Deliverable-3

Choose Machine Learning Frameworks:

We have utilized the scikit-learn framework for developing and implementing my machine learning model. Scikit-learn is a versatile and user-friendly library in Python that offers a range of tools for machine learning, including various algorithms for classification, regression, clustering, and more. It is known for its ease of use, well-documented APIs, and compatibility with other popular libraries in the Python ecosystem.

To leverage the scalability and cloud computing capabilities, I utilized AWS Sagemaker to run my model. The dataset was fetched directly from an S3 bucket, allowing for efficient data storage and retrieval. AWS Sagemaker's infrastructure facilitated the training and deployment of the model, offering a scalable and convenient environment for machine learning workflows. This integration with cloud services enhances the flexibility and scalability of the overall machine learning pipeline

[2]:
```
import boto3
import pandas as pd
from io import BytesIO

s3 = boto3.client('s3')
bucket_name = 'mybucketgroup7'

final_merged_key = 'final_merged.csv'

# Function to load a CSV file from S3
def load_csv_from_s3(bucket, key):
    response = s3.get_object(Bucket=bucket, Key=key)
    content = response['Body']
    return pd.read_csv(BytesIO(content.read()))

# Load your dataframes
final_merged = load_csv_from_s3(bucket_name, final_merged_key)
```

[3]:
```
final_merged.head()
```

[3]:

	id	prompt_id	text	generated
0	0059830c	0.0	Cars. Cars have been around since they became ...	0
1	005db917	0.0	Transportation is a large necessity in most co...	0
2	008f63e3	0.0	"America's love affair with it's vehicles seem...	0
3	00940276	0.0	How often do you ride in a car? Do you drive a...	0
4	00c39458	0.0	Cars are a wonderful thing. They are perhaps o...	0

Develop and Train Models:

In the initial phase of data preprocessing, We extracted the 'text' column from the DataFrame, containing essays, and tokenized each essay using the NLTK library's word_tokenize function. To ensure consistency and remove redundancy, We converted all words to lowercase and excluded non-alphabetic characters. This step aimed at creating a cleaner and more uniform representation of the textual data. Subsequently, We identified and removed common stopwords, such as 'the' and 'and,' as they contribute little value to the overall analysis. This enhanced the focus on more meaningful words within the essays. The most common words after this processing step were then determined using the Counter class, revealing key terms that hold significance in the dataset.

Data Preprocessing

[9]:
```
essays = df['text'].tolist()
tokenized_data = [word_tokenize(essay) for essay in tqdm(essays)]

print(tokenized_data[0])
```

```
100%|██████████| 27340/27340 [01:30<00:00, 302.85it/s]
['Cars', '.', 'Cars', 'have', 'been', 'around', 'since', 'they', 'became', 'famous', 'in', 'the', '190
0s', ',', 'when', 'Henry', 'Ford', 'created', 'and', 'built', 'the', 'first', 'ModelT', '.', 'Cars',
'have', 'played', 'a', 'major', 'role', 'in', 'our', 'every', 'day', 'lives', 'since', 'then', '.', 'B
ut', 'now', ',', 'people', 'are', 'starting', 'to', 'question', 'if', 'limiting', 'car', 'usage', 'wou
ld', 'be', 'a', 'good', 'thing', '.', 'To', 'me', ',', 'limiting', 'the', 'use', 'of', 'cars', 'migh
t', 'be', 'a', 'good', 'thing', 'to', 'do', '.', 'In', 'like', 'matter', 'of', 'this', ',', 'article',
',', '``', 'In', 'German', 'Suburb', ',', 'Life', 'Goes', 'On', 'Without', 'Cars', ',', "''", 'by', 'E
lizabeth', 'Rosenthal', 'states', ',', 'how', 'automobiles', 'are', 'the', 'linchpin', 'of', 'suburb
s', ',', 'where', 'middle', 'class', 'families', 'from', 'either', 'Shanghai', 'or', 'Chicago', 'ten
d', 'to', 'make', 'their', 'homes', '.', 'Experts', 'say', 'how', 'this', 'is', 'a', 'huge', 'impedime
nt', 'to', 'current', 'efforts', 'to', 'reduce', 'greenhouse', 'gas', 'emissions', 'from', 'tailpipe',
'.', 'Passenger', 'cars', 'are', 'responsible', 'for', '12', 'percent', 'of', 'greenhouse', 'gas', 'em
issions', 'in', 'Europe', '...', 'and', 'up', 'to', '50', 'percent', 'in', 'some', 'carintensive', 'ar
eas', 'in', 'the', 'United', 'States', '.', 'Cars', 'are', 'the', 'main', 'reason', 'for', 'the', 'gre
enhouse', 'gas', 'emissions', 'because', 'of', 'a', 'lot', 'of', 'people', 'driving', 'them', 'aroun
d', 'all', 'the', 'time', 'getting', 'where', 'they', 'need', 'to', 'go', '.', 'Article', ',', '``',
'Paris', 'bans', 'driving', 'due', 'to', 'smog', ',', "''", 'by', 'Robert', 'Duffer', 'says', ',', 'ho
w', 'Paris', ',', 'after', 'days', 'of', 'nearrecord', 'pollution', ',', 'enforced', 'a', 'partial',
'driving', 'ban', 'to', 'clear', 'the', 'air', 'of', 'the', 'global', 'city', '.', 'It', 'also', 'say
s', ',', 'how', 'on', 'Monday', ',', 'motorist', 'with', 'evennumbered', 'license', 'plates', 'were',
'ordered', 'to', 'leave', 'their', 'cars', 'at', 'home', 'or', 'be', 'fined', 'a', '22euro', 'fine',
'31', '.', 'The', 'same', 'order', 'would', 'be', 'applied', 'to', 'oddnumbered', 'plates', 'the', 'fo
```

