```
# 1. Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from imblearn.under_sampling import RandomUnderSampler


2.Load dataset
import pandas as pd
# Load dataset
df = pd.read_csv('fake_news_dataset (1).csv')
print("Data loaded successfully.")
print(df.head())
print(df.columns)
```

3.Drop missing values

```
# Drop rows with missing text or label
df = df.dropna(subset=['text', 'label'])  # Replace column names as needed
df.reset_index(drop=True)
```

4.Downloading pakages

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()
def clean_text(text):
    text = re.sub('[^a-zA-Z]', ' ', text)  # Remove non-alphabetic characters
    text = text.lower().split()
    text = [ps.stem(word) for word in text if word not in stop_words]
    return ' '.join(text)
```

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


5. Getting samples
X_sample = vectorizer.fit_transform(df['clean_text'][:1000]).toarray()
print("Starting TF-IDF vectorization...")
X = vectorizer.fit_transform(df['clean_text']).toarray()
X_sample = vectorizer.fit_transform(df['clean_text'][:1000]).toarray()
print("Starting TF-IDF vectorization...")
X = vectorizer.fit_transform(df['clean_text']).toarray()
print("TF-IDF vectorization completed.")
print(df['clean_text'].isnull().sum())
print(df['clean_text'].head())
vectorizer = TfidfVectorizer()
vectorizer = TfidfVectorizer(max_features=1000)
```

6. Resampled shape

```
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)
print("Resampled shape:", X_train_resampled.shape)
```

7. Using randomForest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

model = RandomForestClassifier(random_state=42)
model.fit(X_train_resampled, y_train_resampled)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

8. using visualization

```
# 1. Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from imblearn.under_sampling import RandomUnderSampler
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer

# Download NLTK stopwords if not already downloaded
try:
```

```
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')


stop_words = set(stopwords.words('english'))
ps = PorterStemmer()


def clean_text(text):
    """Cleans the input text."""
    text = str(text) # Ensure text is a string
    text = re.sub('[^a-zA-Z]', ' ', text)  # Remove non-alphabetic characters
    text = text.lower().split()
    text = [ps.stem(word) for word in text if word not in stop_words]
    return ' '.join(text)


# Load dataset
df = pd.read_csv('fake_news_dataset (1).csv')
print("Data loaded successfully.")
print(df.head())
```

```
print(df.columns)


# Apply text cleaning to create the 'clean_text' column
df['clean_text'] = df['text'].apply(clean_text)


# Drop rows with missing clean_text or label
# Now 'clean_text' column exists
df = df.dropna(subset=['clean_text', 'label'])
df.reset_index(drop=True, inplace=True) # Added inplace=True for the change to take effect


print("Rows with missing text or label dropped.")
print("Starting TF-IDF vectorization...")


# Initialize the vectorizer before using it
vectorizer = TfidfVectorizer(max_features=1000)


# Perform TF-IDF vectorization
# Use the full cleaned text data
X = vectorizer.fit_transform(df['clean_text']).toarray()
```

```
y = df['label'] # Assuming 'label' is your target variable


print("TF-IDF vectorization completed.")

print(df['clean_text'].isnull().sum())

print(df['clean_text'].head())


# Split data into training and testing sets

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


# Apply Random Under-Sampling to the training data

rus = RandomUnderSampler(random_state=42)

X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)


print("Resampled shape:", X_train_resampled.shape)


# Train the Random Forest Classifier

model = RandomForestClassifier(random_state=42)

model.fit(X_train_resampled, y_train_resampled)
```

```
# Evaluate the model
y_pred = model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))


# --- Visualization: Pie Chart of Real vs Fake News ---
# Reload or ensure df has the 'label' column (it should from the initial load)
# If you dropped rows, the counts will reflect the remaining data
label_counts = df['label'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(label_counts, labels=['Real', 'Fake'], autopct='%1.1f%%', colors=['skyblue', 'salmon'])
plt.title('Distribution of Real vs Fake News')
plt.axis('equal')
plt.show()


# --- Function to generate alert ---
def generate_alert(text):
    """Generates an alert based on the text provided."""
    cleaned = clean_text(text)
```

```
    # Ensure the vectorizer is fitted and ready to transform
    # It was fitted earlier with the training data
    vector = vectorizer.transform([cleaned]) # Use transform, not fit_transform
    prediction = model.predict(vector)[0]
    if prediction == 1:
        return "ALERT: Possibly FAKE NEWS!"
    else:
        return "Legitimate News."

# Example usage
sample_news = "The president has been kidnapped by aliens."
print(generate_alert(sample_news))
```

9.Using SNS plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
import numpy as np

# --- Bar Chart: Label Distribution ---
plt.figure(figsize=(6, 4))
sns.countplot(x='label', data=df, palette='Set2')
plt.xticks([0, 1], ['Real', 'Fake'])
plt.title('Count of Real vs Fake News')
plt.xlabel('News Type')
plt.ylabel('Count')
plt.show()

