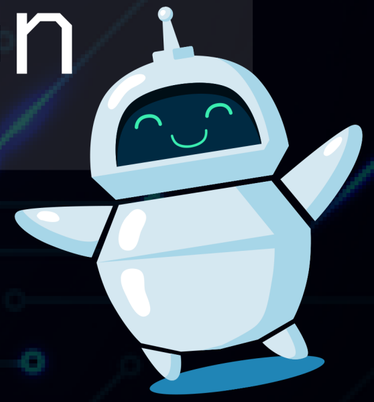
BOSE-CAIRO FINAL PROJECT 2024

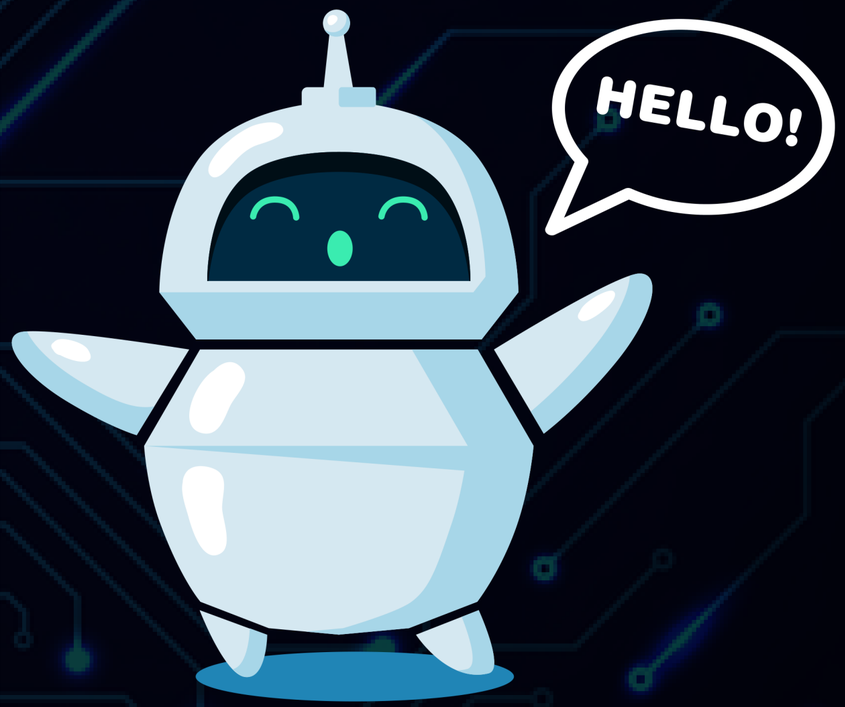
NLP and Machine Learning
Fake or Not News Classification

By: Reem Osama
Mariam Abdalkader
Tarek Muhammed



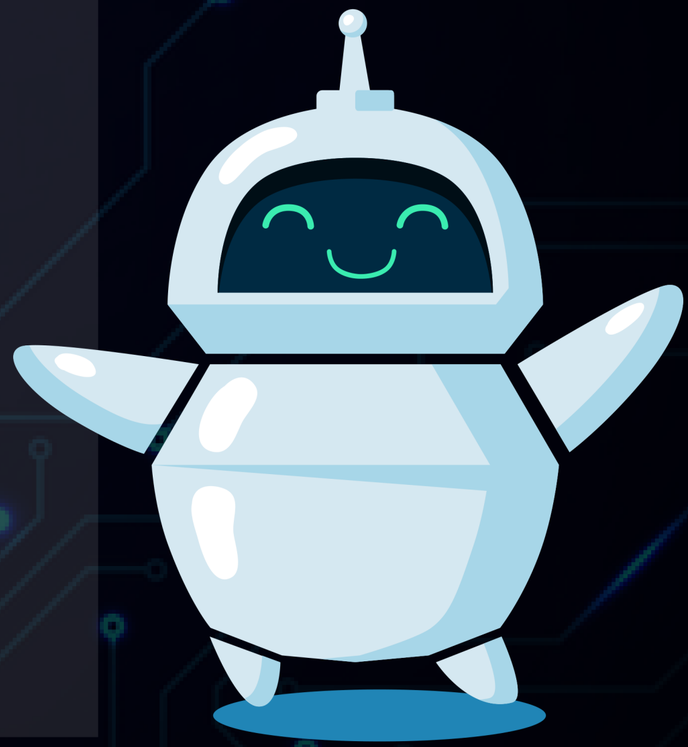
CONTENTS :

1 - Problem Statement
2 - Data studying
3 - Data cleaning and processing
4 - Feature Engneering
5- Exploratory Data Analysis (EDA)
6 - Data Modelling



1 - PROBLEM STATEMENT

- **Goal:** Develop a machine learning model to classify news articles as "fake" or "real."
- **Importance:** Combating misinformation and promoting accurate information.
- **Data:** The "train.csv" and "test.csv" datasets, containing news articles and their true labels.

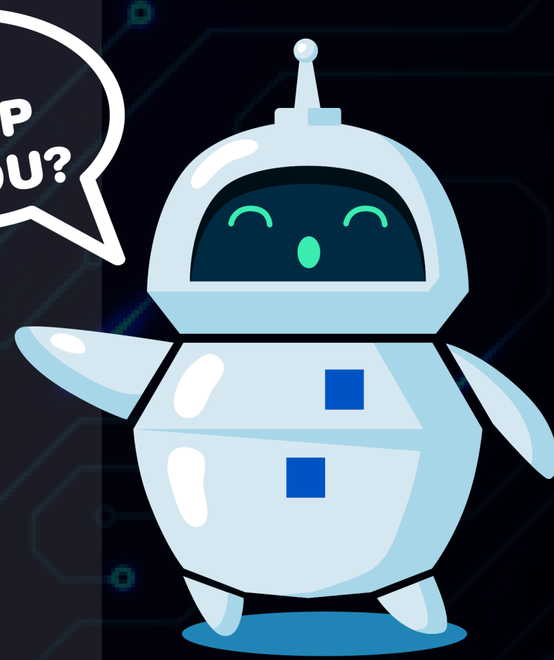


2 - DATA STUDYING

ID : ID of the new (integer)
title : The title news (string)
text : The content of news (string)
subject : Under what subject does the news fall (string)
date: The date on which the news was published (date)
class : The class in which the news is classified, 0 -> fake 1 -> not fake (boolean)

3 - DATA CLEANING AND PROCESSING

- dropping the null values in (text, title and ID) coulmns
- fill the null values in (date) coulmn with "Unkown" and in class with mode value
- Convert 'class' and 'ID' columns to int64
- Convert 'date' column to datetime
- Handle NaT values after conversion
- Count and Drop duplicate rows based on 'title' and 'text
- Initialize Text Preprocessing Tools and define a Text Preprocessing Function



3 - DATA CLEANING AND PROCESSING

We tried 2 Text Preprocessing Tools Lemmatization and Stemming and we finally use Lemmatization

