

Fake news prediction web application

**Created By**

|  |
| --- |
| Students Name |
| Monerah Almobarak |
| Nada Alotaibi |

**Table of Contents**

[Problem Statement: 3](#_Toc136124214)

[Project Aim: 3](#_Toc136124215)

[Project Objectives: 3](#_Toc136124216)

[Description of the dataset: 3](#_Toc136124217)

[methodology 5](#_Toc136124218)

[1.Data collection: 6](#_Toc136124219)

[2.Data pre-processing and exploratory data analysis 6](#_Toc136124220)

[3.Feature extraction and representation 6](#_Toc136124221)

[4.Model selection 7](#_Toc136124222)

[5.Model training and evaluation 7](#_Toc136124223)

[6.Model deployment using Flask: 9](#_Toc136124224)

[References 13](#_Toc136124225)

**Table of Figures**

[Figure 1. word clouds for both fake and not fake news articles to visualize the most common words in each class. 4](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397887)

[Figure 2. heatmap to visualize the presence of missing values in the dataset. 4](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397888)

[Figure 3. Distribution of Fake and Not Fake News Articles. 5](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397889)

[Figure 4. methodology for fake news prediction web application 6](#_Toc136397890)

[Figure 5.SVC Confusion Matrix 9](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397891)

[Figure 6. Gaussian Naive Bayes Confusion Matrix 10](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397892)

[Figure 7 Logistic Regression Confusion Matrix 11](#_Toc136397893)

[Figure 8. chart for Performance Metrics 12](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397894)

[Figure 9. home page 14](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397895)

[Figure 10 result page if fake news. 14](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397896)

[Figure 11. result page if not fake news. 14](file:///C:\Users\moner\Downloads\ML%20project%20report.docx#_Toc136397897)

# **Problem Statement:**

In today's world, the spread of fake news has become a major concern. With the rise of social media, it has become easier than ever to disseminate false information. Fake news can have serious consequences, leading to misinformation, confusion, and even harm. Therefore, it is important to develop effective systems to identify and prevent the propagation of fake news.

# **Project Aim:**

# This project aims to create a fake news prediction web application that accurately classifies news articles as fake or not fake. It will be trained on a labeled dataset using machine learning techniques and deployed as a web application using Flask in Visual Studio Code.

# **Project Objectives:**

* Collect and pre-process the labeled dataset of news articles from Kaggle.
* Explore and analyse the dataset to gain insights into the characteristics of fake and not fake news articles.
* Build and train a machine learning model that can accurately classify news articles as fake or not fake.
* Evaluate the performance of the model using appropriate metrics, such as accuracy, precision, recall, F1 score, and confusion matrix.
* Deploy the model using Flask in Visual Studio Code as a web application, enabling users to verify the veracity of news articles in real time.

**Description of the dataset:** The dataset used for the fake news prediction system is a collection of news articles compiled in a CSV file. It originally consisted of 10,000 records, but after removing 36 duplicate records, the dataset contains 9,964 unique records. The dataset has 5 columns: title, text, subject, date, and class. The class column serves as the target variable, where 1 represents fake news and 0 represents not fake news [1].

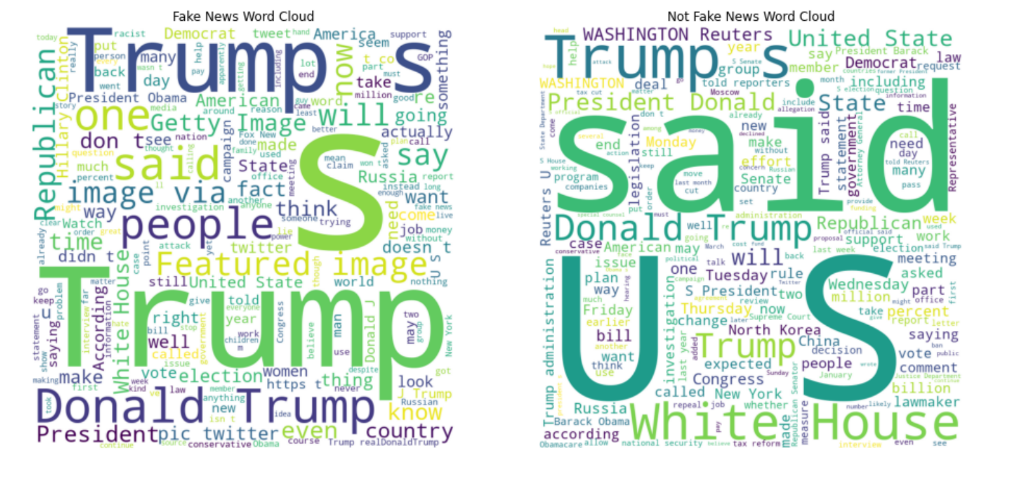


Figure 1. word clouds for both fake and not fake news articles to visualize the most common words in each class.

* **Resource:** The dataset is a CSV file containing news articles labeled as fake (1) or legitimate (0). It was uploaded to Kaggle by Clément Bisaillon.[1]
* **Size**: The dataset contains 10,000 unique records and 5 columns.[1]
* **Features**:
* **title:** The title of the news article.
* **text:** The content of the news article.
* **subject:** The subject of the news article.
* **date:** The date the news article was published.
* **Target**: The class column, where 1 represents fake news and 0 represents not fake news.[1]
* **Quality of Data**:
* **A purple rectangle with text

  Description automatically generated with low confidenceMissing Values:** The dataset is free of missing values, indicating high-quality data in terms of missing value completeness.

Figure 2. heatmap to visualize the presence of missing values in the dataset.

* **A picture containing text, screenshot, plot, diagram

  Description automatically generatedClass Distribution:** A histogram of the class distribution shows a balanced representation of fake and legitimate news articles, ensuring data quality regarding class balance.

Figure 3. Distribution of Fake and Not Fake News Articles.

* **Duplicate Removal:** Duplicate Removal: There are duplicate data that need to be addressed in order to ensure a unique and high-quality dataset.
* **Text Pre-processing:** The text data will require a series of pre-processing steps, including tokenization, lemmatization, stop word removal, sentiment analysis, part-of-speech tagging, named entity recognition, and topic modeling.

These pre-processing techniques will improve the data quality, making it more suitable for training machine learning models. A detailed explanation of these techniques and their implementation will be provided in the methodology section.