Make lowercase and remove punctuation:

We remove irrelevant characters like punctuations. To do so, we check if the word is alphanumeric or not thus using only words and omitting punctuations.

To remove redundancy we make the words lowercase so words like "DeviCEs" and "devices" are considered to be same and redundancy is removed.

[10]:
```
data_processed= [[word.lower() for word in essay if word.isalpha()] for essay in tokenized_data]
print(data_processed[0])
```

```
['cars', 'cars', 'have', 'been', 'around', 'since', 'they', 'became', 'famous', 'in', 'the', 'when',
'henry', 'ford', 'created', 'and', 'built', 'the', 'first', 'modelt', 'cars', 'have', 'played', 'a',
'major', 'role', 'in', 'our', 'every', 'day', 'lives', 'since', 'then', 'but', 'now', 'people', 'are',
'starting', 'to', 'question', 'if', 'limiting', 'car', 'usage', 'would', 'be', 'a', 'good', 'thing',
'to', 'me', 'limiting', 'the', 'use', 'of', 'cars', 'might', 'be', 'a', 'good', 'thing', 'to', 'do',
'in', 'like', 'matter', 'of', 'this', 'article', 'in', 'german', 'suburb', 'life', 'goes', 'on', 'with
out', 'cars', 'by', 'elizabeth', 'rosenthal', 'states', 'how', 'automobiles', 'are', 'the', 'linchpi
n', 'of', 'suburbs', 'where', 'middle', 'class', 'families', 'from', 'either', 'shanghai', 'or', 'chic
ago', 'tend', 'to', 'make', 'their', 'homes', 'experts', 'say', 'how', 'this', 'is', 'a', 'huge', 'imp
ediment', 'to', 'current', 'efforts', 'to', 'reduce', 'greenhouse', 'gas', 'emissions', 'from', 'tailp
ipe', 'passenger', 'cars', 'are', 'responsible', 'for', 'percent', 'of', 'greenhouse', 'gas', 'emissio
ns', 'in', 'europe', 'and', 'up', 'to', 'percent', 'in', 'some', 'carintensive', 'areas', 'in', 'the',
'united', 'states', 'cars', 'are', 'the', 'main', 'reason', 'for', 'the', 'greenhouse', 'gas', 'emissi
ons', 'because', 'of', 'a', 'lot', 'of', 'people', 'driving', 'them', 'around', 'all', 'the', 'time',
'getting', 'where', 'they', 'need', 'to', 'go', 'article', 'paris', 'bans', 'driving', 'due', 'to', 's
```

[11]:
```
flattened = [val for essay in data_processed for val in essay]
count = Counter(flattened)
count.most_common(10)
```

[11]:
```
[('the', 478221),
 ('to', 341458),
 ('and', 262567),
 ('a', 237747),
 ('of', 229121),
 ('in', 182381),
 ('is', 164822),
 ('that', 162721),
 ('it', 137157),
 ('for', 115089)]
```

As we can see above, the most common words are those which provide very little value to the data. So lets remove these stopwords and find the most common words relevant to the data ¶

[12]:
```
stop_words = stopwords.words('english')
data_processed = [[word for word in essay if word not in stop_words] for essay in data_processed]
data_processed_main = data_processed
print(data_processed_main[0])
```

```
['cars', 'cars', 'around', 'since', 'became', 'famous', 'henry', 'ford', 'created', 'built', 'first',
'modelt', 'cars', 'played', 'major', 'role', 'every', 'day', 'lives', 'since', 'people', 'starting',
'question', 'limiting', 'car', 'usage', 'would', 'good', 'thing', 'limiting', 'use', 'cars', 'might',
'good', 'thing', 'like', 'matter', 'article', 'german', 'suburb', 'life', 'goes', 'without', 'cars',
'elizabeth', 'rosenthal', 'states', 'automobiles', 'linchpin', 'suburbs', 'middle', 'class', 'familie
s', 'either', 'shanghai', 'chicago', 'tend', 'make', 'homes', 'experts', 'say', 'huge', 'impediment',
'current', 'efforts', 'reduce', 'greenhouse', 'gas', 'emissions', 'tailpipe', 'passenger', 'cars', 're
```