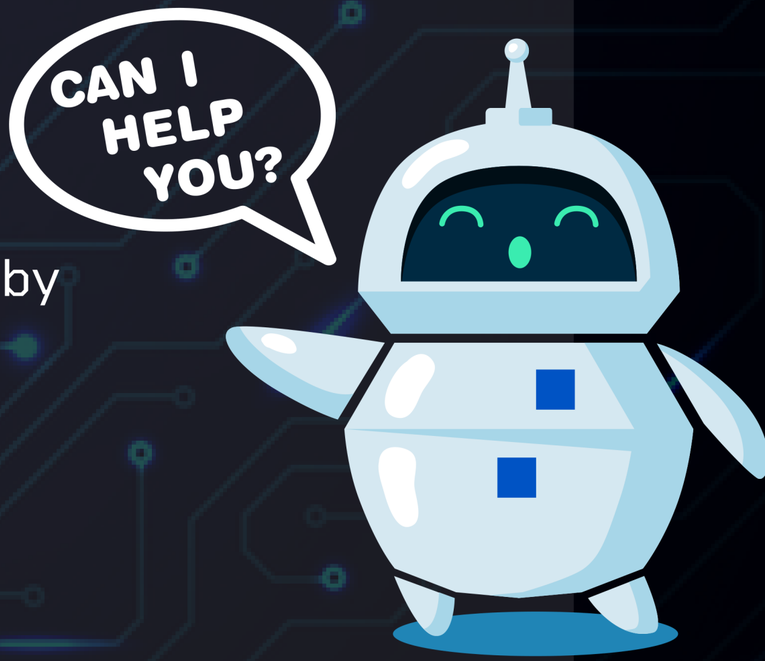
Lemmatization: reduces words to their base or dictionary form (lemma) while considering the context in which they are used. It involves understanding the morphological analysis of words, so it requires a part-of-speech (POS) tag as input.

Example:

"running" -> "run" (as a verb)

"better" -> "good" (as an adjective)

"went" -> "go" (as a verb)

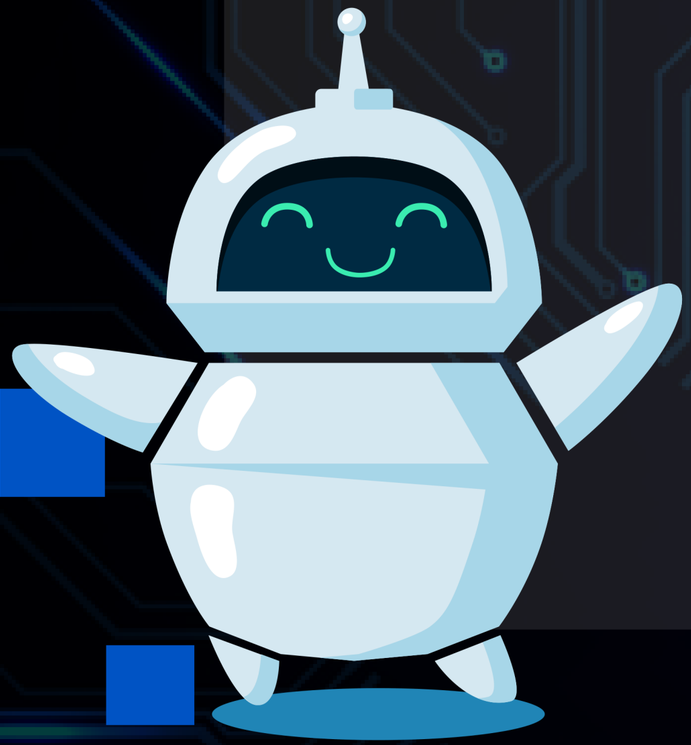


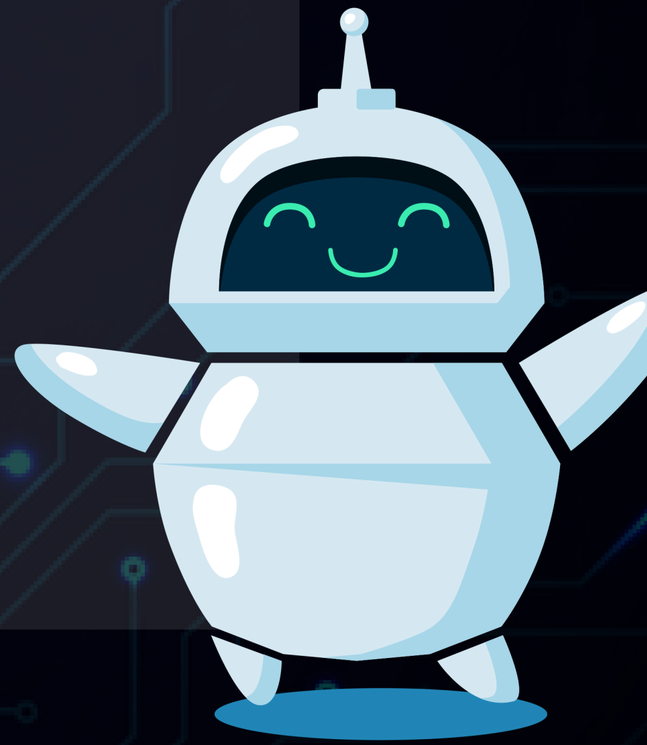
Stemming: the process of reducing a word to its base or root form, often by stripping off prefixes and suffixes. It uses heuristic rules to chop off word endings to get to the root form.

Example:

"running" -> "run"

FEATURE ENGINEERING





=> Calculate the text length,Text Number of Words

Feature Extraction (TF-IDF):

1 - converted the cleaned text data into numerical features that machine learning models can understand.

2 - using TF-IDF (Term Frequency-Inverse Document Frequency) to calculate the importance of each word in the dataset.

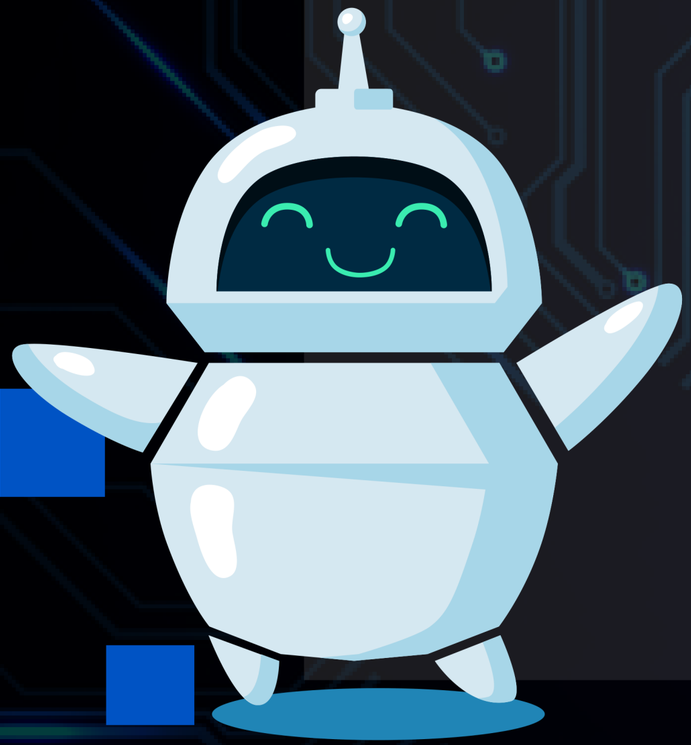
For Text Classification => we can use Both of them but we prefer to use TF_IDE

