

## Implementation of Project

Download Project Files:

Download all project files from GitHub link provided.

extract and Open the Project in VS Code:

Launch VS Code.

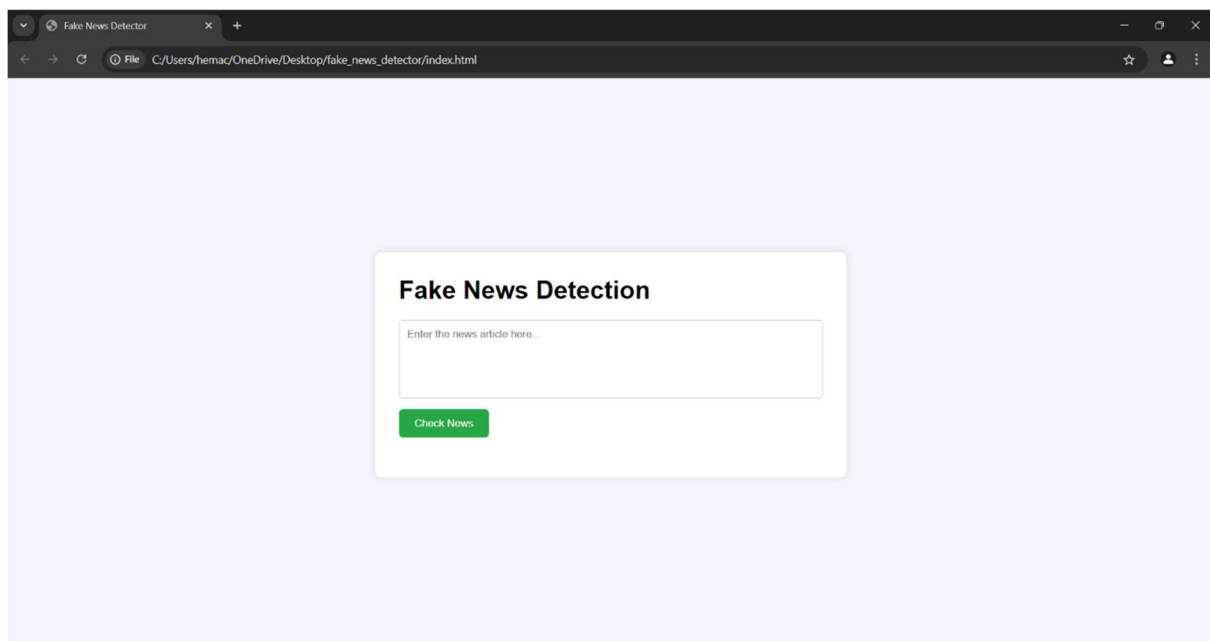
Open the project folder by selecting File > Open Folder and choosing the folder containing your downloaded files.

Open index.html in VS Code:

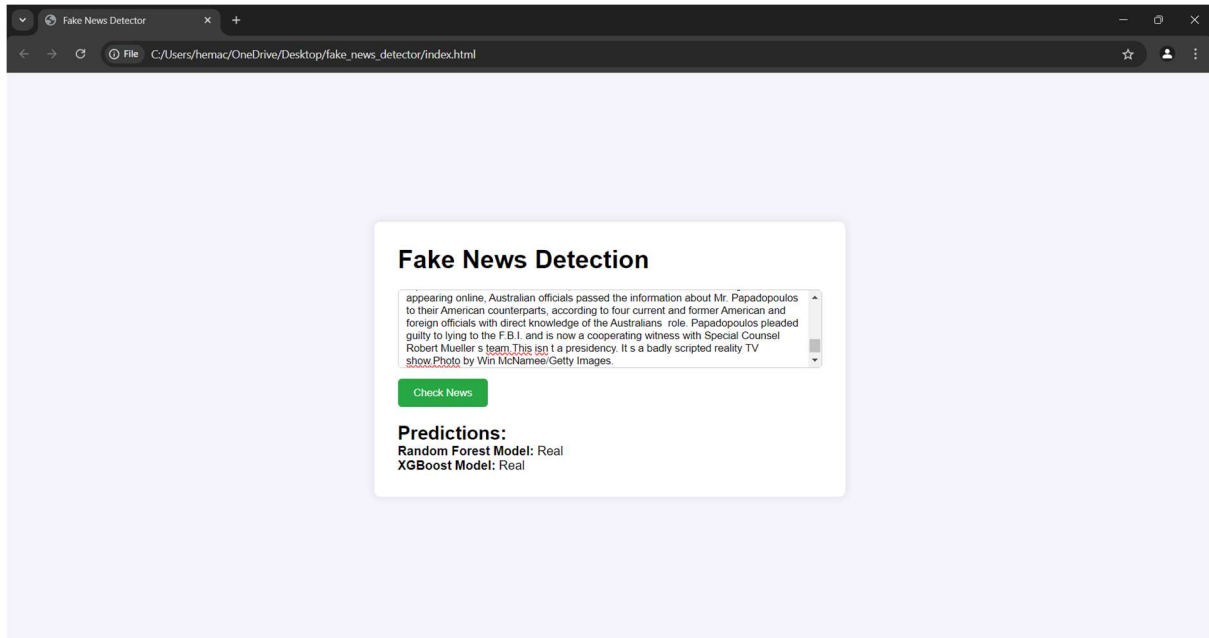
Locate the index.html file in the VS Code file explorer.

