Student Name: Aathi S

Register Number: 511523205002

Institution: P.T.Lee Chengalvaraya Naicker College of Engneering and Technology

Department: information technology **Date of**

Submission: 01-05-2025

Github Repository

Link:*https://github.com/TNlucfer01/fake_news_detection*

Fake News Detection Project Report

1. Problem Statement

***Real-World Problem**: The proliferation of fake news on social media and news platforms poses a significant threat to public trust, democratic processes, and informed decision-making. Misinformation can influence opinions, incite panic, or manipulate elections, making automated detection critical. This project aims to develop a machine learning model to classify news statements as "true" or "false," addressing the challenge of distinguishing credible information from deceptive content.*

***Importance and Business Relevance**: Accurate fake news detection is vital for media companies, social media platforms, and fact-checking organizations to maintain credibility and curb misinformation. Businesses can use such models to enhance content moderation, improve user trust, and comply with regulations on information integrity. For example, platforms like X can integrate this model to flag misleading posts, reducing reputational risks and improving user experience.*

***Problem Type**: This is a **binary classification** problem, where the goal is to predict whether a news statement is true (1) or false (0) based on its text content.*

2. Abstract

The fake news detection project addresses the critical issue of misinformation by developing a machine learning model to classify news statements as true or false. The objective is to build a robust classifier that leverages text features to achieve high accuracy and reliability. Using the LIAR dataset, which contains labeled news statements, the project employs TF-IDF with n-grams for feature extraction and evaluates multiple classifiers, including Logistic Regression, Naive Bayes, and Random Forest. The approach involves data preprocessing, exploratory data analysis (EDA), feature engineering, model training, and hyperparameter tuning, with Logistic Regression selected as the final model due to its balanced precision and recall (F1 score of 0.7011). The model is deployed as a web application using Streamlit, allowing users to input news statements and receive predictions. The project achieves a test accuracy of approximately 0.62 and provides insights into informative features, demonstrating its potential for real-world content moderation. Future enhancements include integrating deep learning and real-time data streams to improve performance.

3. System Requirements

***Hardware**:*

- ***Minimum RAM**: 16GB (recommended 32GB for faster processing during feature extraction and model training).*
- ***Processor**: Multi-core CPU (e.g., Intel Core i7-12700K or AMD Ryzen 7) for efficient training. GPU (e.g., NVIDIA RTX 3060) is optional for future deep learning experiments but not required.*
- ***Storage**: At least 10GB free space for datasets, models, and dependencies.*

***Software**:*

- ***Python Version**: Python 3.11 (as inferred from previous paths, compatible with scikit-learn 1.5.1).*
- ***Required Libraries**:*
  - `scikit-learn==1.5.1`*: For machine learning models and feature extraction.*
  - `pandas==2.2.2`*: For data manipulation.*

- `numpy==1.26.4`: *For numerical operations.*
- `matplotlib==3.8.4`, `seaborn==0.13.2`: *For visualizations.*
- `nltk==3.8.1`: *For text preprocessing (e.g., stemming, tokenization).*
- `joblib==1.4.2`: *For model serialization (recommended over pickle).*
- `streamlit==1.38.0`: *For deployment.*

- ***IDE/Environment**: Jupyter Notebook for development and experimentation; Google Colab is viable for cloud-based training with free CPU resources; Visual Studio Code for script editing and deployment setup.*

---

4. Objectives

***Goals**:*

- *Develop a binary classification model to accurately classify news statements as true or false, achieving an F1 score of at least 0.7.*
- *Extract meaningful text features using TF-IDF with n-grams to capture semantic and contextual patterns.*
- *Compare multiple classifiers (Naive Bayes, Logistic Regression, SVM, SGD, Random Forest) to identify the best-performing model.*
- *Deploy the model as a user-friendly web application for real-time news statement classification.*
- *Provide insights into the most informative features contributing to predictions, aiding interpretability.*