**Methodology:**

A picture containing text, screenshot, diagram, font

Description automatically generatedThe methodology for building a fake news prediction system using the dataset and deploying it with Flask can be broken down into the following steps:

Figure 4. methodology for fake news prediction web application

# **1.Data collection:**

In this step, a dataset containing news articles labeled as fake or not fake is obtained from Kaggle [1]. The dataset includes features such as the title, text, subject, date, and a corresponding class label indicating whether the news is fake (1) or not fake (0).

**2.Data pre-processing and exploratory data analysis**

Data preprocessing and exploratory analysis are crucial steps in any data analysis project. These steps help understand the structure and quality of data, identify any potential issues or outliers, and prepare the data for use in machine learning models or other analyses.

The dataset used in our analysis had no missing values, which is a positive aspect of the data quality. However, during the data cleaning process, we identified 36 duplicate records in the dataset. We carefully examined the nature of the duplicate records, determined that they were the result of a data entry error, and removed them without significantly affecting the overall quality of the data.

In terms of data preprocessing techniques, we decided to tokenization, lemmatization, remove stop word, and sentiment analysis. Given the size of our dataset and the scope of our analysis, we deemed these techniques unnecessary for our purposes.[2]

# **3.Feature extraction and representation**

# In this project, we explored feature extraction and representation techniques in natural language processing using Python and the scikit-learn library. The focus of the project was on the use of term frequency-inverse document frequency (TF-IDF) for feature extraction and representation.

# We started with a collection of raw text data, which was pre-processed using various techniques such as tokenization, stop word removal, and lemmatization, as done in phase 2. We then used the TF-IDF Vectorizer from the scikit-learn library to transform the pre-processed text data into a numerical format using the TF-IDF technique.

# TF-IDF assigns a weight to each word in the text data based on its frequency in the document and rarity in the corpus. The resulting feature matrix is a sparse matrix, where each row represents a document, and each column represents a unique word in the corpus.[3]

# **4.Model selection**

# In the model selection phase, we compared three classification models: Support Vector Classifier (SVC), Gaussian Naive Bayes, and Logistic Regression. Here are the definitions of each model:

# **Support Vector Classifier (SVC):** SVC is a supervised machine learning technique commonly used for classification problems. It aims to find the best hyperplane that separates the data points into two classes by mapping them to a high-dimensional space. The SVC model is effective in handling complex decision boundaries and can work well with both linearly separable and non-linearly separable data.[4]

# **Gaussian Naive Bayes:** Gaussian Naive Bayes is a classification method used in Machine Learning (ML) based on the probabilistic approach and Gaussian distribution. It assumes that features are independent and follow a Gaussian (normal) distribution. This model is computationally efficient and performs well in situations where the assumption of feature independence holds reasonably well.[5]

# **Logistic Regression:** Logistic regression is a data analysis method that uses mathematical techniques to determine the connections between two data factors. It predicts the value of one parameter based on the relationship established with other parameters. In the context of classification, logistic regression models the probability of an instance belonging to a particular class using a logistic function. It is widely used for binary classification problems.[6]

# **5. Model training and evaluation**

To prepare the data for model training, we first split the data into features and target. We used the Term Frequency-Inverse Document Frequency (TF-IDF) technique to vectorize the features and a standard scaler to scale them.

Before training our models, it is crucial to split the data twice. First, we split the data into a 60% training-validation set and a 40% testing set. Second, we further split the training-validation set into separate training and validation sets.

We started by training and evaluating the SVC model using cross-validation. To assess the model's performance, we computed the classification report, test accuracy, and created a confusion matrix. The definitions of these evaluation metrics are as follows:

* Confusion matrix: A table that describes how well a classification method performs.
* Accuracy: Reveals how frequently the machine learning model was overall correct.
* Precision: Measures how well the model predicts a given category.
* Recall: Reveals how frequently a particular category might be identified by the model.
* F1 score: A machine learning assessment statistic that assesses the precision of a model. It combines a model's recall and precision scores.
* Support: The number of samples that accurately represent each class of target values.

These steps were executed using Jupyter Notebook, the goal is to determine the best model for fake news detection based on the evaluation metrics.

# **6.Result:**

After training and evaluating the three models, we found the following results:

* **Support Vector Classifier (SVC):**
* Test accuracy: 0.924.
* Average cross-validation score: 0.917.

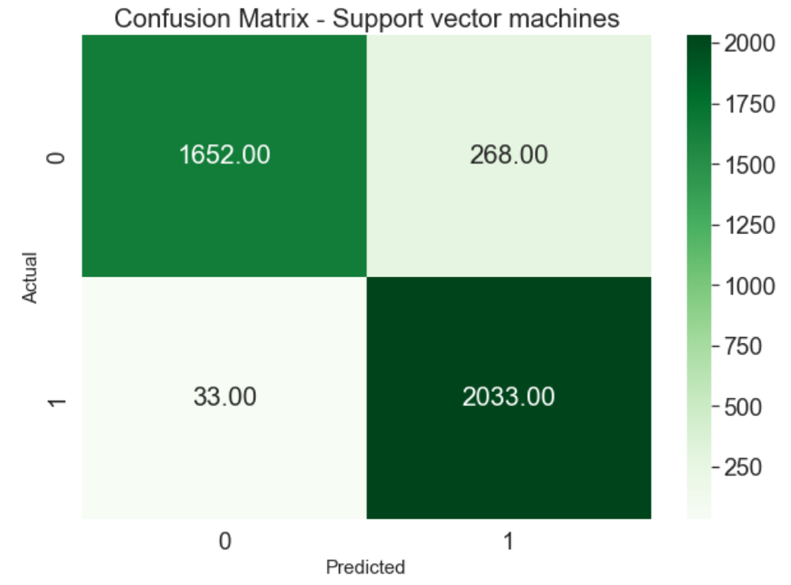
The SVC Confusion Matrix showed that:

Figure 5.SVC Confusion Matrix

* True Negative (TN) = 1652: The model correctly predicted real news as real news.
* True Positive (TP) = 2033: The model correctly predicted fake news as fake news
* False Positive (FP) = 268: The model incorrectly predicted real news as fake news (Type 1 error).
* False Negative (FN) = 33: The model incorrectly predicted fake news as real news (Type 2 error).
* **Gaussian Naive Bayes:**
* Test accuracy: 0.892.
* Average cross-validation score: 0.883.