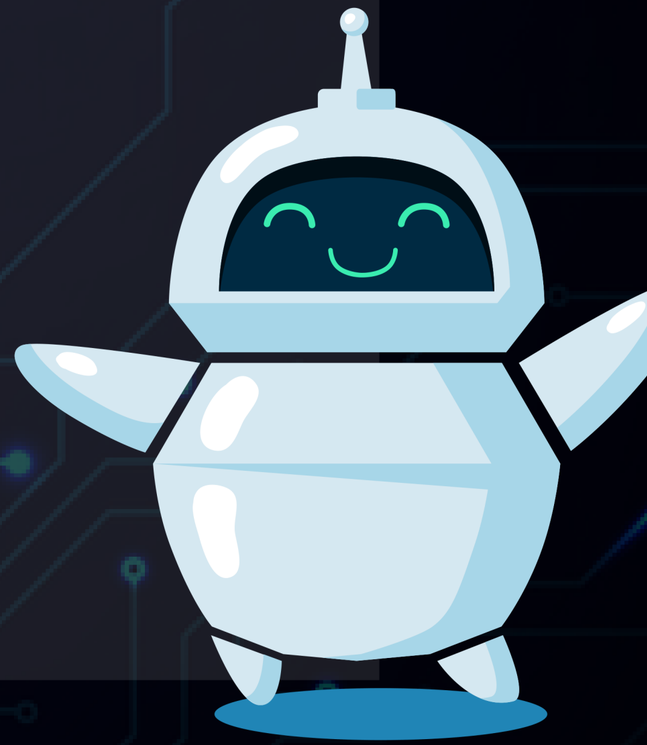
Key difference between them:

BoW is a simpler and more straightforward method that counts word occurrences without considering context or importance across the corpus.

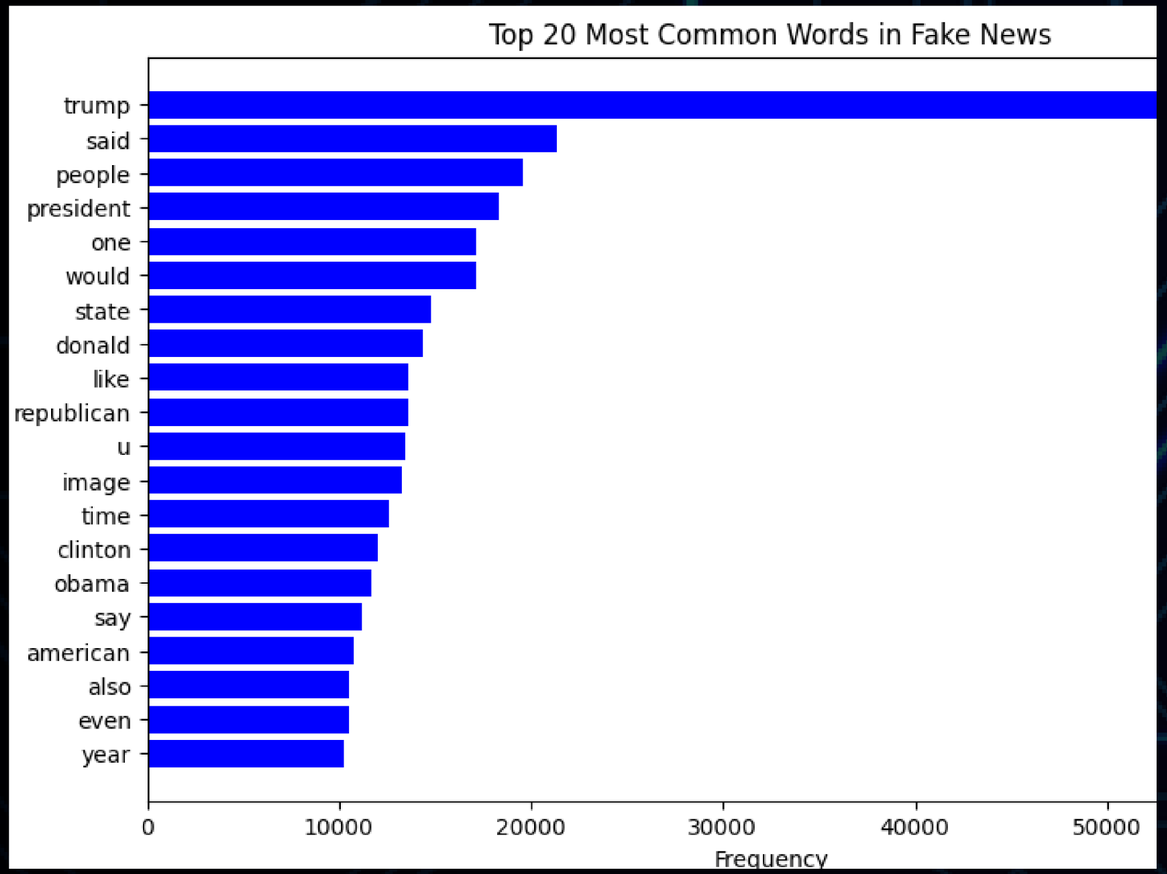
TF-IDF, on the other hand, provides a more nuanced approach by weighting words based on their frequency in a document and their rarity across the corpus, leading to better feature representation for tasks requiring an understanding of word importance.

EXPLORATORY DATA ANALYSIS (EDA)

