Detecting Phishing Emails

Lincoln Brown

Project 3 Milestone 2

DSC680-T301

Professor Iranitalab

Business Problem

Emails are a popular platform for attackers to gain access to a network. Using phishing emails, malicious actors harvest user credentials which they can then use to gain access to sensitive information or to impersonate the end user and continue their attack on associated businesses and entities. Phishing emails are a wide-spread business problem and can be a critical vector of attack for data breaches. This project is focused on producing a model that can be used to strengthen phishing detection using models trained on data obtained from Kaggle.

Background/History

Phishing emails are one of the most common and effective cyber-attacks, targeting both individuals and organizations. Scammers often use phishing emails to trick users into giving them personal and financial information, including passwords, account numbers, or Social Security numbers. This information can then be used to gain access to email accounts, banks, or to sell for profit to other malicious actors. Phishing emails represent a very real and costly problem for businesses and end users alike.

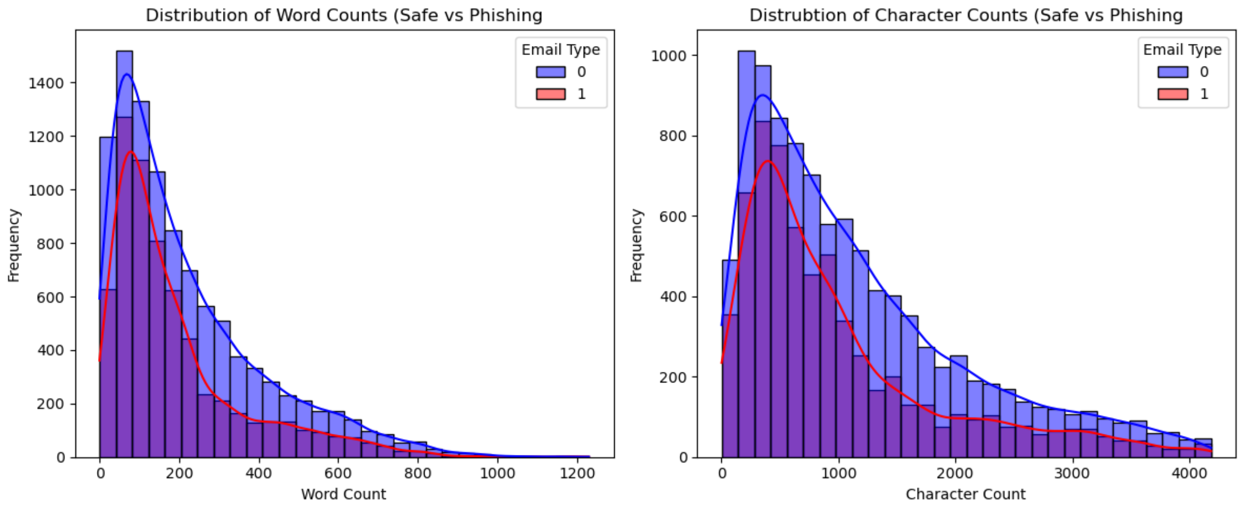
Using machine learning models to detect phishing emails can be used to prevent these emails from reaching end users. Machine learning models can be trained on several features of emails including header information (to/from email addresses), message body linguistic patterns, and embedded links. Due to the limitations of the dataset used in this project, the models will only be trained on message body linguistic patterns. This technology can be implemented by businesses to help reduce the amount of phishing emails that an end user receives, thereby reducing the risk of successful attacks and improving overall operational security.

Data Explanation

The dataset used in this project is the Phishing Email Detection found on Kaggle, uploaded by user Cyber Cop. This dataset contains two features: Email Text and Email Type. The email text feature contains the body of the email, and the email type is the associated classification label phishing or safe.

Initial data preparation consisted of removing null values and records that had empty email text. Furthermore, some records were removed based on the length of the email message. Outliers were removed using the interquartile range method (IQR). Additional preprocessing included removing stop words and characters from the email text, as well as applying a binary encoder to the ‘Email Type’ feature. Additionally, Term Frequency-Inverse Document Frequency (TF-IDF) is used to convert textual content into numerical form for use in the Logistic Regression model. The final dataset used to train the models contained 16,439 records.