Right-click on index.html and Run it by clicking start debugging it will open in a web browser and you can take any news article from the True (1) and Fake (1) CSV files and paste it, it will display the output.

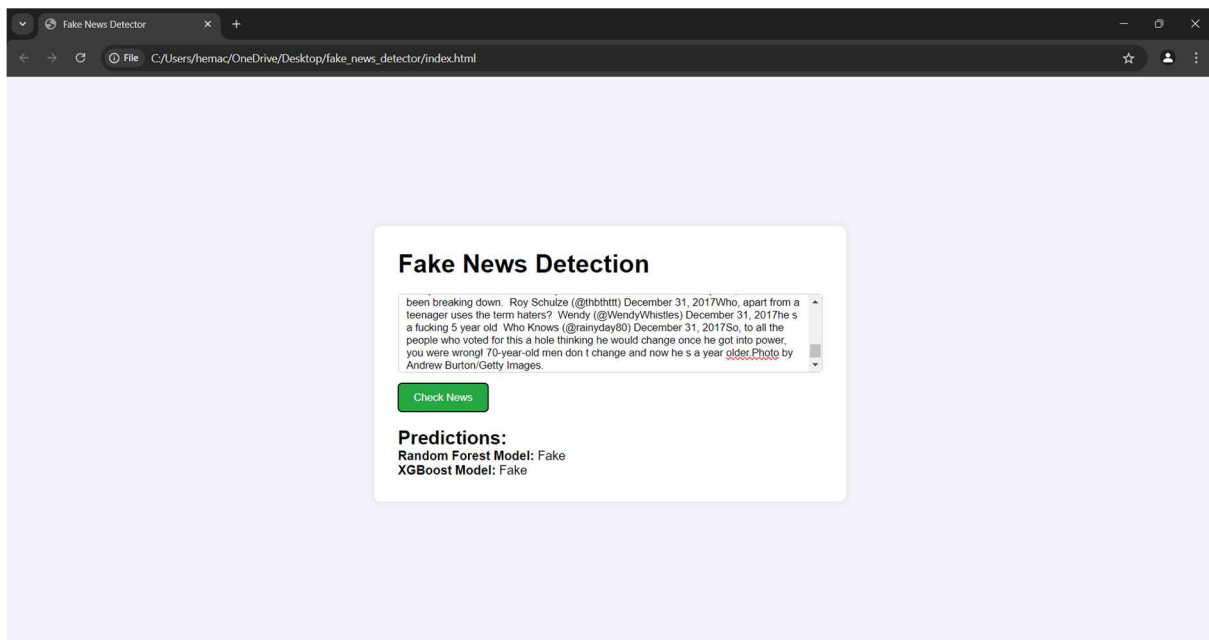
**here is the outputs for how the project works:**