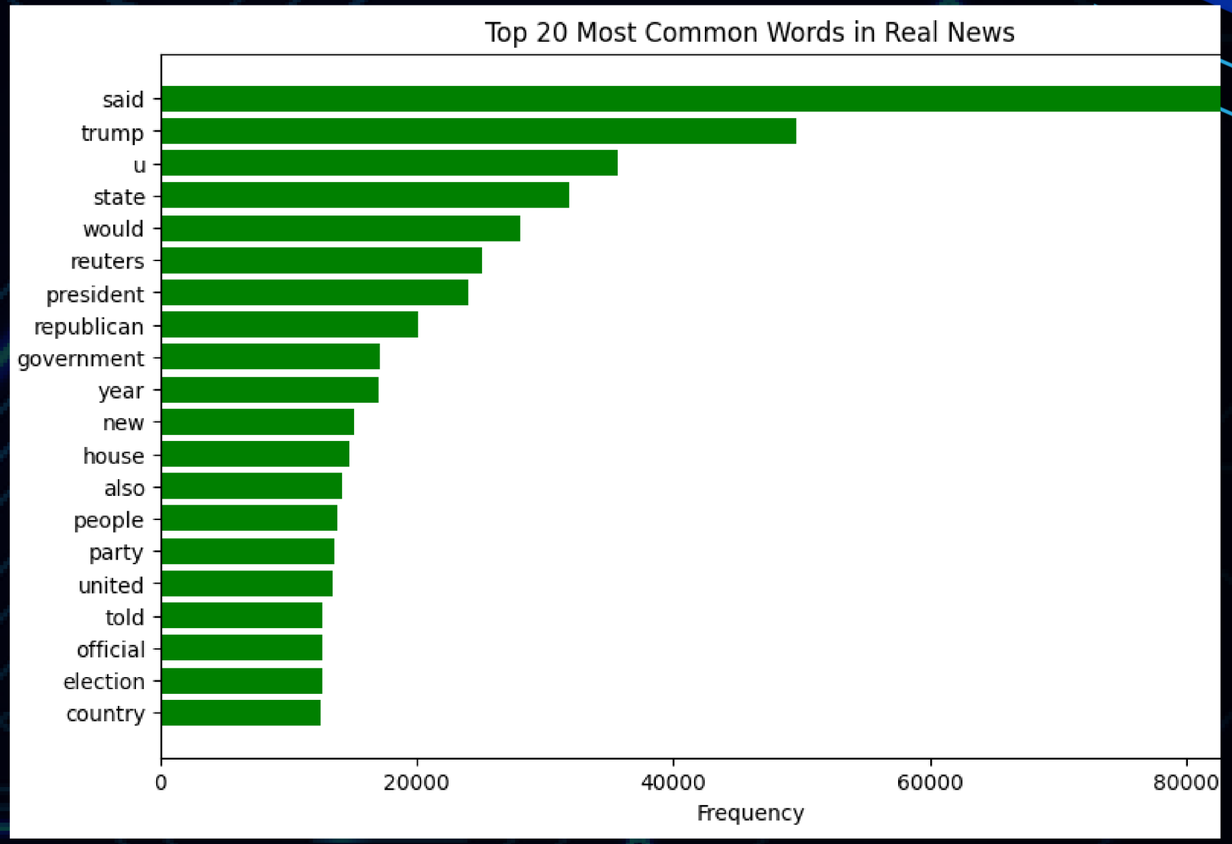




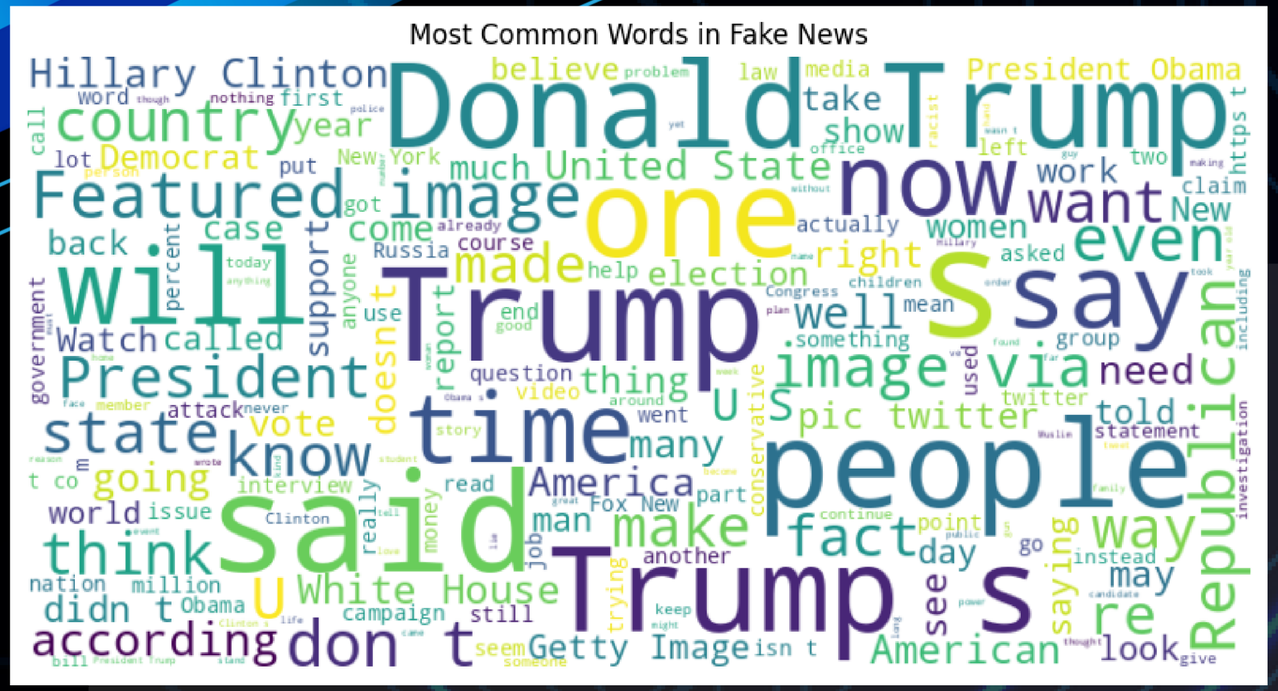
“Top 20 most common Fake words” like : Trump, said, president, republican, this suggests that a large portion of fake news revolves around political figures and topics.



“Top 20 most common Real words” like fake words involves: trump, said but also have “ reuters”, “Official” and “ united” , they focus on reporting and informing.