Exploratory Data Analysis (EDA) began with getting word and character counts for the email text. These counts were used to investigate any potential differences between the average word or character counts for safe vs phishing emails. Additionally, sentiment analysis was performed to see if phishing emails had a trend of having either negative or positive sentiment in their text.



Methods

This project evaluates two machine learning models for detecting phishing emails: a Logistic Regression model and a more advanced Bidirectional Encoder Representation from Transformers (BERT) model.

Logistic Regression Model – Logistic Regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not (GeeksforGeeks, 2025).

BERT Model – BERT is a transformer-based neural network designed to be used for natural language processing (NLP). BERT is a bidirectional model, meaning that it considers both the left and right context of words in a sentence (GeeksforGeeks, 2024).

Both models are trained and tested using the dataset containing legitimate and phishing emails. The models' performance is evaluated using the performance metrics accuracy, precision, rec all, F1-score, and ROC-AUC score. A confusion matrix is used to compare the number of false positives and false negatives to determine the model's effectiveness at classification.

Analysis

The models both performed well, with the LR model achieving 97.75% accuracy and the BERT model achieving 98.57% accuracy. The BERT model performed slightly better than the Logistic Regression across all metrics. This suggests that the BERT model is slightly more capable at correctly identifying phishing emails and minimizing false negatives.

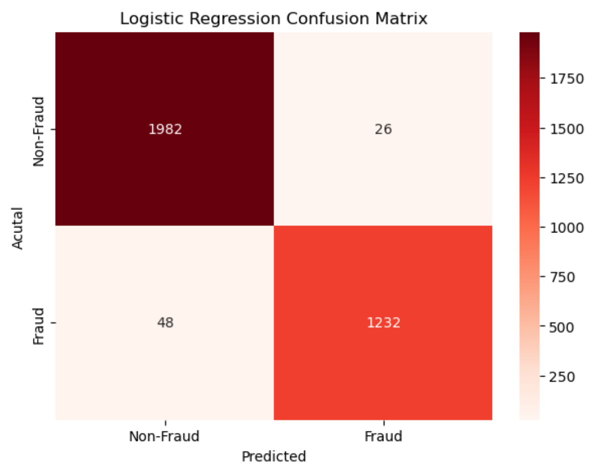
Logistic Regression Model Results:

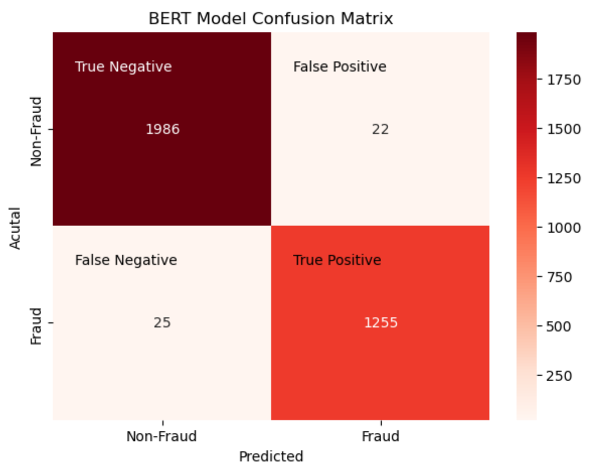
Metric	Class 0 (Negative)	Class 1 (Positive)	Macro Avg	Weighted Avg
Precision	0.98	0.98	0.98	0.98
Recall	0.99	0.96	0.97	0.98
F1-Score	0.98	0.97	0.98	0.98
Support	2008	1280	3288	3288

BERT Model Results:

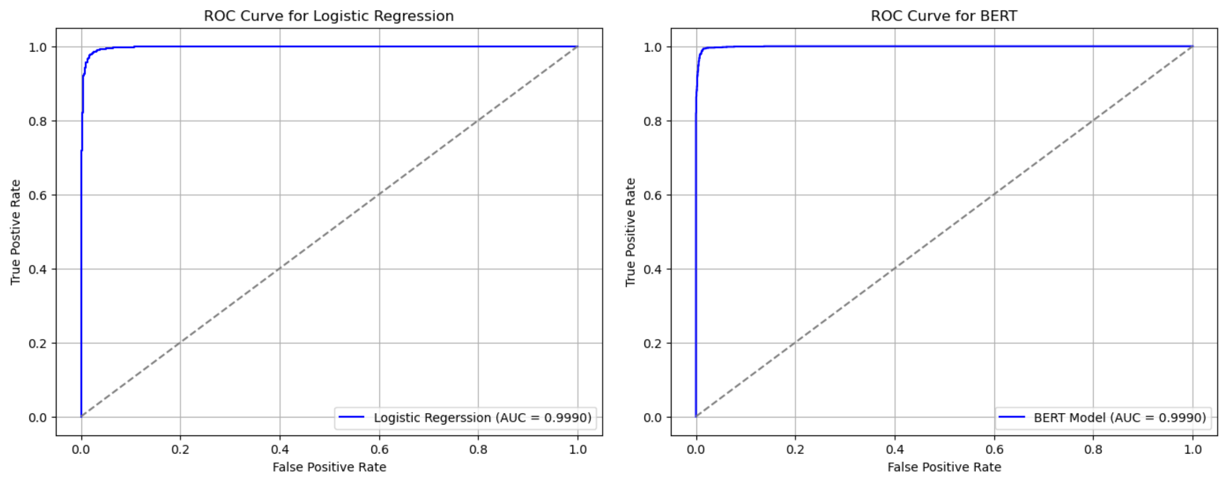
Metric	Class 0 (Negative)	Class 1 (Positive)	Macro Avg	Weighted Avg
Precision	0.99	0.98	0.99	0.99
Recall	0.99	0.98	0.98	0.99
F1-Score	0.99	0.98	0.98	0.99
Support	2008	1280	3288	3288

From the confusion matrices below, we can see the rates of classification error rates for each category.





The ROC Curves for both models had a 99% AUC (Area Under the Curve), indicating that both models are highly effective at distinguishing between phishing and legitimate emails.



Conclusion

This project evaluated the effectiveness of detecting phishing emails using two machine learning models, Logistic Regression and BERT. Both models demonstrated high accuracy, with the Logistic Regression model achieving 97.75% accuracy and the BERT model achieving slightly better with 98.57% accuracy. The ROC AUC scores of 99% for both models demonstrate that both models have strong capability when differentiating between legitimate and phishing emails.

The models both performed very well, with the BERT model performing slightly better. However, this slight increase in performance comes at a high computational cost. The Logistic Regression model was able to compute much quicker. Considering this, the BERT model is an ideal candidate for deployment in environments where superior classification capabilities are crucial, even at a higher computational cost. The Logistic Regression model, however, is a very efficient model that still performs well and may be better for deployment in organizations that would rather balance efficiency with performance.

In conclusion, both models were capable of detecting phishing emails. While the BERT model has slightly better performance, it comes at a computational cost. Furthermore, neither model is going to identify every phishing email, so these models should not be deployed as an alternative to comprehensive user training. End users are the main target of phishing emails and through effective user training, can reduce the threat of compromise. These models, in combination with end user training offer another risk mitigation strategy for organizations to deploy to address the growing threat of phishing emails and to reduce their organization's overall risk.

Assumptions

Several assumptions were made in this project to ensure the performance of the model's and effective evaluation of that performance.

It is assumed that the dataset is balanced. The dataset does not contain a perfectly balanced dataset, but it was assumed that the slight imbalance was not enough to skew the results of the models being evaluated.

Additionally, it is assumed that the dataset is labelled correctly and is an accurate representation of real-world scenarios. The capability of the models' performance rides on the accurate labeling of the input data. It is assumed that these labels were accurate. Furthermore, it is assumed that the emails in the dataset are an accurate representation of legitimate and phishing emails. Given the ever-evolving nature of cyberattacks, it would be prudent to retrain the models on the newest data available to ensure effectiveness against current threats.

Limitations

There are some key limitations to this project. First, cyberattacks are constantly changing. Phishing emails that were once effective are now being recognized by most filters and have been changed by the scammers to something different to avoid the filters. Due to this, the dataset is likely out of date and may not be as effective against current phishing emails. It would be necessary to train this model on more current data before deployment.