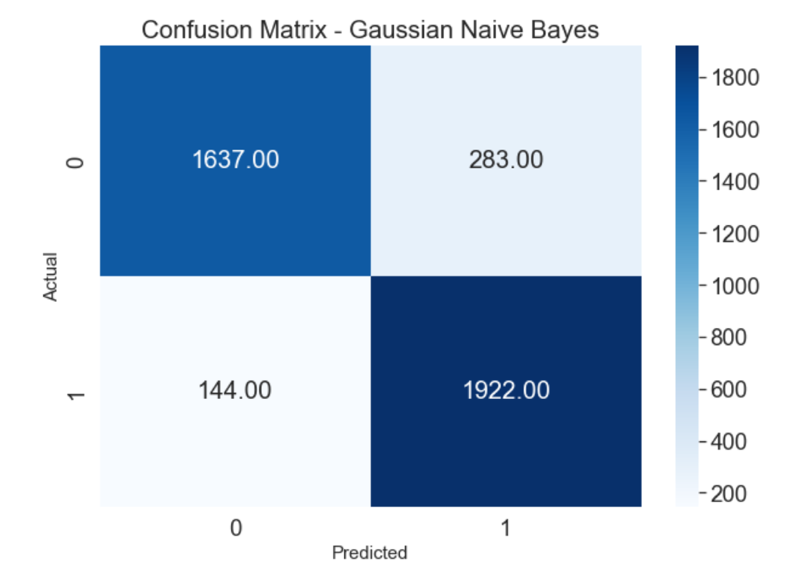
The Gaussian Naive Bayes Confusion Matrix showed that:

Figure 6. Gaussian Naive Bayes Confusion Matrix

* True Negative (TN) = 1637: The model correctly predicted real news as real news.
* True Positive (TP) = 1922: The model correctly predicted fake news as fake news.
* False Positive (FP) = 283: The model incorrectly predicted real news as fake news (Type 1 error).
* False Negative (FN) = 144: The model incorrectly predicted fake news as real news (Type 2 error).

# **Logistic Regression:**

# Test accuracy: 0.985.

# Average cross-validation score: 0.983.

The Logistic Regression Confusion Matrix showed that:

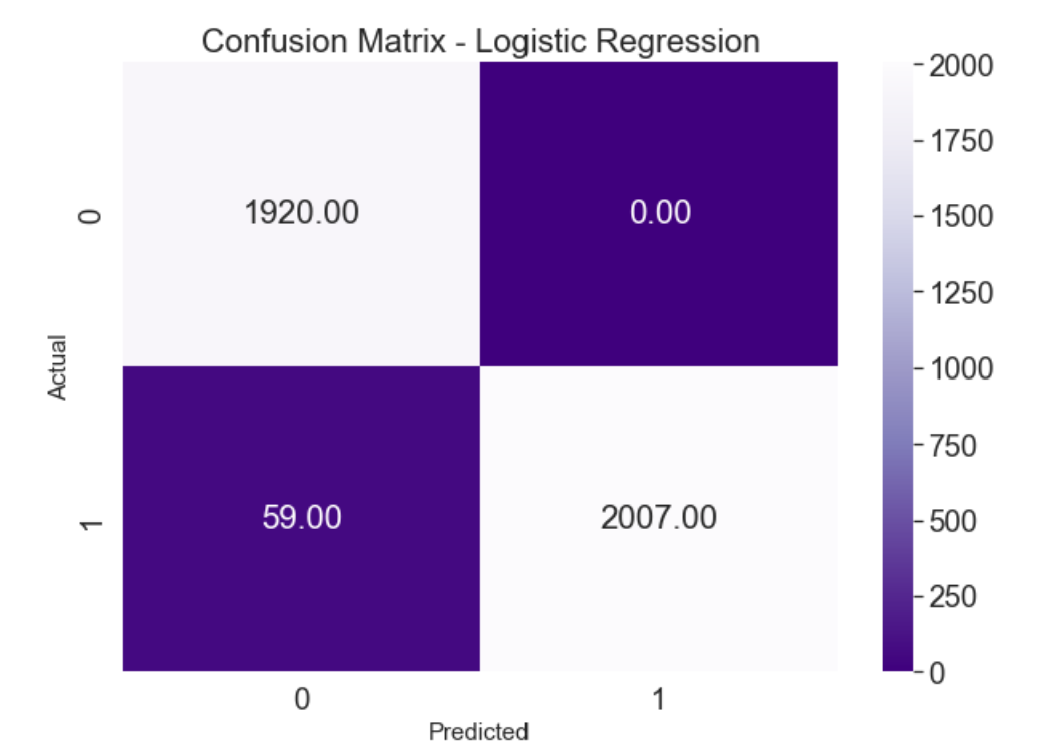


Figure 7 Logistic Regression Confusion Matrix

# True Negative (TN) = 1920: The model correctly predicted real news as real news.

# True Positive (TP) = 2007: The model correctly predicted fake news as fake news.

# False Positive (FP) = 0: The model did not predict any real news incorrectly as fake news (Type 1 error).

# False Negative (FN) = 59: The model incorrectly predicted fake news as real news (Type 2 error).

**The performance matrix:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | accuracy | precision | recall | F1-score |
| Logistic regression | 0.99 | 0.99 | 0.99 | 0.99 |
| Gaussian naïve bayes | 0.89 | 0.89 | 0.89 | 0.89 |
| SVM | 0.92 | 0.93 | 0.92 | 0.92 |
|  |  |  |  |  |

Table 1. Performance Metrics

Figure 8. chart for Performance Metrics

Based on these metrics, Logistic Regression consistently outperforms the other models, achieving the highest accuracy, precision, recall, and F1-score. This indicates that Logistic Regression is more effective in accurately classifying news articles as fake or not fake. Therefore, Logistic Regression will be the chosen model for deployment.

# **7.Model deployment using Flask:**

After training and evaluating the Logistic Regression model using Jupyter Notebook to determine the best model based on the evaluation metrics, we proceeded to deploy the model. To achieve this, we used Visual Studio Code (VSCode), a popular code editor that supports various programming languages and development environments.