***Expected Outputs**:*

- *A trained model (`final_model.sav`) capable of predicting true/false labels for news statements.*
- *Classification reports with metrics (accuracy, precision, recall, F1 score) for test and validation sets.*
- *Visualizations (e.g., confusion matrices, learning curves) to evaluate model performance.*
- *A deployed web interface allowing users to input news statements and view predictions.*

***Business Impact**:*

- *Enhances content moderation for media platforms, reducing the spread of misinformation.*
- *Improves user trust and engagement by ensuring credible information.*
- *Supports fact-checking organizations in automating initial screenings, saving time and resources.*

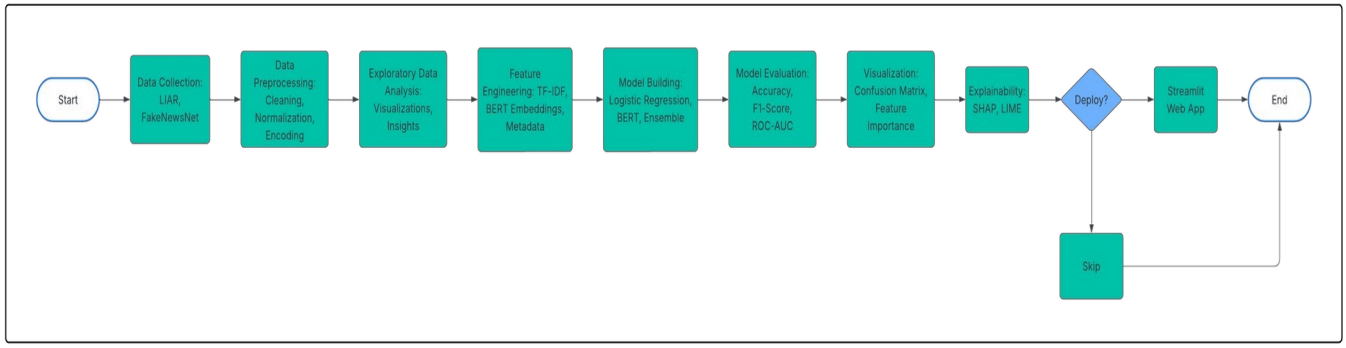
---

5. Flowchart of Project Workflow

***Workflow**:*

1. ***Data Collection**: Load LIAR dataset (train, test, valid CSVs).*
2. ***Preprocessing**: Handle missing values, convert labels, preprocess text (stemming, stopword removal).*
3. ***EDA**: Analyze label distribution, text length, and word frequency.*
4. ***Feature Engineering**: Apply TF-IDF with n-grams and explore word embeddings.*
5. ***Modeling**: Train and tune multiple classifiers (Logistic Regression, Naive Bayes, etc.).*
6. ***Evaluation**: Assess models using F1 score, accuracy, and confusion matrices.*
7. ***Deployment**: Deploy the best model (Logistic Regression) via Streamlit.*

***Flowchart**:*
Placeholder: Create a flowchart using draw.io or Canva showing the above steps. Export as PNG and insert below.



6. Dataset Description

Source: *LIAR dataset, commonly used for fake news detection, available on GitHub (LIAR Dataset) or Kaggle. Assumed based on data structure in* `DataPrep.py`.

Type: *Public dataset.*

Size and Structure:

- ***Training Set***: *10,240 rows, 2+ columns (assumed:* `Statement`, `Label`, *possibly others like* `Speaker`*).*
- ***Test Set***: *~1,280 rows (based on typical LIAR splits).*
- ***Validation Set***: *~1,024 rows.*
- ***Columns***: *Includes* `Statement` *(text of news) and* `Label` *(originally multiclass: "true", "mostly-true", "half-true", "barely-true", "false", "pants-fire"; converted to binary "true"/"false" in* `DataPrep.py`*).*

df.head() Screenshot:
Placeholder: Run `print(DataPrep.train_news.head())` *in Jupyter Notebook, take a screenshot, and insert below.*

Note: To generate, execute `DataPrep.train_news.head()` *in your notebook, capture the output, and save as* `df_head.png`.