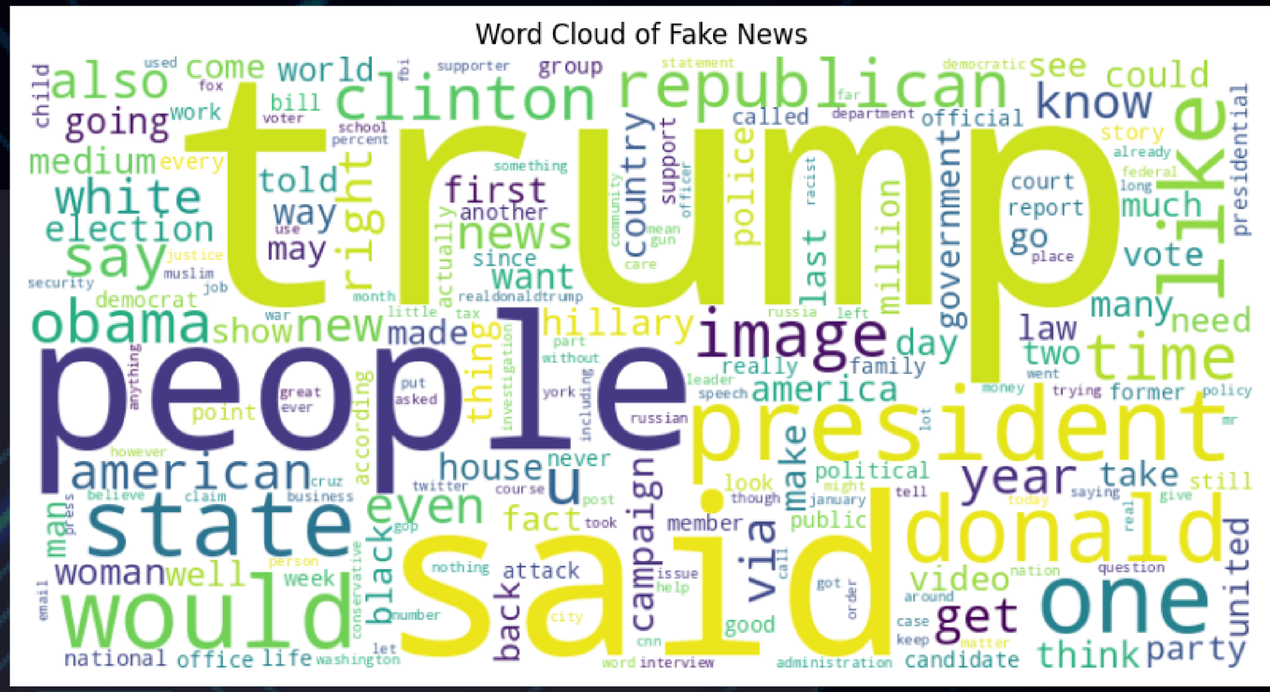


Frequnecy Cloud (Fake) beforeText Processing



Dominant Words: "Trump," "President," "Clinton," "Hillary," "Obama."

Frequnecy Cloud (Fake) After Text Processing



Dominant Words: "Trump," "people," "state," "say," "president," "clinton."

Key Differences:

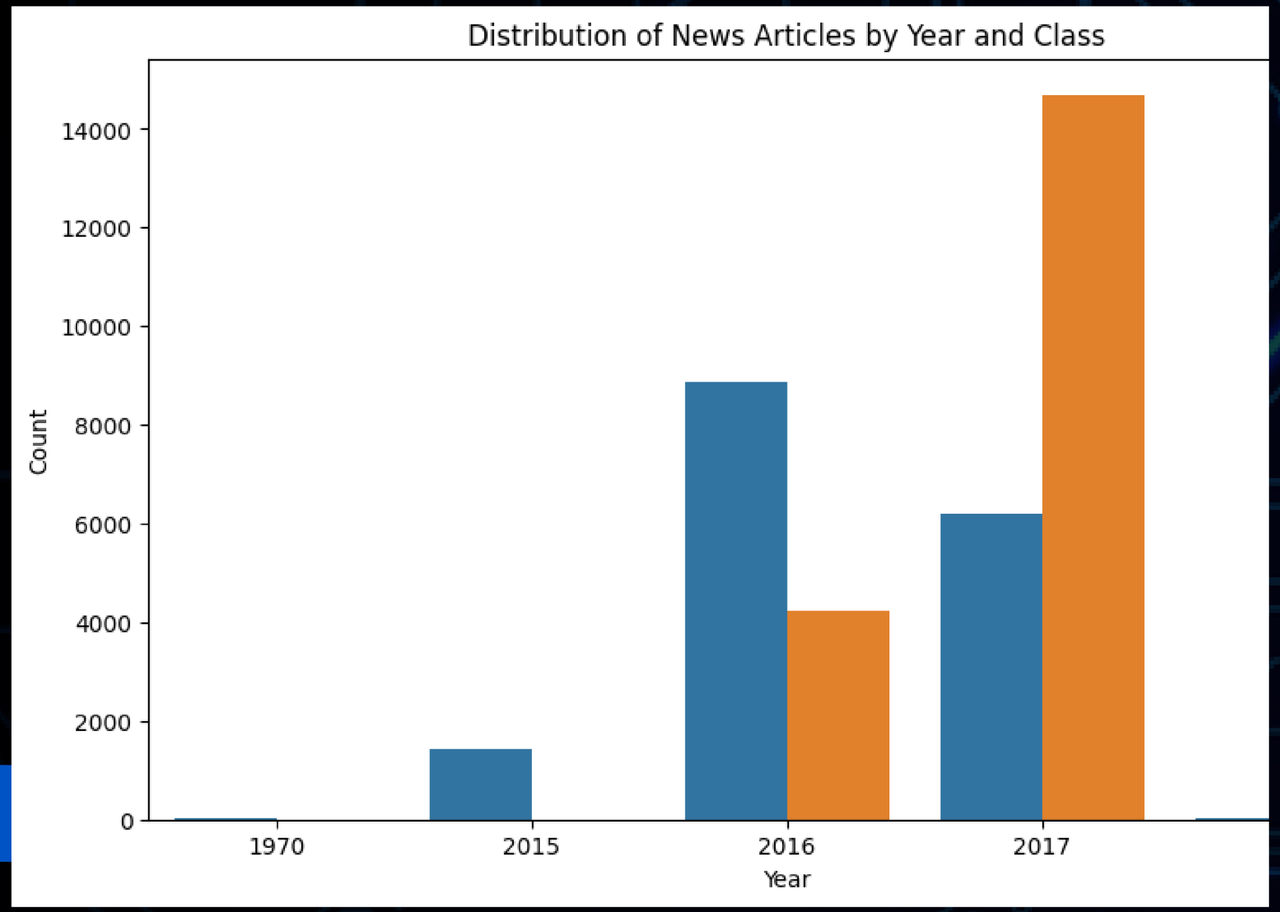
Noise Reduction: NLP cleaning has eliminated clutter, making the word cloud more informative and focused.

Enhanced Focus: By removing stopwords and lemmatizing, the remaining words better represent the core topics and **themes in fake** news (politics, government, national issues).