We deployed the model using Flask, a lightweight web framework for Python. The app.py script contains the necessary data pre-processing and feature extraction steps from phases 2 and 3. It also initializes the Flask app and defines the required routes for handling user input and displaying the prediction result. The input\_news.html and results.html files provide a user-friendly interface for users to interact with the Fake News Detection model.

By deploying the model in this manner, we can integrate it into a web application, enabling users to access and utilize the fake news detection capabilities provided by the trained model. The Flask web application consists of three main components:

**- app.py:** The primary Python script that implements the machine learning model and initializes the Flask app. It includes data pre-processing, feature extraction, and route definitions for handling user input and displaying prediction results.

**- input\_news.html:** An HTML file that provides a user interface, allowing users to input news articles and submit them for authenticity evaluation.

- **results.html:** An HTML file that displays the prediction results, indicating whether the submitted news article is classified as fake or not fake.

The Flask app processes both GET and POST requests. When a user submits a news article, the app pre-processes the text, converts it into feature vectors using the same approach as the training stage, and feeds it to the trained Logistic Regression model for prediction. The prediction result is then displayed using the results.html template.[7]

# **A screenshot of a fake news prediction Description automatically generated The web application interfaces for predicting fake news**

Figure 9. home page

# **A screenshot of a fake news prediction result Description automatically generated**

**A screenshot of a fake news prediction result

Description automatically generated**

Figure 10 result page if fake news.

Figure 11. result page if not fake news.

# **References:**

[1] C. Bisaillon, "Fake and real news dataset," Kaggle, 2020. [Online].

Available: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

[2] Clémence, S. (2021). Data Cleaning in Python: The Ultimate Guide. MonkeyLearn. [Online].

Available: https://monkeylearn.com/blog/data-cleaning-python/. [Accessed: May 30, 2023].

[3] Kumar, A. (2021). Text Cleaning Techniques for Natural Language Processing. Data Knows All. [Online].

Available: https://www.dataknowsall.com/textcleaning.html. [Accessed: May 30, 2023].

[4]pub.towardsai,“gaussian-naive-bayes-explained-and-hands-on-with-scikit-learn,” [Online]. Available:https://pub.towardsai.net/gaussian-naive-bayes-explained-and-hands-on-with-scikit-learn-4183b8cb0e4c.

[5]sciencedirect,“logistic-regression,”[Online].

Available: https://www.sciencedirect.com/topics/computer-science/logistic-regression.

[6] medium, “confusion-matrix,” [Online].

Available: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5.

[7] G. A. Gajula, "How To Make a Web Application Using Flask in Python 3," DigitalOcean, Sep. 27, 2018. [Online].

Available:https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3

**Appendix:**

**App.py:**

*# Import the necessary libraries*

import **warnings**

import **pandas** as **pd**

import **nltk**

from **nltk**.**stem** import **WordNetLemmatizer**

from **nltk**.**corpus** import stopwords

from **nltk**.**tokenize** import **word\_tokenize**

from **nltk**.**sentiment** import **SentimentIntensityAnalyzer**

from **sklearn**.**decomposition** import **LatentDirichletAllocation**

from **sklearn**.**feature\_extraction**.**text** import **TfidfVectorizer**

from **sklearn**.**model\_selection** import **train\_test\_split**, **cross\_val\_score**

from **sklearn**.**preprocessing** import **StandardScaler**

from **sklearn**.**metrics** import **accuracy\_score**

from **sklearn**.**linear\_model** import **LogisticRegression**

from **flask** import **Flask**, **render\_template**, request

*# Download NLTK resources*

**nltk**.download('punkt')

**nltk**.download('wordnet')

**nltk**.download('stopwords')

*# Read the data*

data = **pd**.**read\_csv**(r"C:\Users\moner\Downloads\datamlproject.csv")

*# Ignore warnings*

**warnings**.**filterwarnings**("ignore")

*# Remove duplicates from the DataFrame*

data = data.**drop\_duplicates**()

*# Combine the text columns*

data['combined'] = data['title'] + ' ' + data['text'] + ' ' + data['subject'] + ' ' + data['date']

class **LemmaTokenizer**:

    def **\_\_init\_\_**(self):

        self.wnl = **WordNetLemmatizer**()

        self.stopwords = **set**(stopwords.**words**('english'))

        self.sid = **SentimentIntensityAnalyzer**()

    def **\_\_call\_\_**(self, doc):

        tokens = **word\_tokenize**(doc, preserve\_line=True)  *# Preserve the entire document as a single sentence*

        tokens = [self.wnl.**lemmatize**(t) for t in tokens]  *# Lemmatization*

        tokens = [t for t in tokens if t.**lower**() not in self.stopwords]  *# Stop word removal*

*# Sentiment analysis*

        sent\_scores = self.sid.**polarity\_scores**(doc)

        tokens.**extend**(["sent\_{}".**format**(k) for k, v in sent\_scores.**items**() if v != 0])

        return tokens

class **TextPreprocessor**:

    def **\_\_init\_\_**(self):

        self.wnl = **WordNetLemmatizer**()

        self.stopwords = **set**(stopwords.**words**('english'))

        self.sid = **SentimentIntensityAnalyzer**()

        self.lda = **LatentDirichletAllocation**(n\_components=5, random\_state=42)

    def **preprocess**(self, doc):

*# Tokenization and Lemmatization*

        tokens = **word\_tokenize**(doc)

        tokens = [self.wnl.**lemmatize**(t) for t in tokens]

*# Stop word removal*

        tokens = [t for t in tokens if t.**lower**() not in self.stopwords]

*# Sentiment analysis*

        sent\_scores = self.sid.**polarity\_scores**(doc)

        tokens.**extend**(["sent\_{}".**format**(k) for k, v in sent\_scores.**items**() if v != 0])

        return ' '.**join**(tokens)

*# Apply the preprocessing on the combined text*

preprocessor = **TextPreprocessor**()

data['preprocessed'] = data['combined'].**apply**(preprocessor.**preprocess**)

*# Split the data into features and target*

X = data['preprocessed']

y = data['class']

*# Initialize the vectorizer and scaler*

vectorizer = **TfidfVectorizer**(tokenizer=**LemmaTokenizer**())

standard\_scaler = **StandardScaler**(with\_mean=False)

*# Transform the features using the vectorizer and scaler*

X\_tfidf = vectorizer.**fit\_transform**(X)

X\_scaled = standard\_scaler.**fit\_transform**(X\_tfidf)

