```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import shap
```

```
## Download resources
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
# 1. LOAD AND CLEAN DATA
# ------------------------------------
df = pd.read_csv("/content/WELFake_Dataset.csv", on_bad_lines="skip", engine="python")
df = df.dropna(subset=['label'])
df = df[df['label'].astype(str).str.strip().isin(['0', '1'])]
df['label'] = df['label'].astype(int)
df['title'] = df['title'].fillna('').astype(str)
df['text'] = df['text'].fillna('').astype(str)
df['content'] = (df['title'] + " " + df['text']).fillna('').astype(str)
```

```
# 2. CLEAN TEXT
# ------------------------------------
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r"http\S+|www.\S+", '', text)
    text = re.sub(r"<.*?>", '', text)
    text = re.sub(r"[^a-z\s]", '', text)
    words = text.split()
    words = [w for w in words if w not in stop_words]
    return " ".join(words)

df['content'] = df['content'].apply(clean_text)
```

```
# 3. ADD HANDCRAFTED FEATURES
# ------------------------------------
class TextStats(BaseEstimator, TransformerMixin):
    def fit(self, x, y=None):
        return self
    def transform(self, data):
        # basic stats
        return pd.DataFrame({
            'char_count': data.apply(len),
            'word_count': data.apply(lambda x: len(x.split())),
            'sentence_count': data.apply(lambda x: x.count('.')),
            'uppercase_count': data.apply(lambda x: sum(1 for c in x if c.isupper())),
            'num_count': data.apply(lambda x: sum(1 for c in x if c.isdigit())),
            'punct_count': data.apply(lambda x: sum(1 for c in x if c in "!?"))
        })
```

```
# 4. SETUP FEATURES (TF-IDF WITH N-GRAMS + HANDCRAFTED)
# ------------------------------------
tfidf = TfidfVectorizer(ngram_range=(1,2), max_df=0.8, min_df=3, max_features=12000)

feature_union = FeatureUnion([
    ('tfidf', tfidf),
    ('stats', TextStats())
])



# 5. PREPROCESSING & FEATURE SELECTION PIPELINE
# ------------------------------------
X_features = feature_union.fit_transform(df['content'])