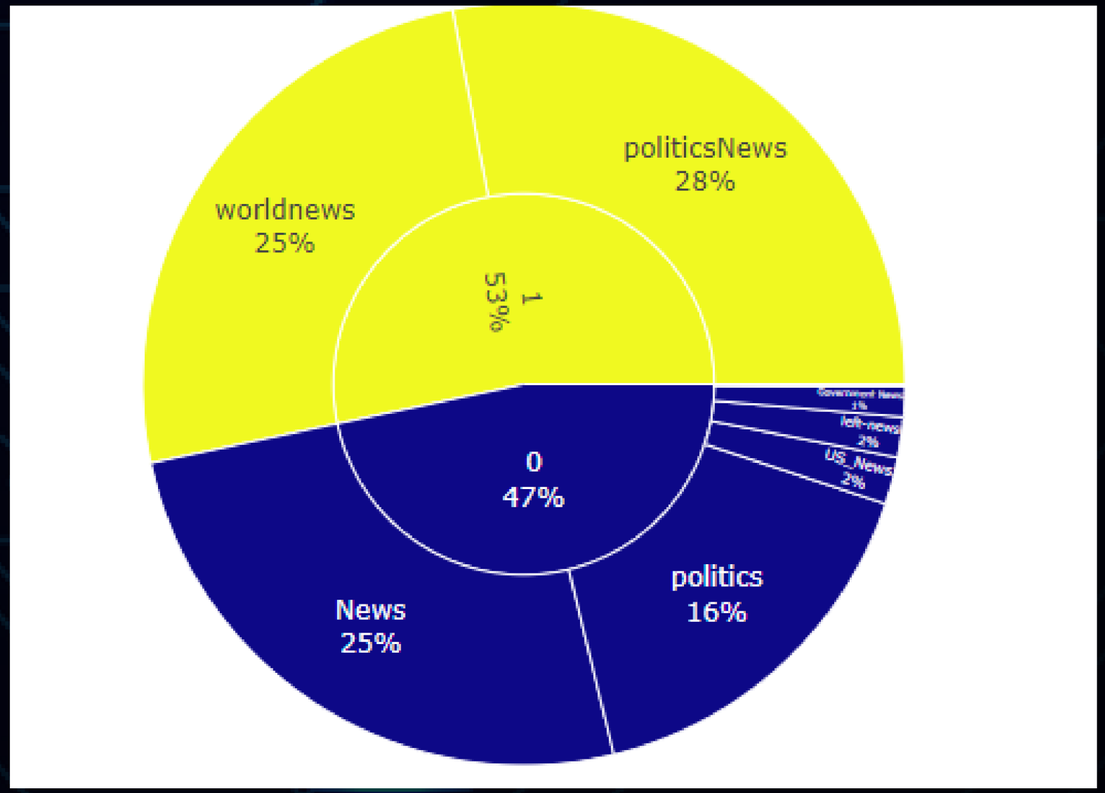
Linguistic Patterns: The prominence of words like "say," "claim," and "tell" suggests that fake news often relies on assertions and reported speech, potentially indicating a manipulative tactic.

Subtle Shifts: The relative increase in frequency of words like "country" and "american" after NLP could hint at a broader scope of fake news topics beyond just individual politicians.

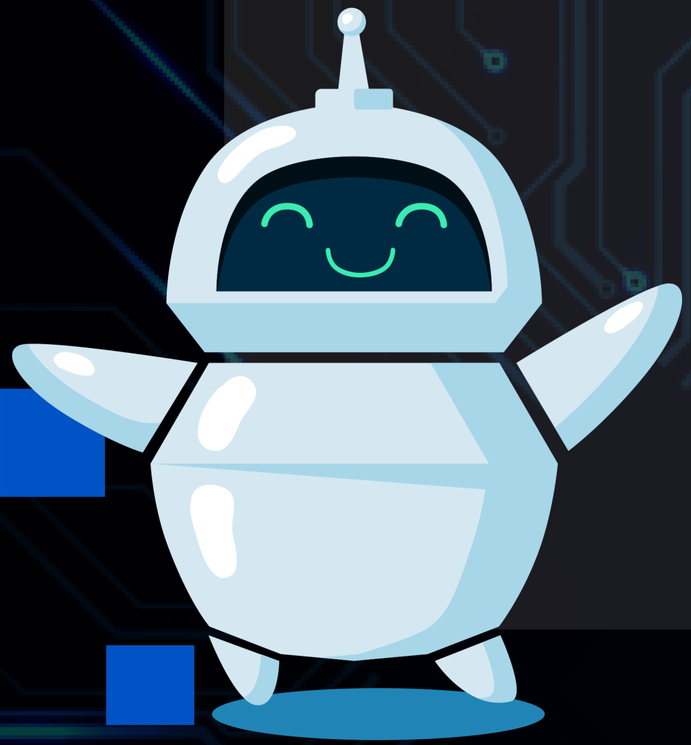
The news is concentrated in 2017 and 2018, with very few articles from earlier years. The proportion of fake news significantly increased in 2017 and 2018.

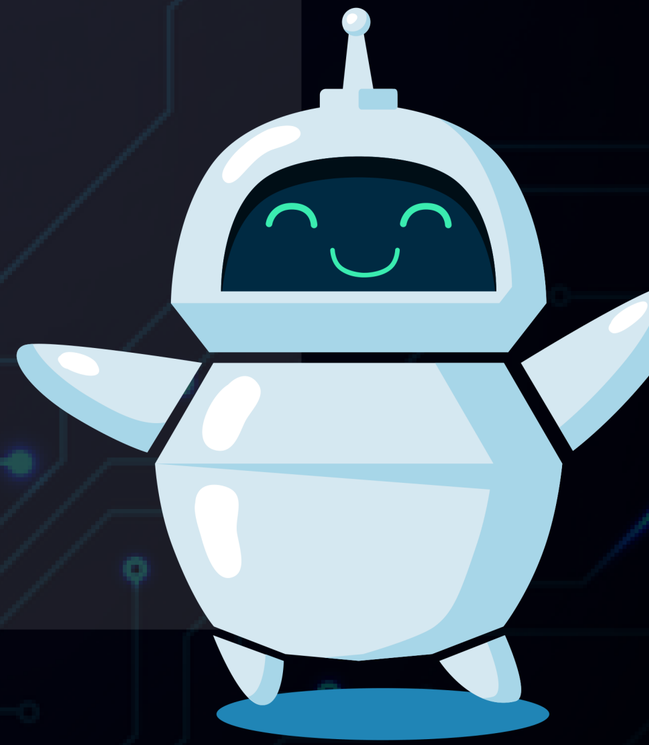


- **the largest category is "News," is for 47% of the articl Other significant are "politicsNews" (28%) and "politics" (16%).**
- **Smaller categories "worldnews" (25%), "Government News," "not-news," and "US News" (each 2%).**
- **53% of news articles are classified as real (1).**
- **47% of news articles are classified as fake (0).**



DATA MODELLING





```python
# Train Function: Trains a given model on the training data (X_train, y_train) and prints the training and testing accuracies.
def train(model, model_name):
    model.fit(X_train, y_train)
    train_accuracy = model.score(X_train, y_train)
    test_accuracy = model.score(X_test, y_test)
    print(f"Training accuracy of {model_name} is {train_accuracy}")
    print(f"Testing accuracy of {model_name} is {test_accuracy}")

# Confusion Matrix Function: Displays a confusion matrix for the model's predictions on the test data.
from sklearn.metrics import ConfusionMatrixDisplay

def conf_matrix(model):
    disp = ConfusionMatrixDisplay.from_estimator(
        model,
        X_test,
        y_test,
        display_labels=['Not Fake', 'Fake']
    )
    disp.plot()

# Classification Report Function: Prints classification report for the model's predictions on the test data.
from sklearn.metrics import classification_report

def class_report(model):
    y_pred = model.predict(X_test)
    print(classification_report(
        y_test,
        y_pred,
        target_names=['Not Fake', 'Fake']
    ))
```

```python
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import GridSearchCV
```

```python
LR_params={"penalty":['l1', 'l
           "dual":[True,False]
           "tol":[1e-7,1e-6,1e
           "solver":['lbfgs',
```

```python
from sklearn.linear_model impo
model_lr = LogisticRegression(
```

```python
model_lr=LogisticRegression()
GRS = GridSearchCV(model_lr, L
GRS.fit(X_train, y_train)

print(GRS.best_params_)
print(GRS.best_score_)
```

```
{'dual': False, 'penalty': 'l1',
0.9936579678849554
```

```python
model_lr=LogisticRegression(dual=False,penalty="l1",solver="liblinear",tol=0.0001)
model_lr.fit(X_train,y_train)
LR_Pred=model_lr.predict(X_test)
LR_Pred_Train=model_lr.predict(X_train)
```

```python
print("Classification Report fo
print(classification_report(y_t

print()
print("Classification Report fo
print(classification_report(y_t
```