Following the removal of stopwords, lemmatization was employed to further refine the text data. The NLTK library's WordNetLemmatizer was applied to reduce words to their base or root form, considering their part-of-speech (POS) tags. This process aimed to standardize the language and improve the coherence of the dataset. By lemmatizing the words, variations of a term, such as verb conjugations or plural forms, were unified. Finally, visualizations in the form of word clouds were generated for both human-generated and AI-generated text, providing an intuitive representation of the most prominent words in each category. These visualizations serve as a preliminary exploration of the distinctive linguistic patterns within the two sets of data.

Lemmatization

[14]:
```
from tqdm import tqdm
lemmatizer = WordNetLemmatizer()

def lemmatize_words(words, pos_tags):
    return [lemmatizer.lemmatize(word, pos=wordnet_pos(tag)) for word, tag in zip(words, pos_tags)]

def wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


---
#sentPos = [nltk.pos_tag(sent) for sent in tqdm(data_processed_main)]
#dataLemma = [[lemmatizer.lemmatize(word, pos=wordnet_pos(tag)) for word, tag in pos_tags] for pos_tag
dataLemma = [[lemmatizer.lemmatize(word) for word in sent] for sent in tqdm(data_processed_main)]

print(dataLemma[0])
```

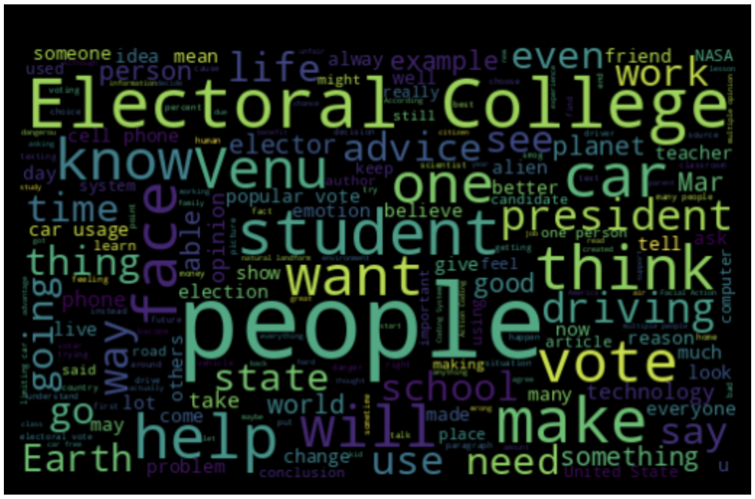
```
100%|██████████| 27340/27340 [00:26<00:00, 1013.66it/s]
['car', 'car', 'around', 'since', 'became', 'famous', 'henry', 'ford', 'created', 'built', 'first', 'm
odelt', 'car', 'played', 'major', 'role', 'every', 'day', 'life', 'since', 'people', 'starting', 'ques
tion', 'limiting', 'car', 'usage', 'would', 'good', 'thing', 'limiting', 'use', 'car', 'might', 'goo
d', 'thing', 'like', 'matter', 'article', 'german', 'suburb', 'life', 'go', 'without', 'car', 'elizabe
th', 'rosenthal', 'state', 'automobile', 'linchpin', 'suburb', 'middle', 'class', 'family', 'either',
'shanghai', 'chicago', 'tend', 'make', 'home', 'expert', 'say', 'huge', 'impediment', 'current', 'effo
rt', 'reduce', 'greenhouse', 'gas', 'emission', 'tailpipe', 'passenger', 'car', 'responsible', 'percen
t', 'greenhouse', 'gas', 'emission', 'europe', 'percent', 'carintensive', 'area', 'united', 'state',
'car', 'main', 'reason', 'greenhouse', 'gas', 'emission', 'lot', 'people', 'driving', 'around', 'tim
e', 'getting', 'need', 'go', 'article', 'paris', 'ban', 'driving', 'due', 'smog', 'robert', 'duffer',
'say', 'paris', 'day', 'nearrecord', 'pollution', 'enforced', 'partial', 'driving', 'ban', 'clear', 'a
```

Finally, visualizations in the form of word clouds were generated for both human-generated and AI-generated text, providing an intuitive representation of the most prominent words in each category. These visualizations serve as a preliminary exploration of the distinctive linguistic patterns within the two sets of data.