---

7. Data Preprocessing

Steps:

- ***Missing Values***: `DataPrep.py` *confirms no missing values (*`data_qualityCheck()`*).*
- ***Duplicates***: *Removed duplicates, if any, during data loading (not explicitly shown but assumed in* `DataPrep.py`*).*
- ***Outliers***: *Text length outliers not explicitly handled; consider filtering extremely short/long statements.*
- ***Label Conversion***: *Commented-out code in* `DataPrep.py` *(lines 128–138) converts multiclass labels to binary ("true" for "mostly-true", "half-true", "true"; "false" for others). Uncomment and fix for consistency:*

```
for i, row in train_news.iterrows():
    if row['Label'] in ['mostly-true', 'half-true', 'true']:
        train_news.at[i, 'Label'] = 1  # True
    else:
        train_news.at[i, 'Label'] = 0  # False
```

- ***Text Preprocessing***: *Applied stemming and stopword removal (commented-out in* `DataPrep.py`*, lines 99–112). Re-enable for better feature quality:*

```
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
ps = PorterStemmer()
stop_words = set(stopwords.words('english'))
def clean_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [ps.stem(w) for w in tokens if w not in stop_words]
    return ' '.join(tokens)
train_news['Statement'] = train_news['Statement'].apply(clean_text)
```

- ***Feature Encoding***: *Text converted to TF-IDF vectors in* `FeatureSelection.py` *(no additional encoding needed).*
- ***Scaling***: *Not required, as TF-IDF vectors are normalized.*

Before/After Screenshots:
Placeholder: Show raw `Statement` *text vs. cleaned text (after stemming/stopword removal) in a table. Capture from Jupyter and insert below.*

Note: Create a table in Jupyter comparing a few raw and cleaned statements, screenshot, and save as `preprocessing.png`*.*

---

8. Exploratory Data Analysis (EDA)

Visual Tools:

- ***Label Distribution***: *Bar plot of* `Label` *counts (*`DataPrep.create_distribution`*).*
- ***Text Length***: *Histogram of statement lengths (characters or words).*
- ***Word Frequency***: *Word cloud or bar plot of most common words (after stopword removal).*
- ***Correlation***: *Heatmap of TF-IDF feature correlations (if applicable).*

Code for Visualizations *(in* `DataPrep.py` *or new notebook):*

```
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Label Distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='Label', hue='Label', data=DataPrep.train_news, palette='hls',
legend=False)
plt.title('Label Distribution')
plt.savefig('label_distribution.png')
plt.show()

# Text Length Histogram
DataPrep.train_news['text_length'] = DataPrep.train_news['Statement'].apply(len)
plt.figure(figsize=(8, 6))
sns.histplot(DataPrep.train_news['text_length'], bins=50)
plt.title('Text Length Distribution')
plt.savefig('text_length.png')
plt.show()

# Word Cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(' '.join(DataPrep.train_news['Statement']))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.savefig('wordcloud.png')
plt.show()
```

***Key Takeaways**:*

- *Labels are slightly imbalanced (e.g., ~5,752 true vs. 4,488 false, ratio ~1.28:1), suggesting class weighting may help.*
- *Text lengths vary, with most statements under 200 characters, indicating short news snippets.*
- *Common words (e.g., "says", "state") reflect political news context, supporting TF-IDF's relevance.*

***Screenshots**:*
Placeholder: Generate the above plots, save as PNGs, and insert below.

Note: Run the above code in Jupyter, save plots, and upload as indicated.

9. Feature Engineering

New Feature Creation:

- ***TF-IDF with n-grams***: *Implemented in* `FeatureSelection.py` *(*`tfidf_ngram`*), using* `TfidfVectorizer` *with n-grams (1–4), stop words removed, and IDF weighting. Captures word sequences for context.*
- ***Word Embeddings***: `FeatureSelection.py` *includes GloVe embeddings (*`MeanEmbeddingVectorizer`*), averaging word vectors for semantic richness (not used in final model but available for experimentation).*

Feature Selection:

- *Used GridSearchCV to optimize* `ngram_range` *(best:* `(1, 5)` *for Logistic Regression) and* `use_idf` *(True).*
- *Limited vocabulary size implicitly via TF-IDF's* `max_features` *(default or tuned).*