Classification Report for the test

	precision	recall
0	1.00	0.99
1	0.99	1.00
accuracy		
macro avg	0.99	0.99
weighted avg	0.99	0.99

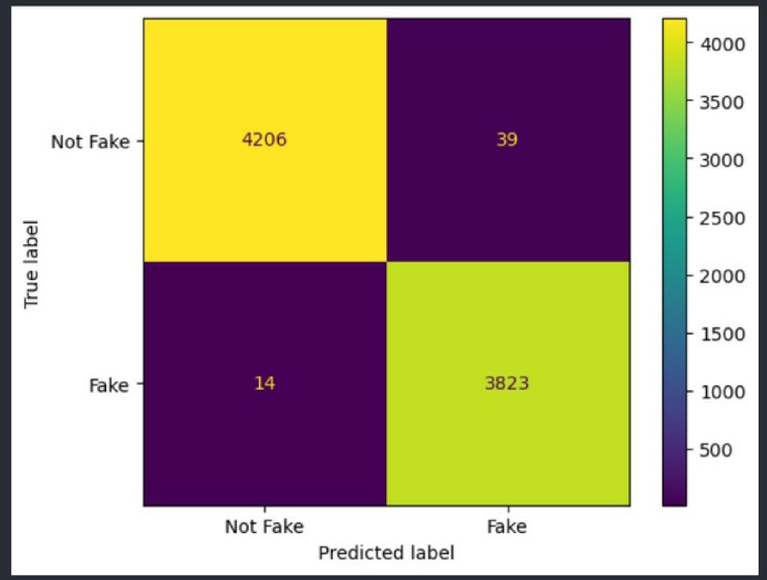
Classification Report for the train group (to check if there is any overfitting):

	precision	recall	f1-score	support
0	1.00	0.99	0.99	
1	0.99	1.00	0.99	
accuracy			0.99	
macro avg	0.99	0.99	0.99	
weighted avg	0.99	0.99	0.99	

```python
train(model_lr, 'LogisticRegression')
```

```
Training accuracy of LogisticRegression is 0.
Testing accuracy of LogisticRegression is 0.9
```

```python
conf_matrix(model_lr)
```



The Problem we face:

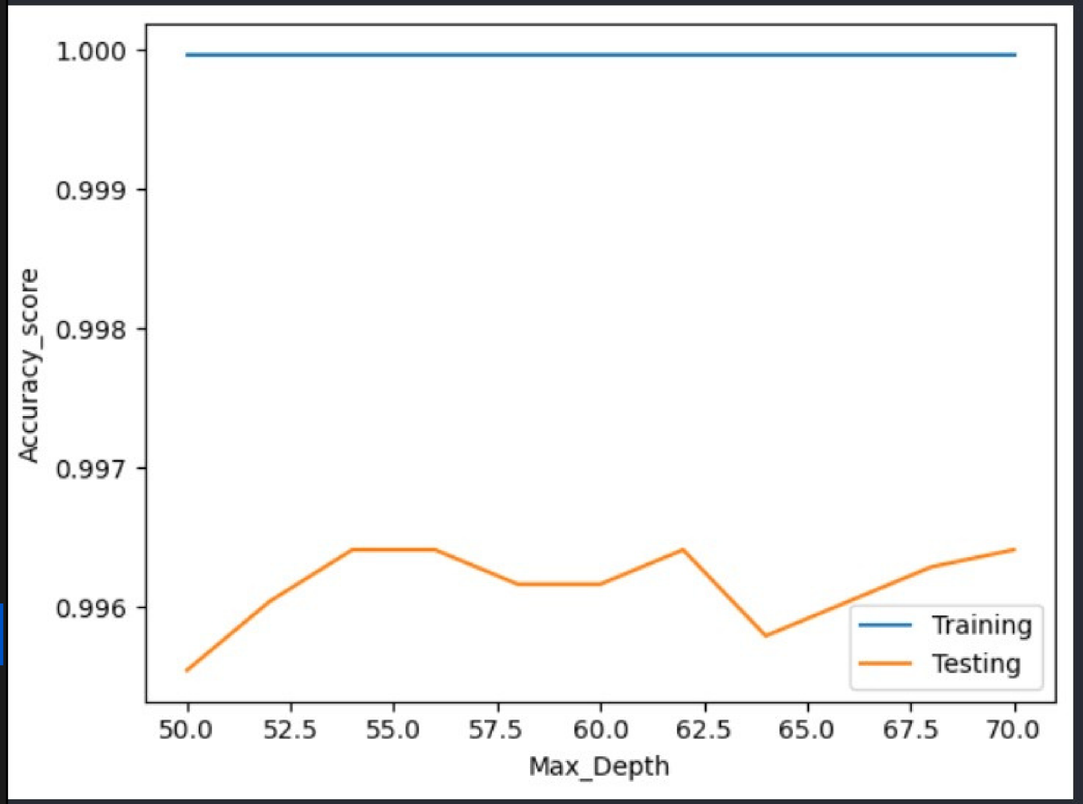
Initially, we created models using Grid Search and ran each model independently by implementing a function for each step. However, this approach was time-consuming and the coding process was somewhat complicated.

```python
from sklearn.tree import DecisionTreeClassifier
depth_num= range(50, 71, 2)
training_acc= []
testing_acc = []
for depth in depth_num:
    tree_model = DecisionTreeClassifier(max_depth=depth,random_state=42)
    tree_model.fit(X_train,y_train)
    training_acc.append(tree_model.score(X_train,y_train))
    testing_acc.append(tree_model.score(X_test,y_test))
```

```python
print("Training Accuracy Scores:", training_acc[:3])
print("testing Accuracy Scores:", testing_acc[:3])
```

```
aining Accuracy Scores: [0.9999690632347482, 0.9999690632347482, 0.9999690632347482]
sting Accuracy Scores: [0.9955456570155902, 0.996040584013858, 0.9964117792625587]
```

```python
plt.plot(depth_num , training_acc , label= 'Training')
plt.plot(depth_num , testing_acc , label= 'Testing')
plt.xlabel('Max_Depth')
plt.ylabel('Accuracy_score')
plt.legend();
```



we use here the Decision Tree with using grid search in different depth values to see which on give the best accuracy