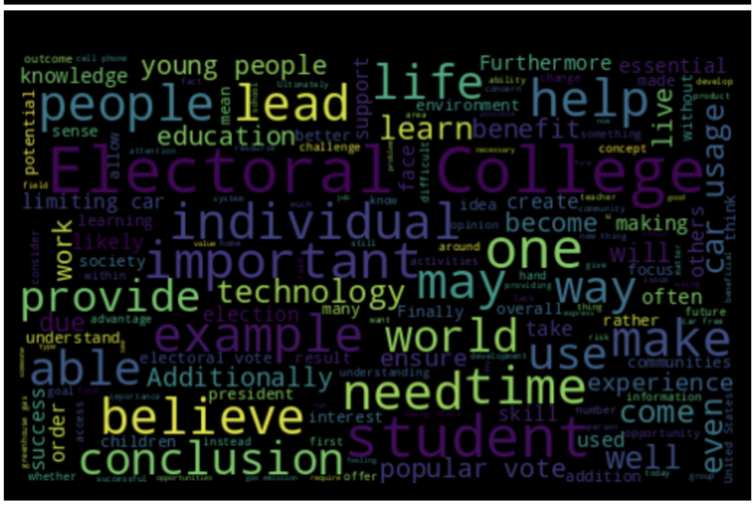
Word Cloud

[17]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
# Filter data for human-generated and AI-generated text
human_data = df[df['generated'] == 0]['text'].values.tolist()
ai_data = df[df['generated'] == 1]['text'].values.tolist()

human_data = ' '.join(human_data)
ai_data = ' '.join(ai_data)
# Function to generate word cloud
def generate_word_cloud(text, title):
    wordcloud = WordCloud(
        width=500,
        height=300,
        background_color='black',
        stopwords=STOPWORDS
    ).generate(text)

    fig, ax = plt.subplots(facecolor='k', edgecolor='k')
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title)
    plt.show()

# Generate word clouds for human-generated and AI-generated text
generate_word_cloud(human_data, 'Human Generated Word Cloud')
generate_word_cloud(ai_data, 'AI Generated Word Cloud')
```



people
Electoral College
student
think
know
venu
one
car
president
want
vote
make
help
will
face
driving
state
school
Earth
use
need
say
go
way
going
thing
time
life
even
work
advice
see
planet
elector
example
friend
NASA
someone
idea
mean
person
good
world
take
lot
something
technology
everyone
look
much
reason
article
now
tell
popular vote
emotion
believe
better
alien
Mar
teacher
candidate
car usage
election
show
give
feel
phone
live
many
change
place
made
come
problem
conclusion
United State
cell phone
day
system
author
keep
really
might
still
used
alway
opinion
able
others
said
road
may
computer
teacher



Electoral College
student
individual
important
example
people
lead
life
help
one
may
way
make
provide
technology
world
needtime
believe
conclusion
able
use
popular vote
Additionally
young people
knowledge
Furthermore
essential
education
learn
benefit
environment
support
live
car usage
create
become
making
limiting car
learning
likely
work
society
will
focus
often
future
Finally
overall
take
rather
understand
due
election
electoral vote
result
ensure
experience
come
president
interest
skill
children
instead
first
used
well
even
addition
success
order
potential
sense
outcome
cell phone
idea
others
think
many
advantage
information
opportunity

Model Training

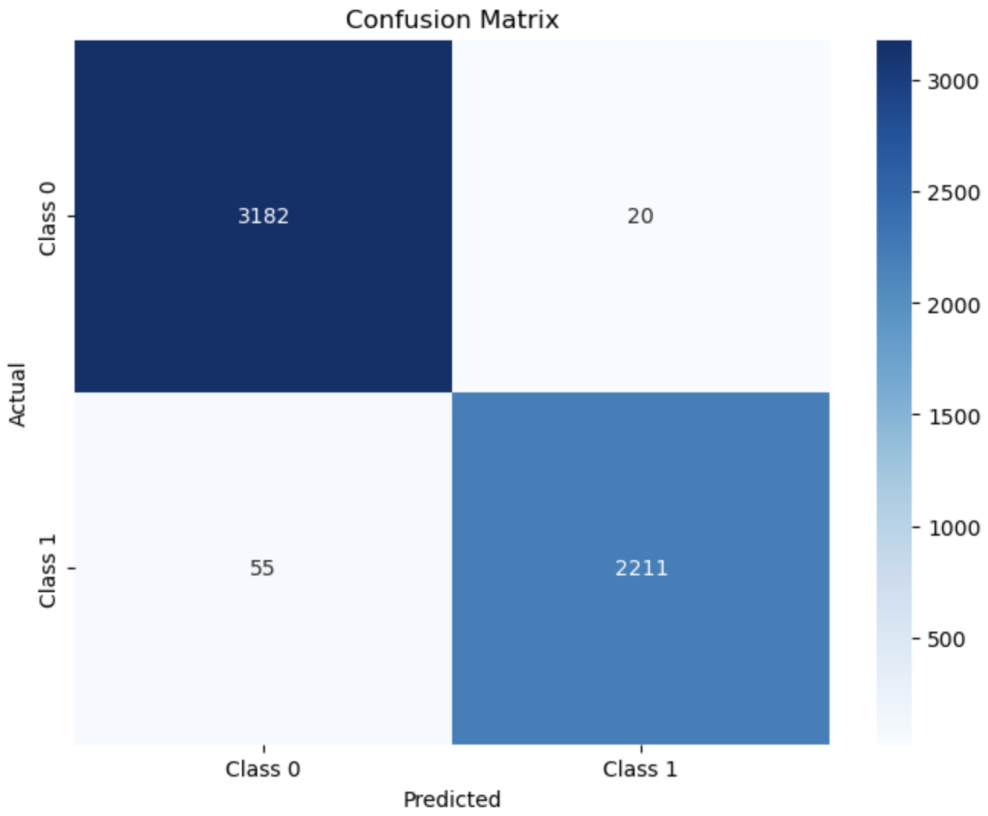
We started the model training with a Logistic Regression classifier. Using scikit-learn's LogisticRegression class, We fit the model on the training data (X_train, y_train) and saved the trained model to a file ('logreg_Train_Essay_Data.pkl') for future use. The evaluation of the model on the test set provided insightful performance metrics. The accuracy, precision, recall, and F1 score were calculated and printed, offering a comprehensive view of the model's predictive capabilities. Additionally, a confusion matrix was generated and visualized as a heatmap, providing a clear representation of the model's classification results.

Base Line Model - Logistic Regression

[22]:
```
from sklearn.metrics import confusion_matrix