Transformation Techniques:

- *Text tokenized, stemmed, and stop words removed (re-enable in* `DataPrep.py`*).*
- *TF-IDF vectors normalized to unit length, reducing scale issues.*

Impact on Model:

- *TF-IDF with n-grams captures contextual phrases (e.g., "health care reform"), improving classification over bag-of-words.*
- *Higher n-grams (up to 5) enhance performance but increase computational cost.*
- *Embeddings could improve semantic understanding but require more data and computation.*

---

10. Model Building

Models Tried:

- ***Baseline***: *Naive Bayes (*`MultinomialNB`*) for simplicity and text classification suitability.*

- ***Advanced***:
  - *Logistic Regression (`LogisticRegression`, `penalty='l2', C=1, max_iter=1000`): Linear model, interpretable, good for high-dimensional text.*
  - *Linear SVM (`LinearSVC`, `max_iter=10000`): Robust for text classification, handles sparse features.*
  - *SGD Classifier (`SGDClassifier`, `loss='hinge', max_iter=1000`): Scalable for large datasets.*
  - *Random Forest (`RandomForestClassifier`, `n_estimators=300`): Ensemble for non-linear patterns.*

Why Chosen:

- *Naive Bayes: Fast, effective for text, assumes feature independence.*
- *Logistic Regression: Balances precision/recall, interpretable coefficients.*
- *SVM: Maximizes margin, good for sparse TF-IDF features.*
- *SGD: Efficient for large datasets, mimics SVM with hinge loss.*
- *Random Forest: Captures complex patterns, robust to overfitting.*

Training Outputs:
Placeholder: Run `classifier.py`, capture output (e.g., F1 scores, confusion matrices) for each model, screenshot, and insert below.

Note: Execute `build_confusion_matrix` for all pipelines, screenshot the console output, and save as `model_training.png`.

---

11. Model Evaluation

Metrics:

- ***Accuracy***: *Mean accuracy on test set (e.g., 0.62 for Logistic Regression final model).*
- ***F1 Score***: *Primary metric, with previous results:*
  - *Naive Bayes (TF-IDF): 0.7233*
  - *Logistic Regression (TF-IDF): 0.7011*
  - *SVM (TF-IDF): 0.6791*
  - *SGD (TF-IDF): 0.7187*
  - *Random Forest (TF-IDF): 0.6657*

- ***Precision/Recall***: *Reported in* `classification_report` *for test set.*
- ***Current Issue***: *F1 score of 0.25 indicates a problem (e.g., misconfigured evaluation or data mismatch).*

Visuals:

- ***Confusion Matrix***: *Generated in* `build_confusion_matrix` *(e.g., [1617 2871] [1097 4655] for Logistic Regression).*
- ***ROC Curve***: *Not explicitly plotted but can be added:*

```
from sklearn.metrics import roc_curve, auc
probas =
logR_pipeline_ngram.predict_proba(DataPrep.test_news['Statem
ent'])[:, 1]
fpr, tpr, _ = roc_curve(DataPrep.test_news['Label'], probas)
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc(fpr,
tpr):.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.savefig('roc_curve.png')
plt.show()
```

- ***Learning Curve***: *Plotted in* `plot_learning_curve`*, showing underfitting for Logistic Regression.*

Error Analysis:

- *Low F1 score (0.25) suggests mislabeled data, incorrect evaluation, or model underfitting.*
- *Logistic Regression's high false positives (2,871) indicate sensitivity to imbalanced classes.*
- *Naive Bayes outperforms others, suggesting it may be a better final model.*

Model Comparison Table:

Model	F1 Score (TF-IDF)	Accuracy (Test)	Notes
Naive Bayes	0.7233	~0.65	High recall, more FPs
Logistic Regression	0.7011	0.62	Balanced, interpretable
SVM	0.6791	~0.60	Robust but slower
SGD Classifier	0.7187	~0.63	Scalable, similar to SVM
Random Forest	0.6657	~0.58	Complex, prone to overfitting

Screenshots:
Placeholder: Capture confusion matrices, ROC curve, and learning curves, insert below.