# --- Vectorization ---
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['clean_text']).toarray()
y = df['label']
```

```
# --- Train/Test Split & Resampling ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)

# --- Train Model ---
model = RandomForestClassifier(random_state=42)
model.fit(X_train_resampled, y_train_resampled)

# --- Feature Importance Visualization ---
importances = model.feature_importances_
feature_names = vectorizer.get_feature_names_out()

# Filter non-zero importances
nonzero_indices = importances > 0
importances = importances[nonzero_indices]
feature_names = feature_names[nonzero_indices]

# Get Top N
```

```
top_n = min(20, len(feature_names))
indices = np.argsort(importances)[-top_n:]
top_features = feature_names[indices]
top_importances = importances[indices]

# --- Plot Top Important Words ---
plt.figure(figsize=(10, 6))
plt.barh(top_features, top_importances, color='teal')
plt.xlabel("Feature Importance")
plt.title("Top 20 Important Words in Fake News Detection")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

10. Using tabulate values

```
  import pandas as pd
from tabulate import tabulate  # If not installed, run: pip install tabulate
import re
from nltk.corpus import stopwords
```

```
from nltk.stem import PorterStemmer
import nltk

# Download NLTK stopwords if not already downloaded
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def clean_text(text):
    """Cleans the input text."""
    text = str(text) # Ensure text is a string
    text = re.sub('[^a-zA-Z]', ' ', text)  # Remove non-alphabetic characters
    text = text.lower().split()
    text = [ps.stem(word) for word in text if word not in stop_words]
    return ' '.join(text)
```

```
# Load the dataset
df = pd.read_csv('fake_news_dataset (1).csv')

# Apply text cleaning to create the 'clean_text' column
# This step was missing in the original problematic cell
df['clean_text'] = df['text'].apply(clean_text)

# Drop rows with missing clean_text or label
# Now 'clean_text' column exists and can be referenced
df = df.dropna(subset=['clean_text', 'label'])

# Invert label mapping: 1 -> 0 (FAKE), 0 -> 1 (REAL becomes 1)
# This mapping might be incorrect if 1 is 'Fake' and 0 is 'Real'.
# If 1 is FAKE and you want the label column to show 0 for FAKE and 1 for REAL,
# the mapping should be {1: 0, 0: 1}. Your current mapping {1: 0, 0: 1} achieves this.
# If you intended 0 to be REAL and 1 to be FAKE, and want the 'Label' column
# to represent this with the same numerical values (0 for REAL, 1 for FAKE),
# then you would use {0: 0, 1: 1} or simply df['Label'] = df['label'].
```

```
df['Label'] = df['label'].map({1: 0, 0: 1})


# Prepare the table DataFrame
# Ensure 'clean_text' exists before renaming
table_df = df[['clean_text', 'Label']].rename(columns={'clean_text': 'News_Content'})


# Show first 20 rows as a sample
print(tabulate(table_df.head(20), headers='keys', tablefmt='pretty', showindex=True))


11. Box plot
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Import necessary libraries for cleaning if not already imported in this cell
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
```

```
# Download NLTK stopwords if not already downloaded
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')


stop_words = set(stopwords.words('english'))
ps = PorterStemmer()


def clean_text(text):
    """Cleans the input text."""
    text = str(text) # Ensure text is a string
    text = re.sub('[^a-zA-Z]', ' ', text)  # Remove non-alphabetic characters
    text = text.lower().split()
    text = [ps.stem(word) for word in text if word not in stop_words]
    return ' '.join(text)


# Load dataset
df = pd.read_csv('fake_news_dataset (1).csv')
```

```
# Drop missing values in required columns
df = df.dropna(subset=['label', 'text']) # Also drop missing text for cleaning

# Apply text cleaning to create the 'clean_text' column
df['clean_text'] = df['text'].apply(clean_text)

# Create a new column: text length
df['text_length'] = df['clean_text'].apply(lambda x: len(str(x).split()))

# Plot boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='text_length', data=df, palette='Set3')
plt.xticks([0, 1], ['REAL (0)', 'FAKE (1)'])
plt.title('Box Plot of News Text Length by Label')
plt.xlabel('News Type')
plt.ylabel('Number of Words in News')
plt.tight_layout()
plt.show()
```

12.Alerting

```
def generate_alert(text):
    cleaned = clean_text(text)
    vector = vectorizer.transform([cleaned]).toarray()
    prediction = model.predict(vector)[0]
    if prediction == 1:
        return "ALERT: Possibly FAKE NEWS!"
    else:
        return "Legitimate News."

# Example usage
sample_news = "The president has been kidnapped by aliens."
print(generate_alert(sample_news))
```