def evaluate(model, X_test, y_test):
    y_pred = model.predict(X_test)
    LRaccuracy = accuracy_score(y_test, y_pred)
    LRprecision = precision_score(y_test, y_pred)
    LRrecall = recall_score(y_test, y_pred)
    LRf1 = f1_score(y_test, y_pred)
    print('Precision is: ',LRprecision)
    print('Accuracy is: ',LRaccuracy)
    print('Recall is: ',LRrecall)
    print('F1 is: ',LRf1)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Plot the confusion matrix as a heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yt
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

[23]:
```
import pickle
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

with open('logreg_Train_Esssay_Data.pkl', 'wb') as model_file:
    pickle.dump(logreg, model_file)

evaluate(logreg, X_test, y_test)
```

```
Precision is:  0.9910354101299865
Accuracy is:  0.9862838332114119
Recall is:  0.9757281553398058
F1 is:  0.9833222148098733
```

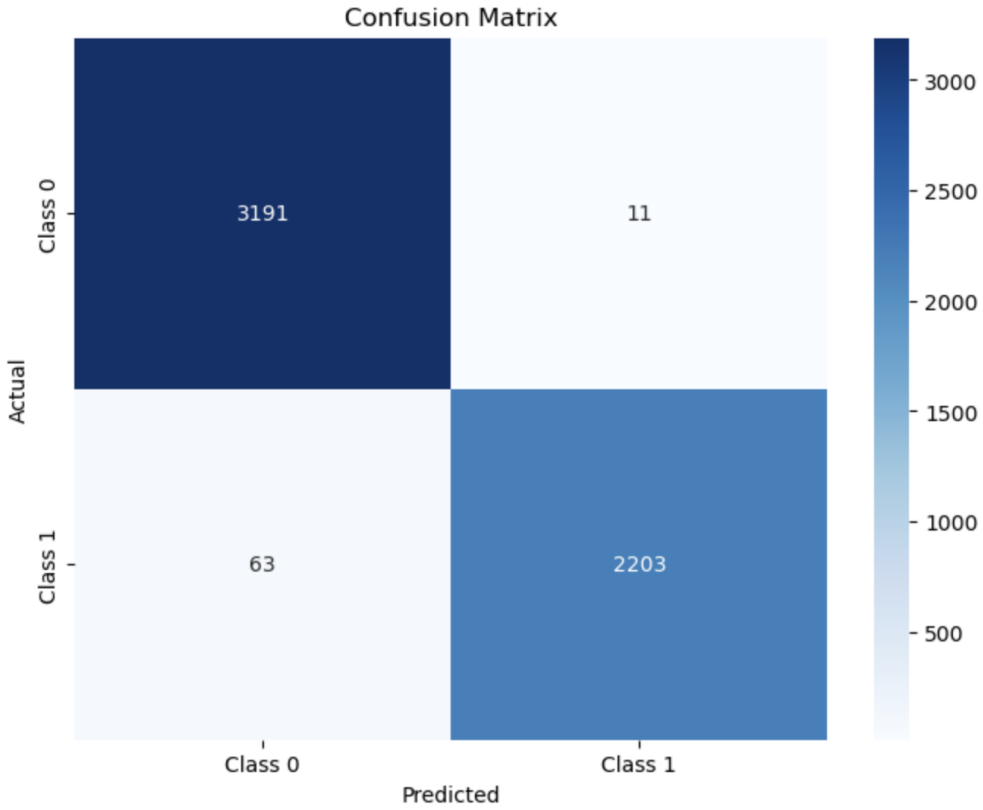

Confusion Matrix
3182
20
55
2211
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
3000
2500
2000
1500
1000
500