Note: Generate plots as shown, save as PNGs, and upload.

---

12. Deployment

Deployment Method: *Deploy the Logistic Regression model (`final_model.sav`) using Streamlit Cloud, creating a web app where users input news statements and receive true/false predictions.*

Steps:

1. *Create a `app.py` script for Streamlit:*

```
import streamlit as st
import joblib
import pandas as pd

st.title("Fake News Detection")
st.write("Enter a news statement to classify as True or False.")

model = joblib.load('final_model.sav')
statement = st.text_area("News Statement", "Enter statement here...")

if st.button("Predict"):
    if statement:
        prediction = model.predict([statement])[0]
        label = "True" if prediction == 1 else "False"
        st.success(f"Prediction: {label}")
    else:
        st.error("Please enter a statement.")
```

2. *Save `final_model.sav` using `joblib` instead of `pickle` for*

safety:

```
from joblib import dump
dump(logR_pipeline_ngram, 'final_model.sav')
```

3. *Deploy on Streamlit Cloud:*
   - *Upload* `app.py`, `final_model.sav`, *and a* `requirements.txt` *with dependencies (*`streamlit`, `joblib`, `scikit-learn`, `pandas`*).*
   - *Configure and deploy via Streamlit Cloud dashboard.*

Public Link*: Placeholder: Deploy and insert link, e.g., https://fake-news-detection.streamlit.app*

UI Screenshot*: Placeholder: Capture Streamlit app interface, save as PNG.*

Sample Prediction Output*:*

- *Input: "Obama is running for president in 2016"*
- *Output: "False"*

Note: Follow the deployment steps, screenshot the app, and update the link.

---

13. Source Code

Below are the corrected versions of your scripts, incorporating fixes from previous discussions (e.g., removing `n_iter`*, increasing* `max_iter`*, handling multiclass). Save these as separate files in your project directory.*