*# Split the data into train and test sets*

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_scaled, y, test\_size=0.40, random\_state=42)

lr\_model = **LogisticRegression**()

*# Train and evaluate Logistic Regression model with cross-validation*

lr\_cv\_scores = **cross\_val\_score**(lr\_model, X\_train, y\_train, cv=5)

lr\_cv\_scores\_mean = lr\_cv\_scores.**mean**()

*# Train the Logistic Regression model on the entire training set*

lr\_model.**fit**(X\_train, y\_train)

*# Make predictions on the test set*

lr\_y\_pred = lr\_model.**predict**(X\_test)

*# Calculate the test accuracy for Logistic Regression*

lr\_test\_accuracy = **accuracy\_score**(y\_test, lr\_y\_pred)

*# Initialize the Flask app*

app = **Flask**(\_\_name\_\_)

**@app.route**('/', methods=['GET', 'POST'])

def **home**():

    if request.method == 'POST':

        user\_input\_news = request.form['news\_input']

        preprocessed\_news = preprocessor.**preprocess**(user\_input\_news)

        vectorized\_news = vectorizer.**transform**([preprocessed\_news])

        scaled\_news = standard\_scaler.**transform**(vectorized\_news)

        prediction = lr\_model.**predict**(scaled\_news)

        prediction\_text = "Fake News" if prediction[0] == 1 else "Not Fake News"

        return **render\_template**('results.html',

                               prediction\_text=prediction\_text)

    return **render\_template**('input\_news.html')

if \_\_name\_\_ == '\_\_main\_\_':

    app.**run**()

**input\_news.html:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Enter News</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

            margin: 0;

            padding: 0;

        }

        header {

            background-color: #808080;

            padding: 20px;

            text-align: center;

            color: white;

        }

        main {

            margin: 20px;

            padding: 20px;

            background-color: white;

            border-radius: 10px;

        }

        label {

            display: block;

            margin-bottom: 10px;

        }

        textarea {

            width: 100%;

            resize: none;

            margin-bottom: 20px;

        }

        input[type="submit"] {

            background-color: #808080;

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        input[type="submit"]:hover {

            background-color: #606060;

        }

        footer {

            text-align: center;

            margin-top: 20px;

            font-size: 0.8em;

            color: #808080;

        }

    </style>

</head>

<body>

    <header>

        <h1>Fake News Prediction </h1>

    </header>

    <main>

        <form *action*="/" *method*="POST">

            <label *for*="news\_input"> Enter News to check if it's fake or not fake:</label>

            <textarea *id*="news\_input" *name*="news\_input" *rows*="10" *cols*="50"></textarea>

            <input *type*="submit" *value*="Check News">

        </form>

    </main>

    <footer>

        <p>

            Supervised by Zuhaira Muhammad Zain

        </p>

        <p>

            Created by Monerah Almobarak , Nada Alotaibi , Sarah Aljuhani , Sarah Altaweel

         </footer>

</body>

</html>

**results.html:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>News Prediction Result</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

            margin: 0;

            padding: 0;

        }

        header {

            background-color: #808080;

            padding: 20px;

            text-align: center;

            color: white;

        }

        main {

            margin: 20px;

            padding: 20px;

            background-color: white;

            border-radius: 10px;

        }

        h2 {

            color: #808080;

        }

        .fake {

            color: red;

        }

        .not-fake {

            color: green;

        }

        a {

            display: inline-block;

            background-color: #808080;

            color: white;

            padding: 10px 20px;

            border-radius: 5px;

            text-decoration: none;

        }

        a:hover {

            background-color: #606060;

        }

    </style>

</head>

<body>

    <header>

        <h1>Fake News Prediction Result</h1>

    </header>

    <main>

        <h2>The news is predicted to be: <span *class*="{{ 'not-fake' if prediction\_text == 'Not Fake News' else 'fake' }}">{{ prediction\_text }}</span></h2>

        <a *href*="/">Check another news</a>

    </main>

</body>

</html>

**Jupyter notebook:**

*# Import the necessary libraries*

**import** warnings

**import** pandas **as** pd

**import** nltk

**from** nltk.stem **import** WordNetLemmatizer

**from** nltk.corpus **import** stopwords

**from** nltk.tokenize **import** word\_tokenize

**from** nltk.sentiment **import** SentimentIntensityAnalyzer

**from** sklearn.decomposition **import** LatentDirichletAllocation

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

**from** sklearn.model\_selection **import** train\_test\_split, cross\_val\_score

**from** sklearn.svm **import** SVC

**import** seaborn **as** sns

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.naive\_bayes **import** GaussianNB

**import** matplotlib.pyplot **as** plt

**from** sklearn.metrics **import** confusion\_matrix, classification\_report

In [2]:

*# Ignore warnings*

warnings**.**filterwarnings("ignore")

In [3]:

*# Download NLTK resources*

nltk**.**download('punkt')

nltk**.**download('wordnet')

nltk**.**download('stopwords')

[nltk\_data] Downloading package punkt to

[nltk\_data] C:\Users\moner\AppData\Roaming\nltk\_data...

[nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package wordnet to

[nltk\_data] C:\Users\moner\AppData\Roaming\nltk\_data...

[nltk\_data] Package wordnet is already up-to-date!

[nltk\_data] Downloading package stopwords to

[nltk\_data] C:\Users\moner\AppData\Roaming\nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!

Out[3]:

True

In [4]:

*# Read the data*

data **=** pd**.**read\_csv("datamlproject.csv")

# explanatory of your data

In [5]:

*# looking the head DataSet*

data**.**head()

Out[5]:

|  | **title** | **text** | **subject** | **date** | **class** |
| --- | --- | --- | --- | --- | --- |
| **0** | Donald Trump Sends Out Embarrassing New Year’... | Donald Trump just couldn t wish all Americans ... | News | 31-Dec-17 | 1 |
| **1** | Drunk Bragging Trump Staffer Started Russian ... | House Intelligence Committee Chairman Devin Nu... | News | 31-Dec-17 | 1 |
| **2** | Sheriff David Clarke Becomes An Internet Joke... | On Friday, it was revealed that former Milwauk... | News | 30-Dec-17 | 1 |
| **3** | Trump Is So Obsessed He Even Has Obama’s Name... | On Christmas day, Donald Trump announced that ... | News | 29-Dec-17 | 1 |
| **4** | Pope Francis Just Called Out Donald Trump Dur... | Pope Francis used his annual Christmas Day mes... | News | 25-Dec-17 | 1 |

In [6]:

*# looking the shape DataSet*

data**.**shape

Out[6]:

(10000, 5)

In [7]:

*#Checking the dtypes of all the columns*

data**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 title 10000 non-null object

1 text 10000 non-null object

2 subject 10000 non-null object

3 date 10000 non-null object

4 class 10000 non-null int64

dtypes: int64(1), object(4)

memory usage: 390.8+ KB

### Class Distribution Bar Chart:

* Create a bar chart showing the distribution of fake and not fake news articles.