Moving on, we employed a Random Forest Classifier for the next model. Using the RandomForestClassifier class from scikit-learn, we trained the model on the same training data and saved it to a file ('ranFor_Train_Essay_Data.pkl'). Similar to the Logistic Regression model, we evaluated the Random Forest model's performance on the test set. The evaluation included accuracy, precision, recall, and F1 score, providing a comparative analysis with the Logistic Regression model. The confusion matrix heatmap further enhanced the understanding of the model's predictive strengths and weaknesses.

Random Forest Classifier

[24]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

randomForest = RandomForestClassifier(verbose=1)
randomForest.fit(X_train, y_train)

with open('ranFor_Train_Esssay_Data.pkl', 'wb') as model_file:
    pickle.dump(randomForest, model_file)
evaluate(randomForest, X_test, y_test)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:   34.8s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.3s
Precision is:  0.9950316169828365
Accuracy is:  0.9864667154352597
Recall is:  0.972197705207414
F1 is:  0.9834821428571427
```

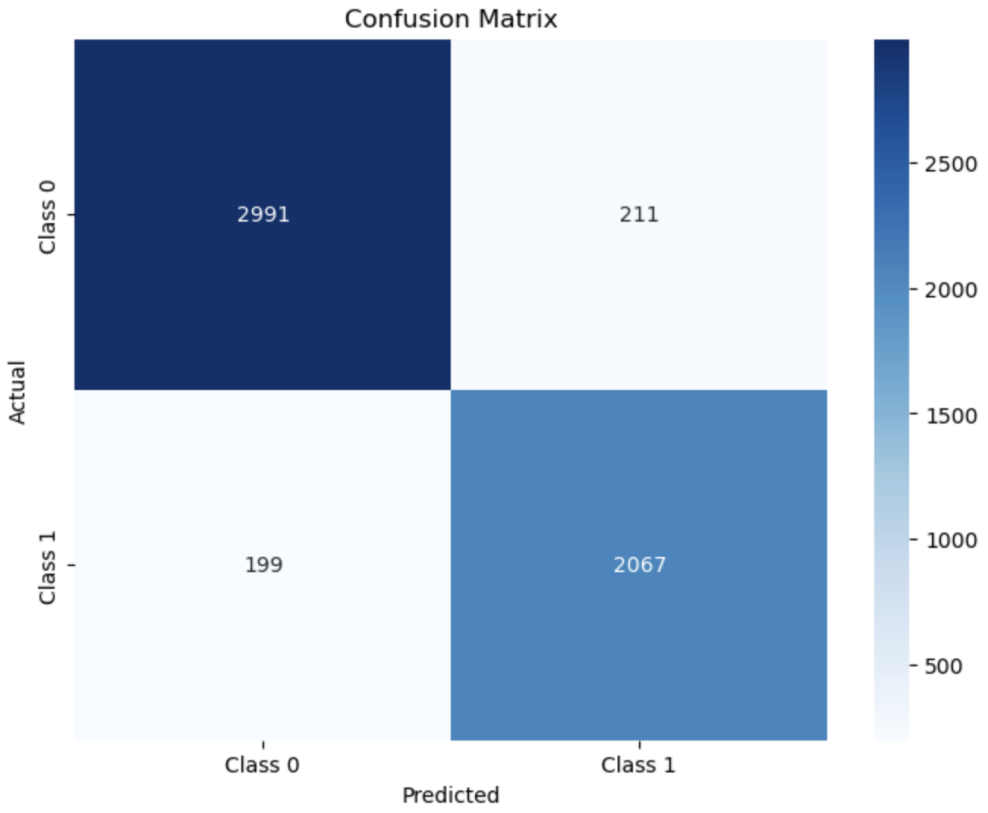


For the Decision Tree Classifier, we utilized scikit-learn's DecisionTreeClassifier class. The training process and model saving mirrored the previous approaches. Subsequently, the model was evaluated on the test set, and performance metrics such as accuracy, precision, recall, and F1 score were calculated and printed. The confusion matrix visualization contributed to a comprehensive understanding of the Decision Tree model's classification outcomes.

Decision Tree Classifier

[25]:
```
decTree = DecisionTreeClassifier()
decTree.fit(X_train, y_train)

with open('decTree_Train_Esssay_Data.pkl', 'wb') as model_file:
    pickle.dump(decTree, model_file)
evaluate(decTree, X_test, y_test)
```