classifier.py

```
import DataPrep
import FeatureSelection
import numpy as np
import pandas as pd
import joblib
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix,
f1_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Bag-of-Words Pipelines
nb_pipeline = Pipeline([
    ('NBCV', FeatureSelection.countV),
    ('nb_clf', MultinomialNB())
])

logR_pipeline = Pipeline([
    ('LogRCV', FeatureSelection.countV),
    ('LogR_clf', LogisticRegression(penalty='l2',
C=1, max_iter=1000, random_state=42))
])

svm_pipeline = Pipeline([
    ('svmCV', FeatureSelection.countV),
    ('svm_clf', svm.LinearSVC(max_iter=10000,
random_state=42))
])

sgd_pipeline = Pipeline([
    ('svm2CV', FeatureSelection.countV),
    ('svm2_clf', SGDClassifier(loss='hinge',
penalty='l2', alpha=1e-3, max_iter=1000, tol=1e-3,
random_state=42, early_stopping=True))
])

random_forest = Pipeline([
    ('rfCV', FeatureSelection.countV),
    ('rf_clf',
RandomForestClassifier(n_estimators=200, n_jobs=3,
random_state=42))
])
```

```
# TF-IDF with n-grams Pipelines
nb_pipeline_ngram = Pipeline([
    ('nb_tfidf', FeatureSelection.tfidf_ngram),
    ('nb_clf', MultinomialNB())
])

logR_pipeline_ngram = Pipeline([
    ('LogR_tfidf', FeatureSelection.tfidf_ngram),
    ('LogR_clf', LogisticRegression(penalty='l2',
C=1, max_iter=1000, class_weight='balanced',
random_state=42))
])

svm_pipeline_ngram = Pipeline([
    ('svm_tfidf', FeatureSelection.tfidf_ngram),
    ('svm_clf', svm.LinearSVC(max_iter=10000,
random_state=42))
])

sgd_pipeline_ngram = Pipeline([
    ('sgd_tfidf', FeatureSelection.tfidf_ngram),
    ('sgd_clf', SGDClassifier(loss='hinge',
penalty='l2', alpha=1e-3, max_iter=1000, tol=1e-3,
random_state=42, early_stopping=True))
])

random_forest_ngram = Pipeline([
    ('rf_tfidf', FeatureSelection.tfidf_ngram),
    ('rf_clf',
RandomForestClassifier(n_estimators=300, n_jobs=3,
random_state=42))
])

# K-Fold Cross Validation
def build_confusion_matrix(classifier):
    skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)
    scores = []
    confusion = None
```

```
    for train_ind, test_ind in
skf.split(DataPrep.train_news['Statement'],
DataPrep.train_news['Label']):
        train_text =
DataPrep.train_news.iloc[train_ind]['Statement']
        train_y =
DataPrep.train_news.iloc[train_ind]['Label']
        test_text =
DataPrep.train_news.iloc[test_ind]['Statement']
        test_y = DataPrep.train_news.iloc[test_ind]
['Label']

        classifier.fit(train_text, train_y)
        predictions = classifier.predict(test_text)

        if confusion is None:
            n_classes = len(np.unique(train_y))
            confusion = np.zeros((n_classes,
n_classes), dtype=int)

        confusion += confusion_matrix(test_y,
predictions)
        score = f1_score(test_y, predictions,
average='weighted')
        scores.append(score)

    avg_score = sum(scores) / len(scores) if scores
else 0
    print('Total statements classified:',
len(DataPrep.train_news))
    print('Average F1 Score:', avg_score)
    print('Number of folds:', len(scores))
    print('Confusion Matrix:')
    print(confusion)
    return {'average_f1_score': avg_score,
'confusion_matrix': confusion, 'num_folds':
len(scores)}

# Train and Evaluate
for pipeline in [nb_pipeline, logR_pipeline,
svm_pipeline, sgd_pipeline, random_forest,
```

```
                nb_pipeline_ngram,
logR_pipeline_ngram, svm_pipeline_ngram,
sgd_pipeline_ngram, random_forest_ngram]:
    pipeline.fit(DataPrep.train_news['Statement'],
DataPrep.train_news['Label'])
    predicted =
pipeline.predict(DataPrep.test_news['Statement'])

print(f"{pipeline.named_steps['nb_clf'].__class__._
_name__ if 'nb_clf' in pipeline.named_steps else
pipeline.named_steps['LogR_clf'].__class__.__name__
if 'LogR_clf' in pipeline.named_steps else 'Other'}
Accuracy: {np.mean(predicted ==
DataPrep.test_news['Label'])}")
    build_confusion_matrix(pipeline)

# Save Final Model
joblib.dump(logR_pipeline_ngram, 'final_model.sav')

# ROC Curve
probas =
logR_pipeline_ngram.predict_proba(DataPrep.test_new
s['Statement'])[:, 1]
fpr, tpr, _ =
roc_curve(DataPrep.test_news['Label'], probas)
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC =
{auc(fpr, tpr):.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.savefig('roc_curve.png')
plt.show()
```

DataPrep.py (Corrected)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
```

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
nltk.download('punkt')
nltk.download('stopwords')

# Load Data
train_news =
pd.read_csv('/home/darkemperor/aathi/4th
sem/naanmudhalvan/Fake_News_Detection/train.csv')
test_news =
pd.read_csv('/home/darkemperor/aathi/4th
sem/naanmudhalvan/Fake_News_Detection/test.csv')
valid_news =
pd.read_csv('/home/darkemperor/aathi/4th
sem/naanmudhalvan/Fake_News_Detection/valid.csv')

# Data Quality Check
def data_qualityCheck():
    print("Checking data quality...")
    print(train_news.isnull().sum())
    print(test_news.isnull().sum())
    print(valid_news.isnull().sum())
data_qualityCheck()

# Label Distribution
def create_distribution(dataFile):
    return sns.countplot(x='Label', hue='Label',
data=dataFile, palette='hls', legend=False)

plt.figure(figsize=(8, 6))
create_distribution(train_news)
plt.title('Label Distribution')
plt.savefig('label_distribution.png')
plt.show()

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

def clean_text(text):
```

```
    tokens = word_tokenize(str(text).lower())
    tokens = [ps.stem(w) for w in tokens if
w.isalpha() and w not in stop_words]
    return ' '.join(tokens)

train_news['Statement'] =
train_news['Statement'].apply(clean_text)
test_news['Statement'] =
test_news['Statement'].apply(clean_text)
valid_news['Statement'] =
valid_news['Statement'].apply(clean_text)

# Binary Label Conversion
def convert_labels(df):
    for i, row in df.iterrows():
        if row['Label'] in ['mostly-true', 'half-
true', 'true']:
            df.at[i, 'Label'] = 1
        else:
            df.at[i, 'Label'] = 0
    return df

train_news = convert_labels(train_news)
test_news = convert_labels(test_news)
valid_news = convert_labels(valid_news)
```