# Feature selection (keep top 8000 overall for speed)
selector = SelectKBest(chi2, k=min(8000, X_features.shape[1]))
X_selected = selector.fit_transform(X_features, df['label'])
y = df['label']



# 6. SPLIT DATA
# ------------------------------------
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42, stratify=y)



# 7. MODELS FOR COMPARISON
# ------------------------------------
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42, n_jobs=-1),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1),
    "Naive Bayes": MultinomialNB(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
}

results = {}
for name, clf in models.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"\n-{name}-")
    print(f"Accuracy: {acc:.4f} | ROC AUC: {roc_auc_score(y_test, y_pred):.3f}")
    print(classification_report(y_test, y_pred))

acc_df = pd.DataFrame([results], index=['Accuracy']).T
print("\nModel Performance Comparison:\n", acc_df)
```

```
-Logistic Regression-
Accuracy: 0.9263 | ROC AUC: 0.925
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       691
           1       0.92      0.94      0.93       774

    accuracy                           0.93      1465
   macro avg       0.93      0.93      0.93      1465
weighted avg       0.93      0.93      0.93      1465


-Random Forest-
Accuracy: 0.9420 | ROC AUC: 0.942
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       691
           1       0.95      0.94      0.94       774

    accuracy                           0.94      1465
   macro avg       0.94      0.94      0.94      1465
weighted avg       0.94      0.94      0.94      1465


-Naive Bayes-
Accuracy: 0.8792 | ROC AUC: 0.879
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       691
           1       0.89      0.89      0.89       774

    accuracy                           0.88      1465
   macro avg       0.88      0.88      0.88      1465
weighted avg       0.88      0.88      0.88      1465


-Decision Tree-
Accuracy: 0.9167 | ROC AUC: 0.917
              precision    recall  f1-score   support

           0       0.91      0.92      0.91       691