Second, this dataset was limited to only the email message body text being available for analysis. Incorporating more meta data from the emails such as header info and the presence of external links could improve the performance of these models.

Finally, the models were trained on 16,439 records, which is a medium-sized dataset. This is a decently sized dataset for models like Logistic Regression, but it is common for deep learning models such as BERT to be trained on bigger datasets. The size of the dataset can introduce the risk of overfitting, which can be addressed by incorporating more data.

Challenges

There were a couple of challenges with this project. First, email messages tend to be messy, so it was necessary to preprocess the data by removing special characters, stop words, and implement tokenization so that the data was ready for the models. Secondly, the BERT model is complex and was challenging to build. Additionally, the BERT model is computationally expensive and took upwards of an hour to train.

Future Uses/Additional Applications

The models in this project were developed as an educational exercise, but the underlying technology has applicability in the real-world. With further fine tuning, these models could be deployed in a business scenario.

Recommendations

Further improvements can be made to these models with more up-to-date datasets, larger datasets, and more features. With these improvements, these models would be better suited for deployment in a real-world scenario and would be better suited for identifying current phishing emails in dynamic environments.

Implementation Plan

Implementing these models in a business environment would need to consider several key points. First, the computational capabilities of the business would determine the type of model to implement. Businesses that have greater capabilities may choose to implement the BERT model, while smaller businesses may opt for the less computationally expensive Logistic Regression model. Next, the business needs to determine where the data used for model training will be obtained from and how the model will be deployed (locally vs cloud). Additional considerations could include whether the model will be part of a continuous integration pipeline where the model is regularly updated. Finally, the business needs to consider the available staff to maintain and deploy the model or if there are commercially available products that have a better or similar cost-benefit analysis.

Ethical Assessment

This project had limited ethical implications. The dataset used is publicly available and licensed under the GNU Lesser Public License 3.0. The dataset did not contain any private email addresses or metadata information from the emails. Implementing phishing email detection models in business environments can introduce privacy concerns that need to be properly addressed to avoid any potential leaks of Personally Identifiable Information (PII).

References

Albon, C. (2018). *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*.

Erim, E. (2024, March 26). Complete guide to building a text classification model using BERT. *Medium*. https://medium.com/@emreerim_65318/complete-guide-to-building-a-text-classification-model-using-bert-abf27b5cb6a1

GeeksforGeeks. (2024, December 10). *BERT Model NLP*. GeeksforGeeks. https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/

GeeksforGeeks. (2025, February 3). *Logistic regression in machine learning*. GeeksforGeeks. https://www.geeksforgeeks.org/understanding-logistic-regression/