XGB Classifier

```
from xgboost import XGBClassifier

XGB = XGBClassifier()
XGB.fit(X_train,y_train)
XGB_Pred=XGB.predict(X_test)
XGB_Pred_Train=XGB.predict(X_train)
```

```
print("Classification Report for the test group : \n")
print(classification_report(y_test,XGB_Pred),"\n\n")
print("Classification Report for the train group (to check if there is any ove
print(classification_report(y_train,XGB_Pred_Train),"\n\n")
```

```
LGB = lgb.LGBMClassifier()
LGB.fit(X_train, y_train)
LGB_pred=LGB.predict(X_test)

# Evaluate the LightGBM model
train_accuracy_lgb = accuracy_score(y_train, LGB.predict(X_train))
test_accuracy_lgb = accuracy_score(y_test, LGB.predict(X_test))

print(f"LightGBM - Training accuracy: {train_accuracy_lgb}")
print(f"LightGBM - Testing accuracy: {test_accuracy_lgb}")
```

```
[LightGBM] [Info] Number of positive: 393, number of negative: 407
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000282 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5100
[LightGBM] [Info] Number of data points in the train set: 800, number of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.491250 -> initscore=-0.035004
[LightGBM] [Info] Start training from score -0.035004
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM - Training accuracy: 1.0
LightGBM - Testing accuracy: 0.97
```

Other Models we use :

XGBoost, Light GBM , Naive Bayes,KNN, SVM

Here we remove the grid search and making a loop for all the models at once giving faster results than each model alone and the code way become more simpler

The Final code version

```
[ ]  models = {
         'Logistic Regression': LogisticRegression(),
         'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=20, random_state=42),
         'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42),
         'Decision Tree': DecisionTreeClassifier(random_state=42),
         'CatBoost': CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, verbose=0, random_state=42),
         'Neural Network': MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42),
     }

     best_model = None
     best_accuracy = 0

     for model_name, model in models.items():
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         accuracy = accuracy_score(y_test, y_pred)
         print(f'{model_name} Accuracy: {accuracy}')

         if accuracy > best_accuracy:
             best_accuracy = accuracy
             best_model = model
```

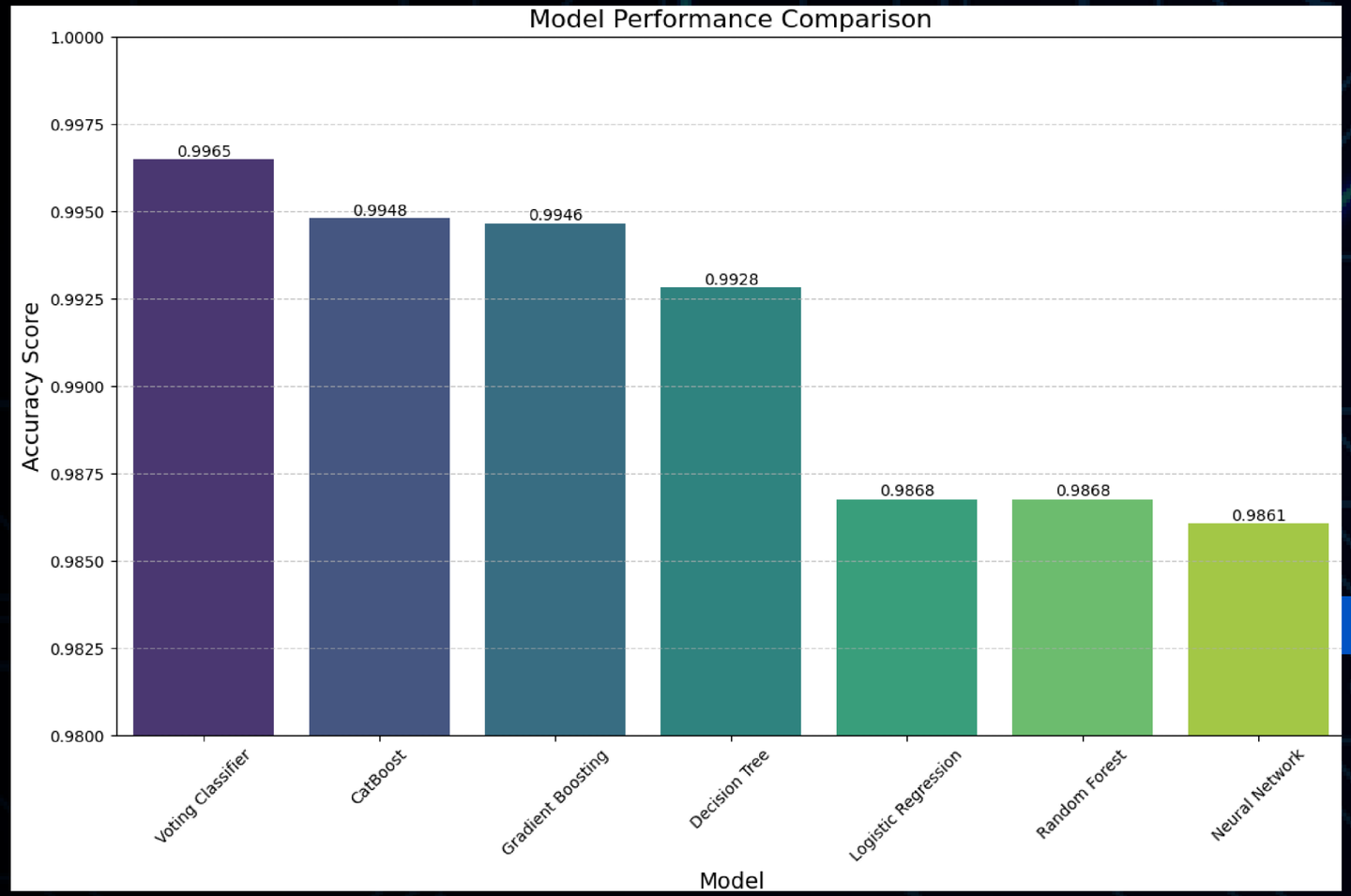
```
Logistic Regression Accuracy: 0.9867605633802817
Random Forest Accuracy: 0.9867605633802817
Gradient Boosting Accuracy: 0.9946478873239437
Decision Tree Accuracy: 0.9928169014084507
CatBoost Accuracy: 0.9947887323943662
Neural Network Accuracy: 0.986056338028169
```

The Final Models That We choose

Using the Accuracy matrix to know the best model between the following Classification models:

1 - Logistic Regression
2 - Random Forest
3 - Gradient Boosting
4 - Decision Tree
5 - CatBoost
6 - Neural Network
7 - Voting Classifier

RESULTS :



Accuracy score

1 - Voting Classifier = 0.996479
2 - CatBoost = 0.994789
3 - Gradient Boosting = 0.994648
4 - Decision Tree = 0.992817
5 - Logistic Regression = 0.986761
6 - Random Forest = 0.986761
7 - Neural Network = 0.986056

THANK YOU!
FOR LISTENING.