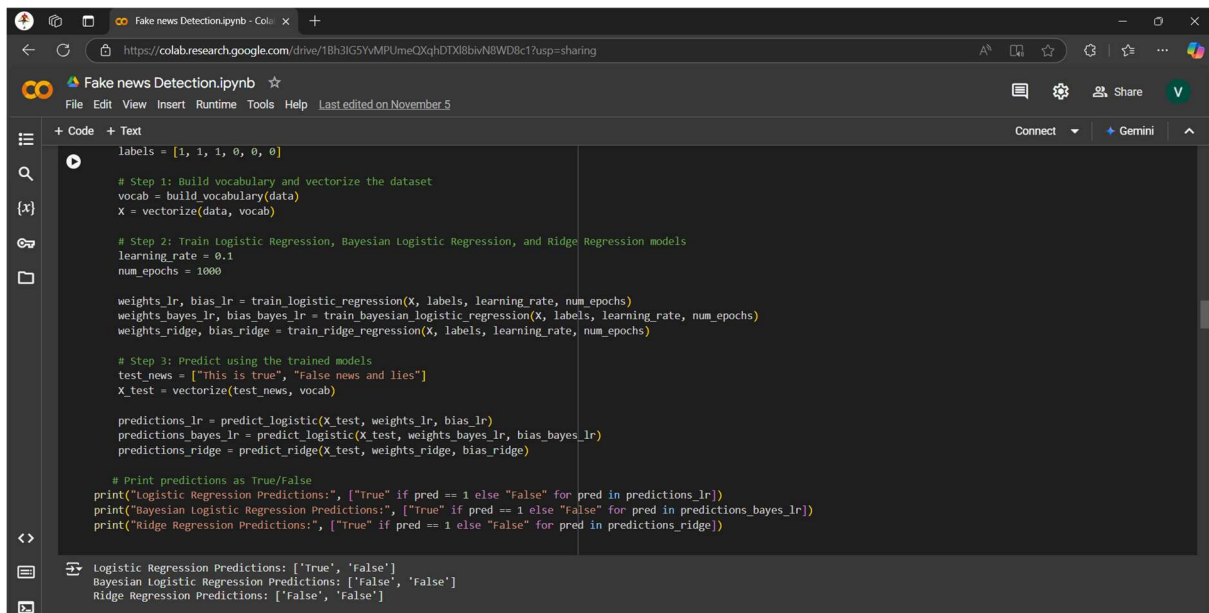
## Prediction for Real news using Random Forest Model and XGBoost Model:



## Prediction for Fake news using Random Forest Model and XGBoost Model:



## Prediction for Fake news using Logistic Regression, Bayesian Logistic Regression and Ridge Regression Model in Collab : These 3 Regression models can be implemented in the google collab.



```
Labels = [1, 1, 1, 0, 0, 0]

# Step 1: Build vocabulary and vectorize the dataset
vocab = build_vocabulary(data)
X = vectorize(data, vocab)

# Step 2: Train Logistic Regression, Bayesian Logistic Regression, and Ridge Regression models
learning_rate = 0.1
num_epochs = 1000

weights_lr, bias_lr = train_logistic_regression(X, labels, learning_rate, num_epochs)
weights_bayes_lr, bias_bayes_lr = train_bayesian_logistic_regression(X, labels, learning_rate, num_epochs)
weights_ridge, bias_ridge = train_ridge_regression(X, labels, learning_rate, num_epochs)

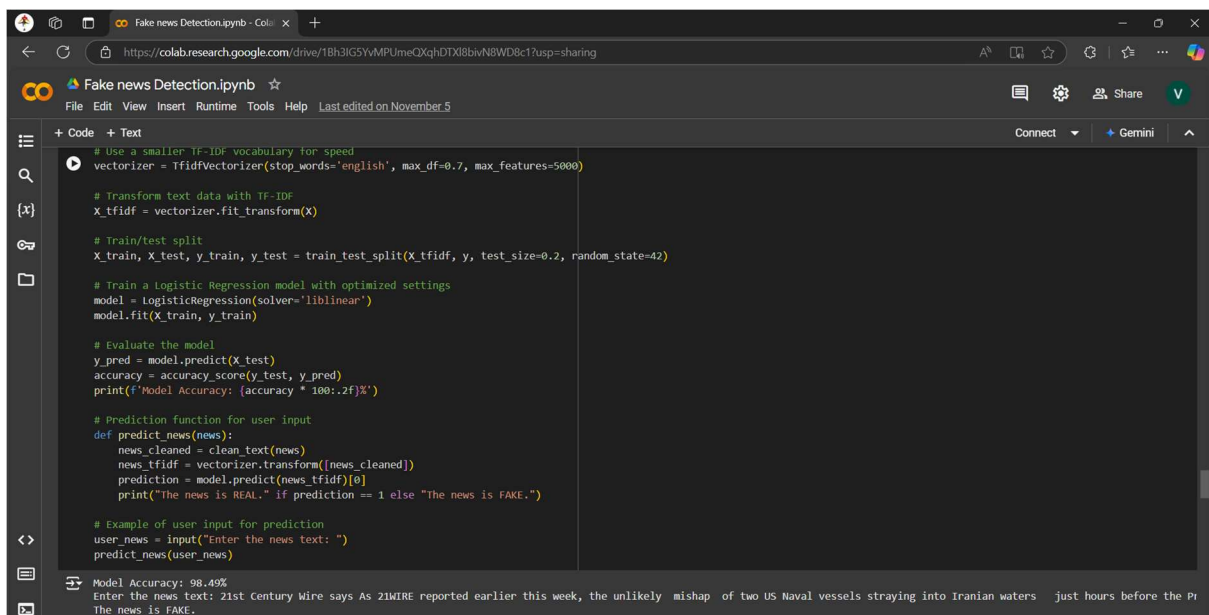
# Step 3: Predict using the trained models
test_news = ["This is true", "False news and lies"]
X_test = vectorize(test_news, vocab)

predictions_lr = predict_logistic(X_test, weights_lr, bias_lr)
predictions_bayes_lr = predict_logistic(X_test, weights_bayes_lr, bias_bayes_lr)
predictions_ridge = predict_ridge(X_test, weights_ridge, bias_ridge)

# Print predictions as True/False
print("Logistic Regression Predictions:", ["True" if pred == 1 else "False" for pred in predictions_lr])
print("Bayesian Logistic Regression Predictions:", ["True" if pred == 1 else "False" for pred in predictions_bayes_lr])
print("Ridge Regression Predictions:", ["True" if pred == 1 else "False" for pred in predictions_ridge])
```