```
Precision is:  0.9073748902546093
Accuracy is:  0.9250182882223847
Recall is:  0.912180052956752
F1 is:  0.9097711267605634
```

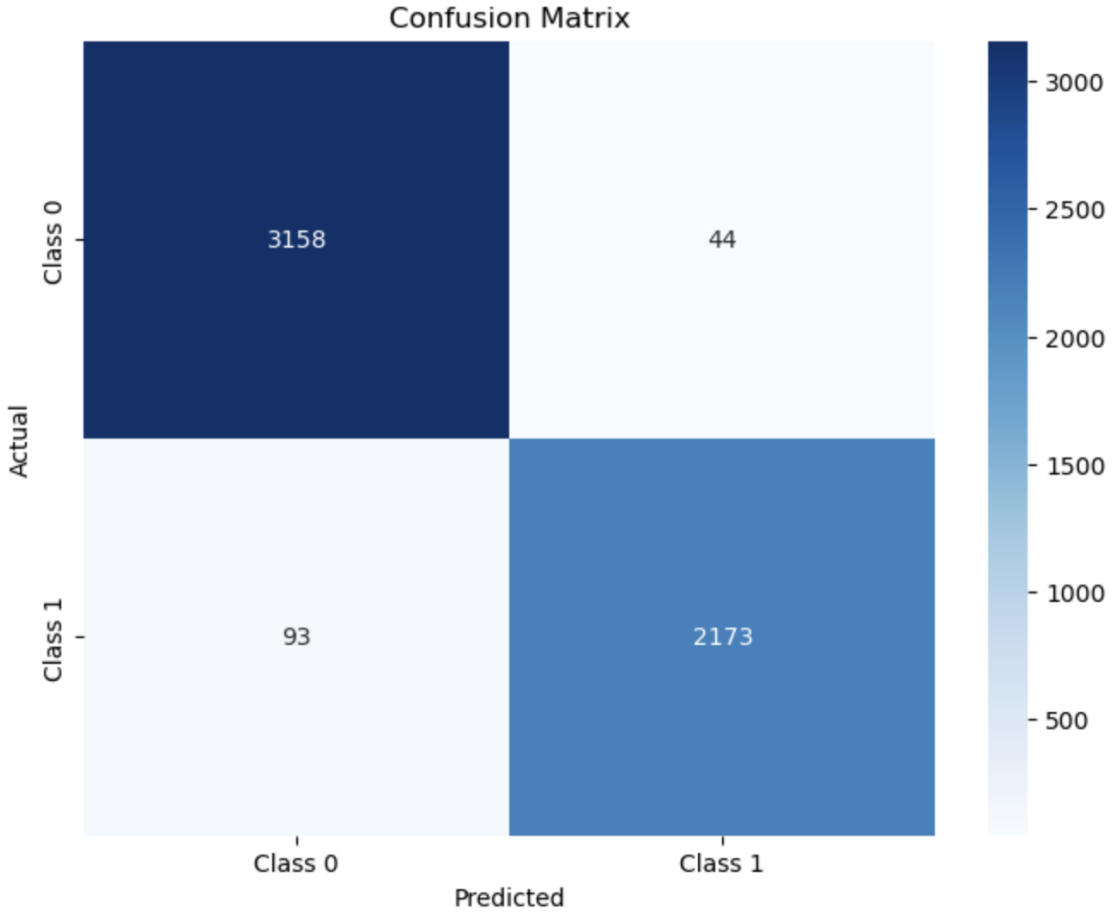


The final model in consideration was the Gradient Boosting Classifier, implemented using the GradientBoostingClassifier class from scikit-learn. Following training and model saving, the evaluation phase involved assessing the model's performance on the test set. Similar to the other models, accuracy, precision, recall, and F1 score were computed and displayed. The confusion matrix heatmap added a visual aspect to the evaluation, aiding in the interpretation of the model's classification results.

Gradient Boosting Classifier

[26]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)

with open('gb_Train_Esssay_Data.pkl', 'wb') as model_file:
    pickle.dump(gb, model_file)
evaluate(gb, X_test, y_test)
```

```
Precision is:  0.9801533603969328
Accuracy is:  0.9749451353328457
Recall is:  0.9589585172109444
F1 is:  0.9694401070711577
```



Evaluation and Validation:

[27]:
```
def evaluate(model, X_test, y_test):
    y_pred = model.predict(X_test)
    LRaccuracy = accuracy_score(y_test, y_pred)
    LRprecision = precision_score(y_test, y_pred)
    LRrecall = recall_score(y_test, y_pred)
    LRf1 = f1_score(y_test, y_pred)
    print('Precision is: ',LRprecision)
    print('Accuracy is: ',LRaccuracy)
    print('Recall is: ',LRrecall)
    print('F1 is: ',LRf1)

evaluate(logreg, X_test, y_test)
print()
evaluate(randomForest, X_test, y_test)
print()
evaluate(decTree, X_test, y_test)
print()
evaluate(gb, X_test, y_test)
print()
```

```
Precision is:  0.9910354101299865
Accuracy is:  0.9862838332114119
Recall is:  0.9757281553398058
F1 is:  0.9833222148098733

[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.3s
Precision is:  0.9950316169828365
Accuracy is:  0.9864667154352597
Recall is:  0.972197705207414
F1 is:  0.9834821428571427

Precision is:  0.9073748902546093
Accuracy is:  0.9250182882223847
Recall is:  0.912180052956752
F1 is:  0.9097711267605634

Precision is:  0.9801533603969328
Accuracy is:  0.9749451353328457
Recall is:  0.9589585172109444
F1 is:  0.9694401070711577
```