           1       0.93      0.91      0.92       774

    accuracy                           0.92      1465
   macro avg       0.92      0.92      0.92      1465
weighted avg       0.92      0.92      0.92      1465


Model Performance Comparison:
                      Accuracy
Logistic Regression  0.926280
Random Forest        0.941980
Naive Bayes          0.879181
Decision Tree        0.916724
```

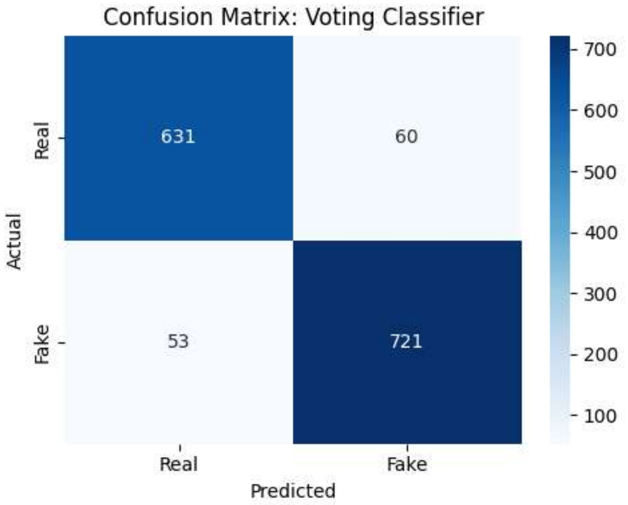
```
# 8. ENSEMBLE VOTING CLASSIFIER
# ------------------------------------
voting_clf = VotingClassifier([
    ('lr', LogisticRegression(max_iter=1000, random_state=42, n_jobs=-1)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)),
    ('nb', MultinomialNB())
], voting='soft', n_jobs=-1)
voting_clf.fit(X_train, y_train)
voting_pred = voting_clf.predict(X_test)
print("\n--Voting Classifier--")
print("Accuracy:", accuracy_score(y_test, voting_pred))
print("ROC AUC:", roc_auc_score(y_test, voting_pred))
print(classification_report(y_test, voting_pred))
```

```
--Voting Classifier--
Accuracy: 0.9228668941979522
ROC AUC: 0.9223469338149781
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       691
           1       0.92      0.93      0.93       774

    accuracy                           0.92      1465
   macro avg       0.92      0.92      0.92      1465
weighted avg       0.92      0.92      0.92      1465
```

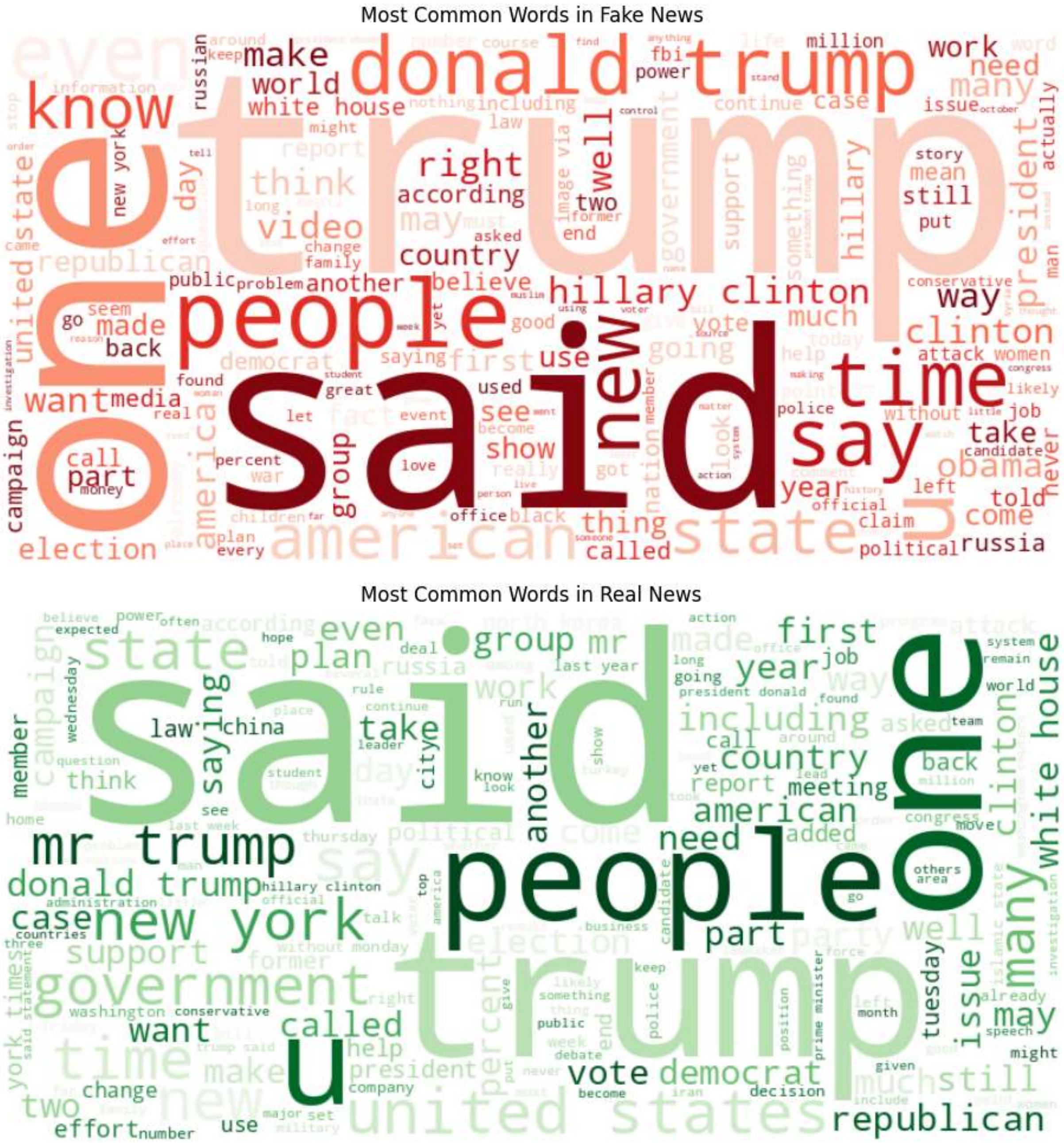
```
# 9. CROSS-VALIDATION FOR ENSEMBLE
# ------------------------------------
cv_accuracy = cross_val_score(voting_clf, X_train, y_train, scoring='accuracy', cv=3)
cv_roc = cross_val_score(voting_clf, X_train, y_train, scoring='roc_auc', cv=3)
print(f"\nVoting CV Accuracy: {cv_accuracy.mean():.3f} (+/- {cv_accuracy.std():.3f})")
print(f"Voting CV ROC AUC: {cv_roc.mean():.3f} (+/- {cv_roc.std():.3f})")
```

```
Voting CV Accuracy: 0.905 (+/- 0.005)
Voting CV ROC AUC: 0.971 (+/- 0.002)
```

```
# 10. CONFUSION MATRIX FOR VOTING CLASSIFIER
# -----------------------------------------
cm = confusion_matrix(y_test, voting_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Real", "Fake"], yticklabels=["Real", "Fake"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix: Voting Classifier")