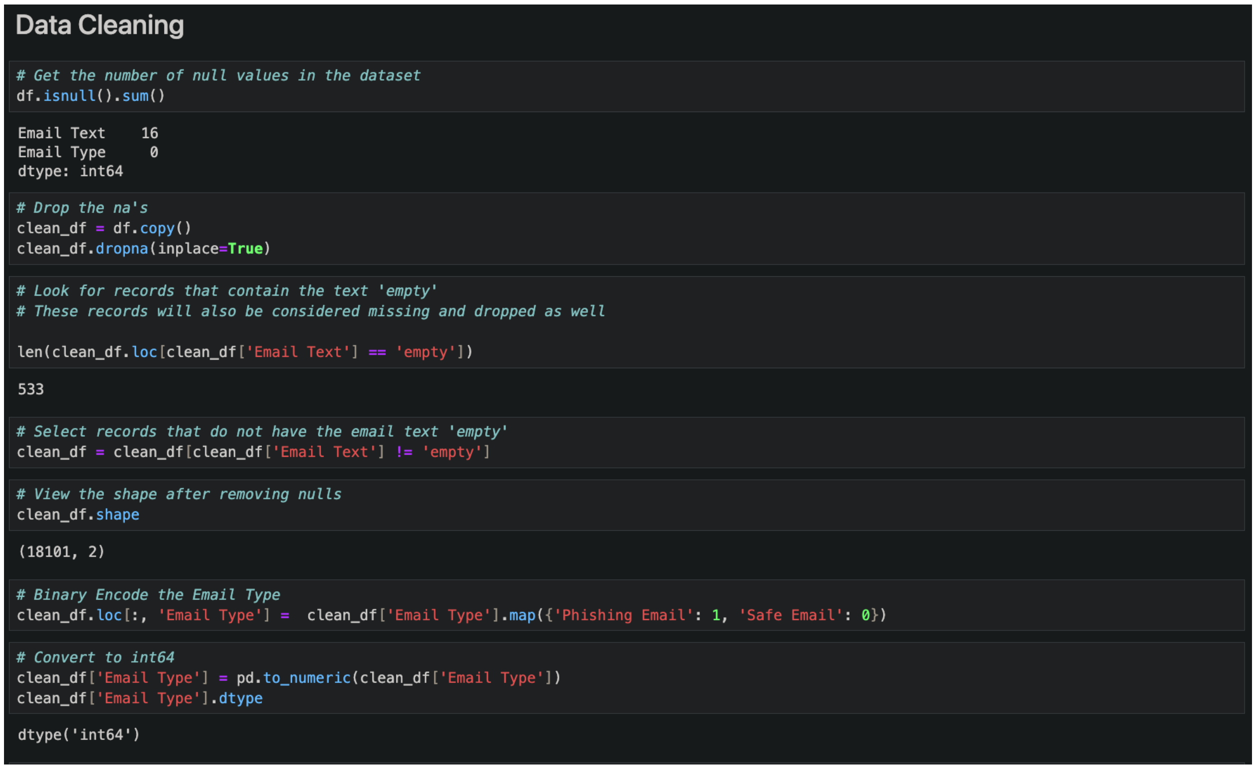
Phishing email Detection. (n.d.). https://www.kaggle.com/datasets/subhajournal/phishingemails

Science, D. (2023, July 5). *Building text classification models using BERT*. Data-Driven Science. https://datadrivenscience.com/building-text-classification-models-using-bert/

Appendix

Below are some of the EDA visualizations and code snippets that are not included in the white paper.

Data cleaning steps:



Data Cleaning

```
# Get the number of null values in the dataset
df.isnull().sum()
```

```
Email Text    16
Email Type     0
dtype: int64
```

```
# Drop the na's
clean_df = df.copy()
clean_df.dropna(inplace=True)
```

```
# Look for records that contain the text 'empty'
# These records will also be considered missing and dropped as well

len(clean_df.loc[clean_df['Email Text'] == 'empty'])
```

```
533
```

```
# Select records that do not have the email text 'empty'
clean_df = clean_df[clean_df['Email Text'] != 'empty']
```

```
# View the shape after removing nulls
clean_df.shape
```

```
(18101, 2)
```

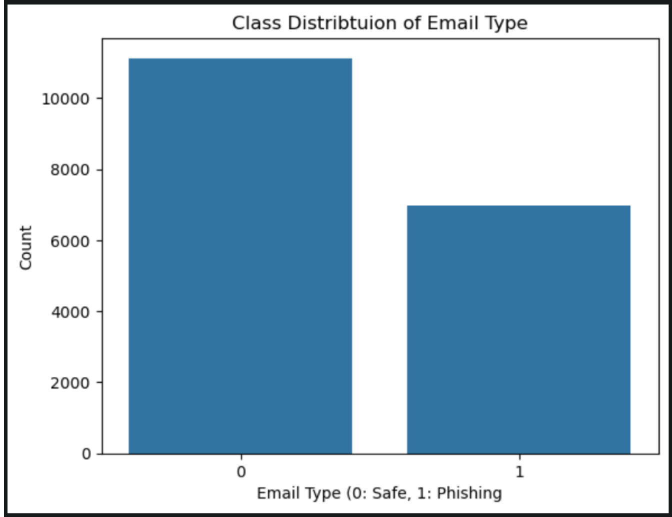
```
# Binary Encode the Email Type
clean_df.loc[:, 'Email Type'] =  clean_df['Email Type'].map({'Phishing Email': 1, 'Safe Email': 0})
```

```
# Convert to int64
clean_df['Email Type'] = pd.to_numeric(clean_df['Email Type'])
clean_df['Email Type'].dtype
```

```
dtype('int64')
```