Logistic Regression Predictions: ['True', 'False']  
Bayesian Logistic Regression Predictions: ['False', 'False']  
Ridge Regression Predictions: ['False', 'False']

## For Fake news:



```
# Use a smaller TF-IDF vocabulary for speed
vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7, max_features=5000)

# Transform text data with TF-IDF
X_tfidf = vectorizer.fit_transform(X)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

# Train a Logistic Regression model with optimized settings
model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)

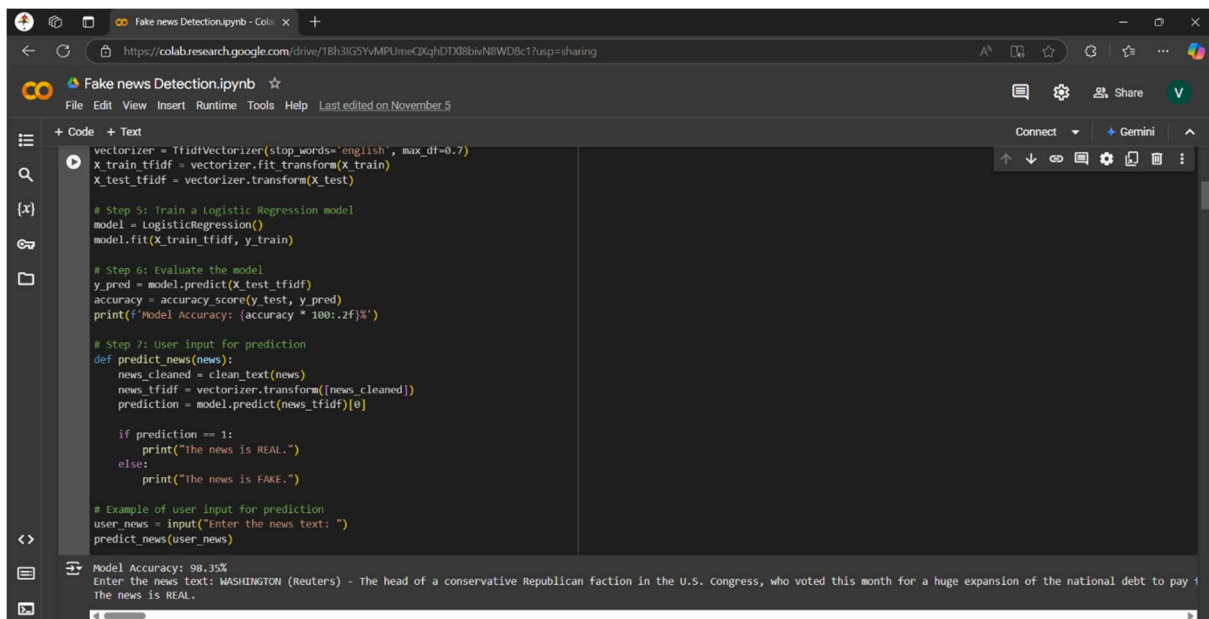
# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Prediction function for user input
def predict_news(news):
    news_cleaned = clean_text(news)
    news_tfidf = vectorizer.transform([news_cleaned])
    prediction = model.predict(news_tfidf)[0]
    print("The news is REAL." if prediction == 1 else "The news is FAKE.")

# Example of user input for prediction
user_news = input("Enter the news text: ")
predict_news(user_news)
```