In [8]:

sns**.**countplot(x**=**'class', data**=**data)

plt**.**xlabel('News Classification')

plt**.**ylabel('Count')

plt**.**title('Distribution of Fake and Not Fake News Articles')

plt**.**show()

**A blue and orange rectangles

Description automatically generated with low confidence**

* the Class Distribution Bar Char show balanced distribution so the data is in good quality.

### Word Clouds:

* Generate word clouds for both fake and not fake news articles to visualize the most common words in each class.

In [9]:

**from** wordcloud **import** WordCloud

*# Create separate dataframes for fake and not fake news articles*

fake\_news **=** data[data['class'] **==** 1]

not\_fake\_news **=** data[data['class'] **==** 0]

*# Generate word clouds for both classes*

fake\_wordcloud **=** WordCloud(width**=**800, height**=**800, background\_color**=**'white')**.**generate(' '**.**join(fake\_news['text']))

not\_fake\_wordcloud **=** WordCloud(width**=**800, height**=**800, background\_color**=**'white')**.**generate(' '**.**join(not\_fake\_news['text']))

*# Plot word clouds*

fig, (ax1, ax2) **=** plt**.**subplots(1, 2, figsize**=**(16, 8))

ax1**.**imshow(fake\_wordcloud)

ax1**.**set\_title('Fake News Word Cloud')

ax1**.**axis('off')

ax2**.**imshow(not\_fake\_wordcloud)

ax2**.**set\_title('Not Fake News Word Cloud')

ax2**.**axis('off')

plt**.**show()

**A collage of words

Description automatically generated with low confidence**

## Data preprocessing

In [10]:

*# count the number of missing values in each column*

data**.**isnull()**.**sum()

Out[10]:

title 0

text 0

subject 0

date 0

class 0

dtype: int64

### Missing Values Heatmap:

* Use a heatmap to visualize the presence of missing values in the dataset.

In [11]:

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

plt**.**figure(figsize**=**(10, 6))

sns**.**heatmap(data**.**isnull(), cbar**=False**, yticklabels**=False**, cmap**=**'viridis')

plt**.**show()

**A purple rectangle with black border

Description automatically generated with low confidence**

* it clean heatmap with no bright spots indicates that there are no missing values, so the data is in good quality.

### there are no missing values in each rows in the DataFrame

In [12]:

*# Check for duplicates in rows*

duplicate\_rows **=** data**.**duplicated()

*# Count the total number of duplicates*

total\_duplicates **=** duplicate\_rows**.**sum()

*# Print the total number of duplicates*

print(f"Total duplicates: {total\_duplicates}")

Total duplicates: 36

In [13]:

*# Remove duplicates from the DataFrame*

data **=** data**.**drop\_duplicates()

*# Verify the removal of duplicates*

print("DataFrame shape after removing duplicates:", data**.**shape)

DataFrame shape after removing duplicates: (9964, 5)

In [14]:

data**.**shape

Out[14]:

(9964, 5)

### there are some duplicates values and we remove it to improve quality

In [15]:

*# Combine the text columns*

data['combined'] **=** data['title'] **+** ' ' **+** data['text'] **+** ' ' **+** data['subject'] **+** ' ' **+** data['date']

In [16]:

**class** LemmaTokenizer:

**def** \_\_init\_\_(self):

self**.**wnl **=** WordNetLemmatizer()

self**.**stopwords **=** set(stopwords**.**words('english'))

self**.**sid **=** SentimentIntensityAnalyzer()

**def** \_\_call\_\_(self, doc):

tokens **=** word\_tokenize(doc) *# Tokenization using NLTK's word\_tokenize function*

tokens **=** [self**.**wnl**.**lemmatize(t) **for** t **in** tokens] *# Lemmatization*

tokens **=** [t **for** t **in** tokens **if** t**.**lower() **not** **in** self**.**stopwords] *# Stop word removal*

*# Sentiment analysis*

sent\_scores **=** self**.**sid**.**polarity\_scores(doc)

tokens**.**extend(["sent\_{}"**.**format(k) **for** k, v **in** sent\_scores**.**items() **if** v **!=** 0])

**return** tokens

**class** TextPreprocessor:

**def** \_\_init\_\_(self):

self**.**wnl **=** WordNetLemmatizer()

self**.**stopwords **=** set(stopwords**.**words('english'))

self**.**sid **=** SentimentIntensityAnalyzer()

self**.**lda **=** LatentDirichletAllocation(n\_components**=**5, random\_state**=**42)

**def** preprocess(self, doc):

*# Tokenization and Lemmatization*

tokens **=** word\_tokenize(doc)

tokens **=** [self**.**wnl**.**lemmatize(t) **for** t **in** tokens]

*# Stop word removal*

tokens **=** [t **for** t **in** tokens **if** t**.**lower() **not** **in** self**.**stopwords]

*# Sentiment analysis*

sent\_scores **=** self**.**sid**.**polarity\_scores(doc)

tokens**.**extend(["sent\_{}"**.**format(k) **for** k, v **in** sent\_scores**.**items() **if** v **!=** 0])

**return** ' '**.**join(tokens)

*# Apply the preprocessing on the combined text*

preprocessor **=** TextPreprocessor()

data['preprocessed'] **=** data['combined']**.**apply(preprocessor**.**preprocess)