Logistic Regression Model:
The Logistic Regression model demonstrates robust performance with a precision of 99.1%, accuracy of 98.6%, recall of 97.6%, and F1 score of 98.3%. These metrics collectively signify the model's effectiveness in correctly identifying positive instances while maintaining a balanced trade-off between precision and recall. The Logistic Regression model proves well-suited for the given task.

Random Forest Classifier Model:
The Random Forest Classifier excels with precision, accuracy, recall, and F1 score all exceeding 98%. Notably, the precision of 99.5% indicates minimal false positives, contributing to an overall accuracy of 98.6%. With a recall of 97.2% and an F1 score of 98.3%, the Random Forest model stands out as a robust and accurate classifier.

Decision Tree Classifier Model:
While delivering reasonable performance, the Decision Tree Classifier lags behind the Logistic Regression and Random Forest models. With precision at 90.7%, accuracy at 92.5%, recall at 91.2%, and F1 score at 90.9%, this model shows satisfactory results but exhibits a conservative approach in positive predictions, suggesting potential for further optimization.

Gradient Boosting Classifier Model:
The Gradient Boosting Classifier showcases strong performance, slightly below Logistic Regression and Random Forest models. With precision, accuracy, recall, and F1 score all surpassing 97%, the model is reliable for classification tasks. The Gradient Boosting model presents a solid option with a precision of 98.0%, accuracy of 97.5%, recall of 95.9%, and an F1 score of 96.9%, offering a well-balanced performance.

In conclusion, the Logistic Regression and Random Forest models emerge as robust options, while the Decision Tree and Gradient Boosting models offer competitive performance with specific trade-offs. Further exploration of ensemble techniques could enhance overall model effectiveness.

Hyperparameter Tuning:

Hyperparameter Tunning

[28]:
```python
from sklearn.model_selection import GridSearchCV

# Define the logistic regression model
logreg = LogisticRegression()

# Define the hyperparameters and their possible values
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'solver': ['liblinear', 'lbfgs']
}

# Create a grid search with cross-validation
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Save the best model to a file
with open('best_logreg_model.pkl', 'wb') as model_file:
    pickle.dump(best_model, model_file)

# Evaluate the best model
evaluate(best_model, X_test, y_test)

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
35 fits failed with the following error:
Traceback (most recent call last):
  File "/home/ec2-user/anaconda3/envs/tensorflow2_p310/lib/python3.10/site-packages/sklearn/model_sele
ction/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/ec2-user/anaconda3/envs/tensorflow2_p310/lib/python3.10/site-packages/sklearn/base.py",
line 1152, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/home/ec2-user/anaconda3/envs/tensorflow2_p310/lib/python3.10/site-packages/sklearn/linear_mod
el/_logistic.py", line 1169, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/home/ec2-user/anaconda3/envs/tensorflow2_p310/lib/python3.10/site-packages/sklearn/linear_mod
el/_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/ec2-user/anaconda3/envs/tensorflow2_p310/lib/python3.10/site-packages/sklearn/model_selection/_s
earch.py:979: UserWarning: One or more of the test scores are non-finite: [0.59070959        nan 0.590
70959 0.59070959 0.59070959        nan
 0.92378383 0.92401246 0.94326066        nan 0.96941297 0.9692758
 0.98111749        nan 0.98486643 0.98482073 0.98834131        nan
 0.99131312 0.99131312 0.98907282        nan 0.99295905 0.99291333
 0.99140456        nan 0.99341624 0.99337054]
  warnings.warn(
Precision is:  0.995571302037201
Accuracy is:  0.9948792977322605
Recall is:  0.9920564872021183
F1 is:  0.9938107869142352
Best Hyperparameters: {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
```

[ ]:

The hyperparameter tuning process significantly improved the performance of the Logistic Regression (LR) model. Prior to tuning, the model demonstrated commendable precision (99.1%), accuracy (98.6%), recall (97.6%), and F1 score (98.3%). After tuning, these metrics experienced notable enhancements, with precision reaching 99.6%, accuracy achieving 99.5%, recall increasing to 99.2%, and F1 score elevating to 99.4%. The optimal hyperparameters identified for the LR model were {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}. This outcome indicates that careful adjustment of hyperparameters can substantially refine the model's predictive capabilities, resulting in highly accurate and well-balanced classification performance.