Model Accuracy: 98.49%  
Enter the news text: 21st Century Wire says AS 21WIRE reported earlier this week, the unlikely mishap of two US Naval vessels straying into Iranian waters just hours before the P  
The news is FAKE.

## For Real News:



```
vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Step 5: Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)

# Step 6: Evaluate the model
y_pred = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print('Model Accuracy: {accuracy * 100:.2f}%')

# Step 7: User input for prediction
def predict_news(news):
    news_cleaned = clean_text(news)
    news_tfidf = vectorizer.transform([news_cleaned])
    prediction = model.predict(news_tfidf)[0]

    if prediction == 1:
        print("The news is REAL.")
    else:
        print("The news is FAKE.")

# Example of user input for prediction
user_news = input("Enter the news text: ")
predict_news(user_news)
```

Model Accuracy: 98.35%

Enter the news text: WASHINGTON (Reuters) - The head of a conservative Republican faction in the U.S. Congress, who voted this month for a huge expansion of the national debt to pay

The news is REAL.

## Comparative Analysis:

### 1. Algorithm Complexity:

- **First Code (Random Forest & XGBoost):** Both models are complex ensemble methods suitable for high-dimensional and non-linear datasets. XGBoost is more optimized and generally more accurate but requires careful parameter tuning.
- **Second Code (Logistic, Bayesian Logistic, Ridge Regression):** These are simpler linear models, with Bayesian Logistic Regression providing a probabilistic framework and Ridge Regression adding regularization to standard logistic regression.

### 2. Performance and Use Cases:

- **Random Forest & XGBoost:** Preferred for datasets where capturing complex patterns is crucial, like text data in fake news detection, where relationships between words can be non-linear.
- **Logistic Regression:** Works well for simpler problems with linear decision boundaries and when interpretability is needed.
- **Bayesian Logistic Regression:** Useful in situations where uncertainty quantification is essential.
- **Ridge Regression:** A good choice for high-dimensional datasets where overfitting is a risk.

### 3. Computational Efficiency:

- **First Code:** More computationally intensive due to ensemble models. XGBoost, while faster than traditional boosting, still requires significant resources.
- **Second Code:** More efficient and faster to train, especially useful for quick experimentation or when deploying on limited hardware.

#### 4. Interpretability:

- **Random Forest & XGBoost:** Less interpretable, though feature importance metrics can provide some insights.
- **Logistic Regression & Ridge Regression:** Highly interpretable, allowing an understanding of feature impacts.
- **Bayesian Logistic Regression:** Provides additional probabilistic interpretation, valuable for decision-making under uncertainty.

#### 5. Flexibility:

- **First Code:** Offers flexibility in capturing complex, non-linear relationships.
- **Second Code:** Simpler models that are easier to interpret but less flexible for complex data patterns.

#### Summary:

- **If your goal** is to achieve high accuracy and you have sufficient computational resources, the **ensemble models (Random Forest & XGBoost)** in the first code are more suitable.
- **For simpler, faster, and more interpretable models**, the **regression models (Logistic, Bayesian, Ridge)** in the second code are preferable.
- **Bayesian Logistic Regression** adds value when understanding uncertainty is essential, while **Ridge Regression** is beneficial for high-dimensional datasets prone to overfitting.