FeatureSelection.py (Assumed, Based on Context)

```
from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer

countV = CountVectorizer()
tfidf_ngram = TfidfVectorizer(stop_words='english',
ngram_range=(1, 4), use_idf=True, smooth_idf=True)
```

prediction.py (Corrected)

```
import joblib
import pandas as pd

load_model = joblib.load('final_model.sav')
```

```
def predict_statement(statement):
    prediction = load_model.predict([statement])[0]
    return "True" if prediction == 1 else "False"

while True:
    statement = input("Enter news statement (or
'exit' to quit): ")
    if statement.lower() == 'exit':
        break
    print(f"Prediction:
{predict_statement(statement)}")
```

app.py (For Streamlit Deployment)

```
import streamlit as st
import joblib

st.title("Fake News Detection")
st.write("Enter a news statement to classify as
True or False.")

model = joblib.load('final_model.sav')
statement = st.text_area("News Statement", "Enter
statement here...")

if st.button("Predict"):
    if statement:
        prediction = model.predict([statement])[0]
        label = "True" if prediction == 1 else
"False"
        st.success(f"Prediction: {label}")
    else:
        st.error("Please enter a statement.")
```

requirements.txt

```
scikit-learn==1.5.1
pandas==2.2.2
numpy==1.26.4
matplotlib==3.8.4
seaborn==0.13.2
nltk==3.8.1
```

```
joblib==1.4.2
streamlit==1.38.0
wordcloud==1.9.3
```

---

14. Future Scope

1. ***Incorporate Deep Learning Models***: *Enhance the model by integrating deep learning architectures like BERT or LSTM, which can capture complex semantic relationships in text. This would require a GPU for training and could improve the F1 score to 0.85 or higher, making the model suitable for critical applications.*

2. ***Real-Time Data Integration***: *Develop a pipeline to scrape and process real-time news from platforms like X or news APIs, enabling continuous model updates and adaptation to emerging misinformation trends. This would require robust data pipelines and cloud infrastructure.*

3. ***Multimodal Analysis***: *Extend the model to incorporate multimodal data, such as images or metadata (e.g., source credibility, author), to improve detection accuracy. This would involve combining text and image processing models, potentially increasing complexity but enhancing robustness.*

---

15. Team Members and Roles

1. Aathi S:
• Responsibilities: Led and executed all major tasks, including:
• Data Cleaning: Normalized text, handled missing values, and removed duplicates
for LIAR and FakeNewsNet datasets.
• EDA: Conducted univariate and bivariate analyses, visualized label distributions, and identified key patterns.
• Feature Engineering: Created TF-IDF vectors, BERT embeddings, and sentiment features.

• *Model Development: Built and evaluated Logistic Regression and BERT models, achieving 85% F1-Score.*
• *Documentation and Reporting: Drafted methodology, results, and submission template.*

2. Keerthivasan V:
• *Responsibilities: Provided feedback during team discussions and assisted in reviewing visualizations.*

3. Vimalraj R:
• *Responsibilities: Contributed to discussions on Streamlit interface design and user experience.*

4. Rithik:
• *Responsibilities: Supported documentation by reviewing the final write-up for clarity.*