plt.tight_layout()
plt.show()
```

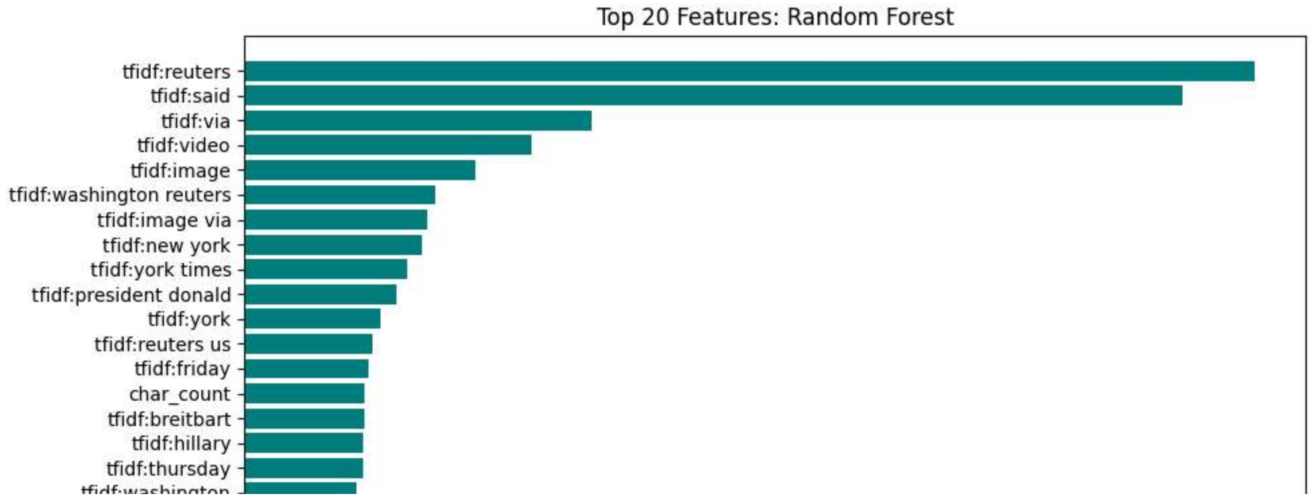


```
# 11. WORD CLOUD VISUALIZATION
# -----------------------------------------
for label, color, title in [(1, 'Reds', 'Fake News'), (0, 'Greens', 'Real News')]:
    txt = " ".join(df[df['label'] == label]['content'])
    wc = WordCloud(width=800, height=400, background_color='white', colormap=color).generate(txt)
    plt.figure(figsize=(10,5))
    plt.imshow(wc, interpolation='bilinear')
    plt.axis('off')
    plt.title(f"Most Common Words in {title}")
    plt.tight_layout()
    plt.show()
```


Most Common Words in Fake News

Most Common Words in Real News

```
# 12. FEATURE IMPORTANCE (RANDOM FOREST)
# -----------------------------------------
if hasattr(models["Random Forest"], "feature_importances_"):
    importances = models["Random Forest"].feature_importances_
    idx_top = np.argsort(importances)[-20:][::-1]
    feature_names = np.array([f"tfidf:{f}" for f in tfidf.get_feature_names_out()] + \
                             list(['char_count','word_count','sentence_count','uppercase_count','num_count','punct_count']))
    top_feat_names = feature_names[selector.get_support(indices=True)][idx_top]
    plt.figure(figsize=(10,5))
    plt.barh(top_feat_names, importances[idx_top], color='teal')
    plt.xlabel("Importance")
    plt.title("Top 20 Features: Random Forest")
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()
```



```
# 13. ERROR ANALYSIS: False Positives/Negatives
# -----------------------------------------
X_test_idx = y_test.reset_index(drop=True)
false_pos_idx = np.where((y_test==0)&(voting_pred==1))[0]
false_neg_idx = np.where((y_test==1)&(voting_pred==0))
print("\nSample False Positives (Real misclassified as Fake):")
print(df.iloc[X_test_idx.index[false_pos_idx]][['title','text']].head(2))
print("\nSample False Negatives (Fake misclassified as Real):")
print(df.iloc[X_test_idx.index[false_neg_idx]][['title','text']].head(2))
```