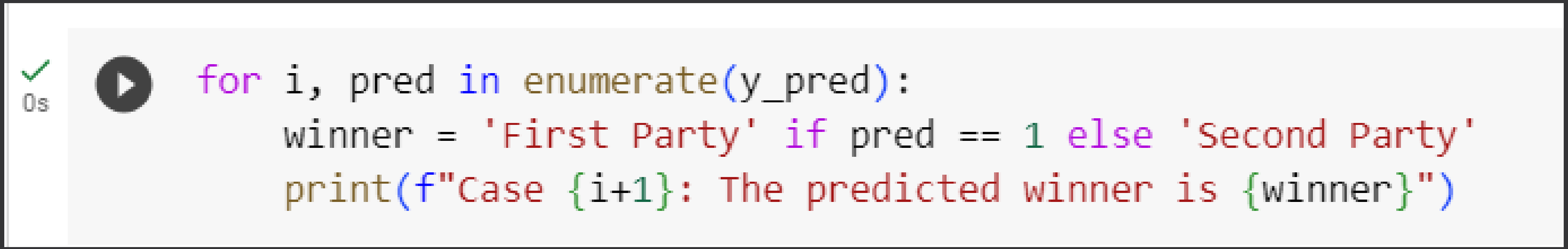
- The importance of each feature in the model is extracted and displayed. This shows which features have the most influence on the model's predictions.



```
[20] # Feature Importance
     feature_importances = model.feature_importances_
     feature_names = list(tfidf_vectorizer.get_feature_names_out()) + list(X_train_others.columns)
     feature_importance_dict = dict(zip(feature_names, feature_importances))
     sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda item: item[1], reverse=True)

     print("Feature Importances:", sorted_feature_importance[:20])  # Print top 20 features

     Feature Importances: [('disposition_encoded', 0.4942284193169842), ('decision_type_encoded', 0.009894616876911393), ('court', 0.005106734727919689), ('issue_area_encoded', 0.0036323714
```

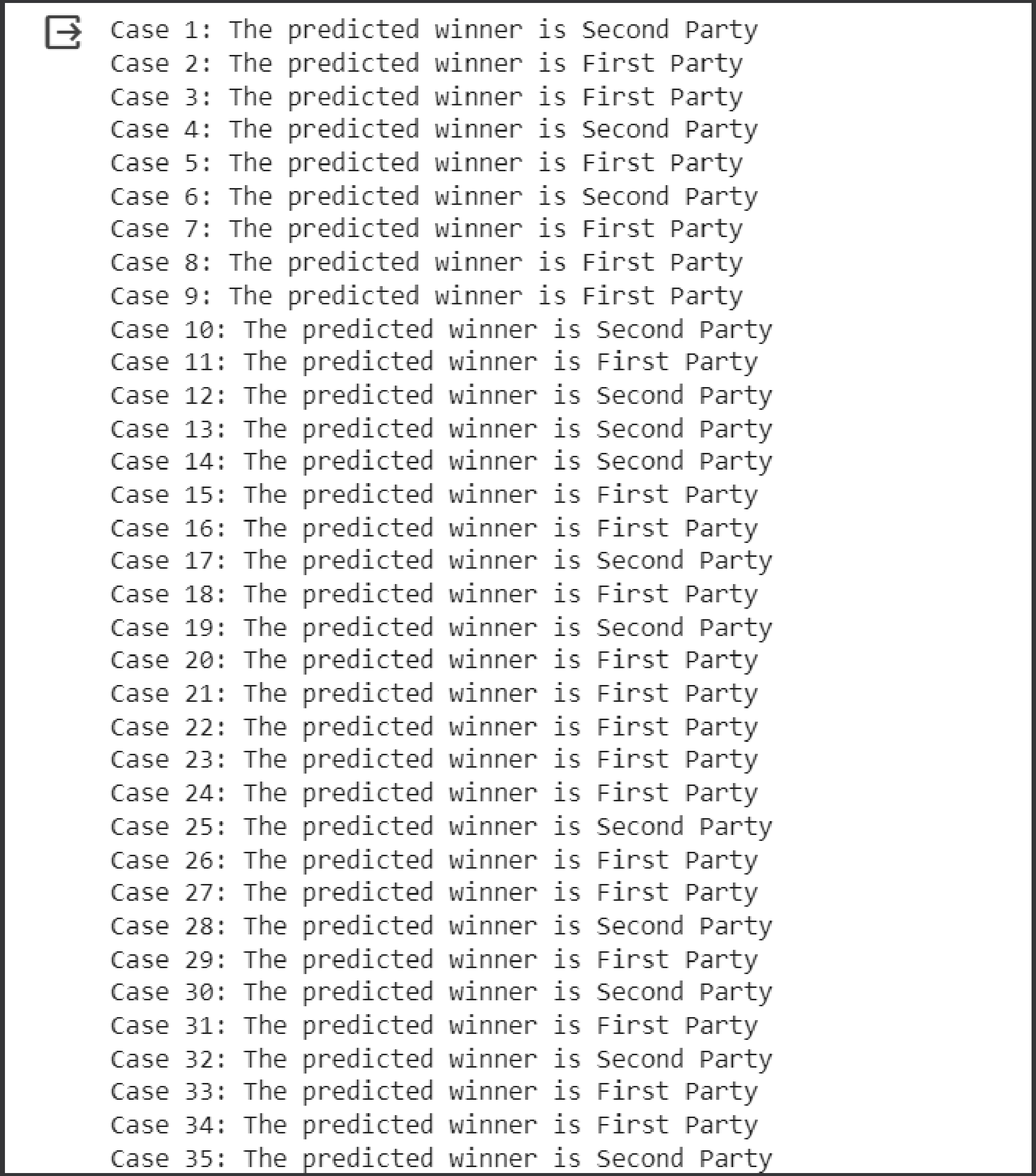
Making Predictions:

- Finally, for each case in the test set, the code predicts whether the first or second party is the winner and prints this prediction.



```
for i, pred in enumerate(y_pred):
    winner = 'First Party' if pred == 1 else 'Second Party'
    print(f"Case {i+1}: The predicted winner is {winner}")
```

Predictions:



```
Case 1: The predicted winner is Second Party
Case 2: The predicted winner is First Party
Case 3: The predicted winner is First Party
Case 4: The predicted winner is Second Party
Case 5: The predicted winner is First Party
Case 6: The predicted winner is Second Party
Case 7: The predicted winner is First Party
Case 8: The predicted winner is First Party
Case 9: The predicted winner is First Party
Case 10: The predicted winner is Second Party
Case 11: The predicted winner is First Party
Case 12: The predicted winner is Second Party
Case 13: The predicted winner is Second Party
Case 14: The predicted winner is Second Party
Case 15: The predicted winner is First Party
Case 16: The predicted winner is First Party
Case 17: The predicted winner is Second Party
Case 18: The predicted winner is First Party
Case 19: The predicted winner is Second Party
Case 20: The predicted winner is First Party
Case 21: The predicted winner is First Party
Case 22: The predicted winner is First Party
Case 23: The predicted winner is First Party
Case 24: The predicted winner is First Party
Case 25: The predicted winner is Second Party
Case 26: The predicted winner is First Party
Case 27: The predicted winner is First Party
Case 28: The predicted winner is Second Party
Case 29: The predicted winner is First Party
Case 30: The predicted winner is Second Party
Case 31: The predicted winner is First Party
Case 32: The predicted winner is Second Party
Case 33: The predicted winner is First Party
Case 34: The predicted winner is First Party
Case 35: The predicted winner is Second Party
```

Total of 658 cases are predicted.

We have used machine learning pipeline, where data is read, preprocessed, and used to train a model, which is then evaluated and used to make predictions.

You can refer to this code…!!!!

You use this code in various fields.

1. **Healthcare and Medical Diagnosis.**
2. **Finance and Credit Scoring**.
3. **Customer Behavior Analysis in Retail.**
4. **Supply Chain and Inventory Management**.
5. **Sports Analytics.**
6. **Social Media and Sentiment Analysis**.