* Tokenization: Breaking text into individual words or phrases, called tokens. This is usually the first step in any NLP task.
* Lemmatization: Reducing words to their base form, such as converting "running" to "run".
* Stop word removal: Removing common words that don't carry much meaning, such as "the", "and", and "of".
* Sentiment analysis: Determining the sentiment or emotional tone of the text, such as positive, negative, or neutral.
* Part-of-speech (POS) tagging: Identifying the part of speech of each word in the text, such as noun, verb, adjective, etc.
* Named entity recognition (NER): Identifying and classifying named entities in the text, such as people, organizations, and locations.
* Topic modeling: Discovering the underlying topics in the text, which can be useful for organizing and summarizing large amounts of text

In [17]:

*# data after preprocess*

data

Out[17]:

|  | **title** | **text** | **subject** | **date** | **class** | **combined** | **preprocessed** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Donald Trump Sends Out Embarrassing New Year’... | Donald Trump just couldn t wish all Americans ... | News | 31-Dec-17 | 1 | Donald Trump Sends Out Embarrassing New Year’... | Donald Trump Sends Embarrassing New Year ’ Eve... |
| **1** | Drunk Bragging Trump Staffer Started Russian ... | House Intelligence Committee Chairman Devin Nu... | News | 31-Dec-17 | 1 | Drunk Bragging Trump Staffer Started Russian ... | Drunk Bragging Trump Staffer Started Russian C... |
| **2** | Sheriff David Clarke Becomes An Internet Joke... | On Friday, it was revealed that former Milwauk... | News | 30-Dec-17 | 1 | Sheriff David Clarke Becomes An Internet Joke... | Sheriff David Clarke Becomes Internet Joke Thr... |
| **3** | Trump Is So Obsessed He Even Has Obama’s Name... | On Christmas day, Donald Trump announced that ... | News | 29-Dec-17 | 1 | Trump Is So Obsessed He Even Has Obama’s Name... | Trump Obsessed Even Obama ’ Name Coded Website... |
| **4** | Pope Francis Just Called Out Donald Trump Dur... | Pope Francis used his annual Christmas Day mes... | News | 25-Dec-17 | 1 | Pope Francis Just Called Out Donald Trump Dur... | Pope Francis Called Donald Trump Christmas Spe... |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **9994** | U.S. Agriculture secretary nominee submits eth... | (Reuters) - U.S. President Donald Trump’s nomi... | politicsNews | 13-Mar-17 | 0 | U.S. Agriculture secretary nominee submits eth... | U.S. Agriculture secretary nominee submits eth... |
| **9995** | Trump aides attack agency that will analyze he... | WASHINGTON (Reuters) - Aides to U.S. President... | politicsNews | 12-Mar-17 | 0 | Trump aides attack agency that will analyze he... | Trump aide attack agency analyze health bill '... |
| **9996** | Highlights: The Trump presidency on March 12 a... | (Reuters) - Highlights of the day for U.S. Pre... | politicsNews | 12-Mar-17 | 0 | Highlights: The Trump presidency on March 12 a... | Highlights : Trump presidency March 12 8:42 p.... |
| **9997** | Obama lawyers move fast to join fight against ... | WASHINGTON (Reuters) - When Johnathan Smith re... | politicsNews | 13-Mar-17 | 0 | Obama lawyers move fast to join fight against ... | Obama lawyer move fast join fight Trump WASHIN... |
| **9998** | NY mayor criticizes Trump's closing public atr... | NEW YORK (Reuters) - A showdown between New Yo... | politicsNews | 1-Jun-16 | 0 | NY mayor criticizes Trump's closing public atr... | NY mayor criticizes Trump 's closing public at... |

9964 rows × 7 columns

In [18]:

*# Split the data into features and target*

X **=** data['preprocessed']

y **=** data['class']

In [19]:

*# Initialize the vectorizer and scaler*

vectorizer **=** TfidfVectorizer(tokenizer**=**LemmaTokenizer())

standard\_scaler **=** StandardScaler(with\_mean**=False**)

*# Transform the features using the vectorizer and scaler*

X\_tfidf **=** vectorizer**.**fit\_transform(X)

X\_scaled **=** standard\_scaler**.**fit\_transform(X\_tfidf)

In [20]:

*# Split the data into train and test sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X\_scaled, y, test\_size**=**0.40, random\_state**=**42)

In [21]:

*# Initialize models*

svm\_model **=** SVC()

lr\_model **=** LogisticRegression()

nb\_model **=** GaussianNB()

# First model is SVM model

In [22]:

*# Train and evaluate SVM model with cross-validation*

svm\_cv\_scores **=** cross\_val\_score(svm\_model, X\_train, y\_train, cv**=**5)

print(f"SVM CV scores: {svm\_cv\_scores}")

print(f"SVM CV scores mean: {svm\_cv\_scores**.**mean()}")

*# Train the SVM model on the entire training set*

svm\_model**.**fit(X\_train, y\_train)

*# Make predictions on the test set*

svm\_y\_pred **=** svm\_model**.**predict(X\_test)

svm\_y\_pred

SVM CV scores: [0.92140468 0.92725753 0.90551839 0.90460251 0.92887029]

SVM CV scores mean: 0.9175306810707939

Out[22]:

array([0, 0, 0, ..., 1, 0, 0], dtype=int64)

In [23]:

*#Printing the training and testing accuracies*

print('Training Accuracy : {:.3f}'**.**format(svm\_model**.**score(X\_train, y\_train)))

print('Test Accuracy : {:.3f}'**.**format(svm\_model**.**score(X\_test, y\_test)))

Training Accuracy : 0.996

Test Accuracy : 0.924

In [24]:

*# Compute the classification report for SVM*

svm\_report **=** classification\_report(y\_test, svm\_y\_pred)

print("SVM Classification Report:")

print(svm\_report)

*# Compute the confusion matrix for SVM*

svm\_cm **=** confusion\_matrix(y\_test, svm\_y\_pred)

sns**.**set(font\_scale**=**1.7)

plt**.**figure(figsize**=**(10,7))

sns**.**heatmap(svm\_cm, annot**=True**, cmap**=**"Greens", fmt**=**'.2f')

plt**.**title('Confusion Matrix - Support vector machines')

plt**.**xlabel("Predicted", fontsize**=**15)

plt**.**ylabel("Actual", fontsize**=**15)

plt**.**show()

SVM Classification Report:

precision recall f1-score support

0 0.98 0.86 0.92 1920

1 0.88 0.98 0.93 2066

accuracy 0.92 3986

macro avg 0.93 0.92 0.92 3986

weighted avg 0.93 0.92 0.92 3986

**A screenshot of a graph

Description automatically generated with low confidence**

It turned out that in SVC Confusion Matrix:

* TP = 2033, so that the model predicted the right news correctly predicted as the actual right news
* TN = 1652, so that the model predicted fake news correctly predicted as an actual fake news
* FP = 268, so that the model predicted the right news incorrectly predicted the actual right news. This is considered a type 1 error
* FN = 33, so that the model predicted the fake news incorrectly predicted the actual fake news. This is considered a type 2 error

# Second model is Gaussian Naive Bayes

In [25]:

*# Train and evaluate Gaussian Naive Bayes model with cross-validation*

nb\_cv\_scores **=** cross\_val\_score(nb\_model, X\_train**.**toarray(), y\_train, cv**=**5)

print(f"Gaussian Naive Bayes CV scores: {nb\_cv\_scores}")

print(f"Gaussian Naive Bayes CV scores mean: {nb\_cv\_scores**.**mean()}")

*# Train the Gaussian Naive Bayes model on the entire training set*

nb\_model**.**fit(X\_train**.**toarray(), y\_train)

*# Make predictions on the test set*

nb\_y\_pred **=** nb\_model**.**predict(X\_test**.**toarray())

nb\_y\_pred

Gaussian Naive Bayes CV scores: [0.8729097 0.88963211 0.88043478 0.88619247 0.88953975]

Gaussian Naive Bayes CV scores mean: 0.8837417612403968

Out[25]:

array([0, 1, 0, ..., 1, 0, 0], dtype=int64)

In [26]:

*#Printing the training and testing accuracies*

print('Training Accuracy : {:.3f}'**.**format(nb\_model**.**score(X\_train**.**toarray(), y\_train)))

print('Test Accuracy : {:.3f}'**.**format(nb\_model**.**score(X\_test**.**toarray(), y\_test)))

Training Accuracy : 1.000

Test Accuracy : 0.893

In [27]:

*# Compute the classification report for Gaussian Naive Bayes*

nb\_report **=** classification\_report(y\_test, nb\_y\_pred)

print("Gaussian Naive Bayes Classification Report:")

print(nb\_report)

**import** seaborn **as** sns

**from** sklearn.metrics **import** confusion\_matrix

**import** matplotlib.pyplot **as** plt

nb\_cm **=** confusion\_matrix(y\_test, nb\_y\_pred)

sns**.**set(font\_scale**=**1.7)

plt**.**figure(figsize**=**(10,7))

sns**.**heatmap(nb\_cm, annot**=True**, cmap**=**"Blues", fmt**=**'.2f')

plt**.**title('Confusion Matrix - Gaussian Naive Bayes')

plt**.**xlabel("Predicted", fontsize**=**15)

plt**.**ylabel("Actual", fontsize**=**15)

plt**.**show()

Gaussian Naive Bayes Classification Report:

precision recall f1-score support

0 0.92 0.85 0.88 1920

1 0.87 0.93 0.90 2066

accuracy 0.89 3986

macro avg 0.90 0.89 0.89 3986

weighted avg 0.89 0.89 0.89 3986

**A screenshot of a graph

Description automatically generated with low confidence**

It turned out that in Gaussian Naive Bayes Confusion Matrix:

* TP = 1922, so that the model predicted the right news correctly predicted as the actual right news
* TN = 1637, so that the model predicted fake news correctly predicted as an actual fake news
* FP = 283, so that the model predicted the right news incorrectly predicted the actual right news. This is considered a type 1 error
* FN = 144, so the model predicted the fake news incorrectly predicted the actual fake news. This is considered a type 2 error

# Third model is Logistic Regression model

In [28]:

*# Train and evaluate Logistic Regression model with cross-validation*

lr\_cv\_scores **=** cross\_val\_score(lr\_model, X\_train, y\_train, cv**=**5)

print(f"Logistic Regression CV scores: {lr\_cv\_scores}")

print(f"Logistic Regression CV scores mean: {lr\_cv\_scores**.**mean()}")

*# Train the Logistic Regression model on the entire training set*

lr\_model**.**fit(X\_train, y\_train)

*# Make predictions on the test set*

lr\_y\_pred **=** lr\_model**.**predict(X\_test)

lr\_y\_pred

Logistic Regression CV scores: [0.97826087 0.98913043 0.99080268 0.97656904 0.98242678]

Logistic Regression CV scores mean: 0.9834379591665383

Out[28]:

array([0, 0, 0, ..., 1, 0, 0], dtype=int64)

In [29]:

*#Printing the training and testing accuracies*

print('Training Accuracy : {:.3f}'**.**format(lr\_model**.**score(X\_train, y\_train)))

print('Test Accuracy : {:.3f}'**.**format(lr\_model**.**score(X\_test, y\_test)))

Training Accuracy : 1.000

Test Accuracy : 0.985

In [30]:

*# Compute the classification report for Logistic Regression*

lr\_report **=** classification\_report(y\_test, lr\_y\_pred)

print("Logistic Regression Classification Report:")

print(lr\_report)

*# Compute the confusion matrix for Logistic Regression*

lr\_cm **=** confusion\_matrix(y\_test, lr\_y\_pred)

sns**.**set(font\_scale**=**1.7)

plt**.**figure(figsize**=**(10,7))

sns**.**heatmap(lr\_cm, annot**=True**, cmap**=**"Purples\_r", fmt**=**'.2f')

plt**.**title('Confusion Matrix - Logistic Regression')

plt**.**xlabel("Predicted", fontsize**=**15)

plt**.**ylabel("Actual", fontsize**=**15)

plt**.**show()

Logistic Regression Classification Report:

precision recall f1-score support

0 0.97 1.00 0.98 1920

1 1.00 0.97 0.99 2066

accuracy 0.99 3986

macro avg 0.99 0.99 0.99 3986

weighted avg 0.99 0.99 0.99 3986

**A screenshot of a graph

Description automatically generated with medium confidence**

It turned out that in Logistic Regression Confusion Matrix:

* TP = 2007, so that the model predicted the right news correctly predicted as the actual right news
* TN = 1920, so that the model predicted fake news correctly predicted as an actual fake news
* FP = 0, so the model didn't predict any right news incorrectly predicted the actual right news. This is considered a type 1 error
* FN = 59, so that the model predicted the fake news incorrectly predicted the actual fake news. This is considered a type 2 error