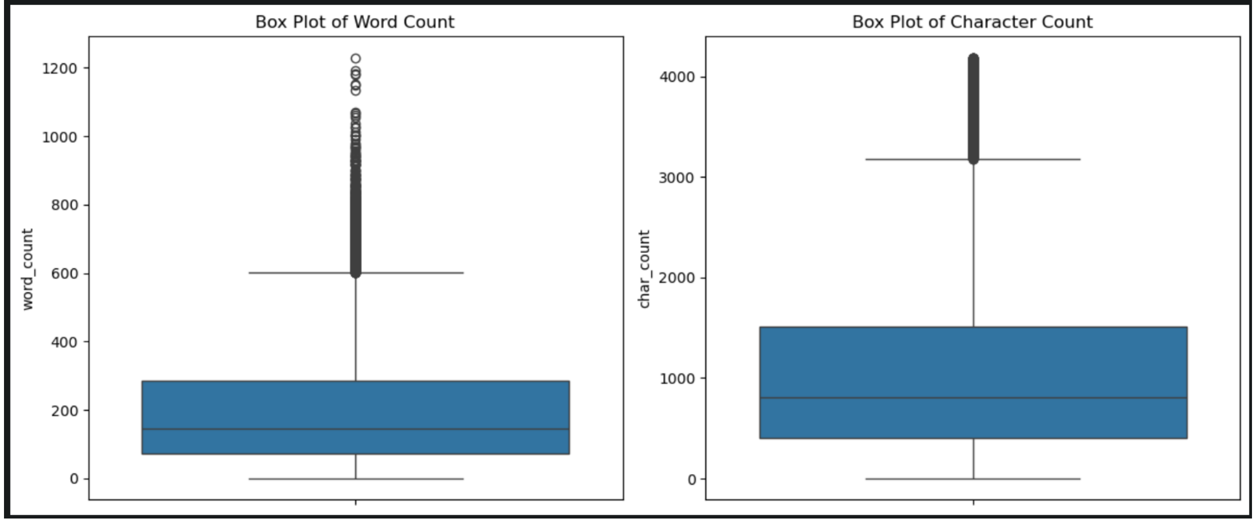
Class Distribution

```
# Create a count plot of the class distribution
sns.countplot(x=eda_df['Email Type'])
plt.title('Class Distribtuion of Email Type')
plt.xlabel('Email Type (0: Safe, 1: Phishing')
plt.ylabel('Count')
plt.show()
```



Box plots of word/character counts after outliers have been removed using IQR:

```python
# Review the distributions after removing outliers
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Box plot of word_count
sns.boxplot(filtered_df['word_count'], ax=axes[0])
axes[0].set_title('Box Plot of Word Count')

# Box plot of char_count
sns.boxplot(filtered_df['char_count'], ax=axes[1])
axes[1].set_title('Box Plot of Character Count')

plt.tight_layout()
plt.show()
```

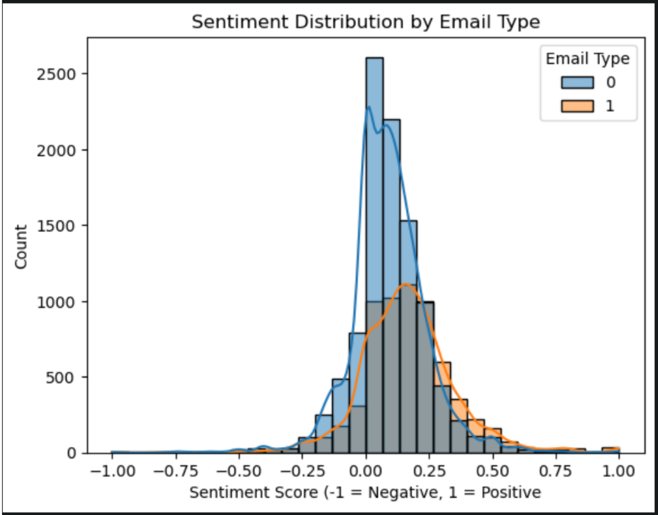


Sentiment analysis of phishing vs legitimate emails:

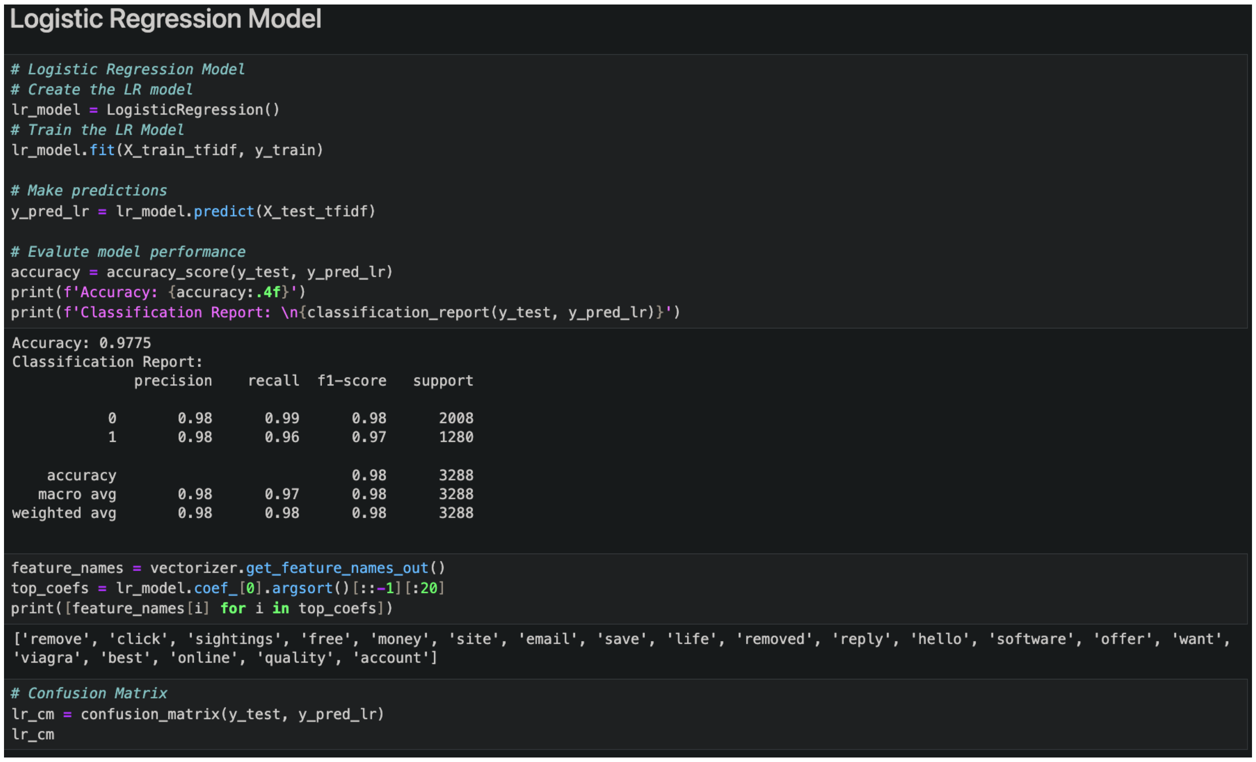
Sentiment Analysis

```python
filtered_df['sentiment'] = filtered_df.loc[:,'Email Text'].apply(lambda x: TextBlob(x).sentiment.polarity)
```

```python
# Plot sentiment distribution
sns.histplot(filtered_df, x='sentiment', hue='Email Type', bins=30, kde=True)
plt.title('Sentiment Distribution by Email Type')
plt.xlabel('Sentiment Score (-1 = Negative, 1 = Positive')
plt.show()
```



Logistic Regression Model Creation:



Logistic Regression Model

```python
# Logistic Regression Model
# Create the LR model
lr_model = LogisticRegression()
# Train the LR Model
lr_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_lr = lr_model.predict(X_test_tfidf)

# Evalute model performance
accuracy = accuracy_score(y_test, y_pred_lr)
print(f'Accuracy: {accuracy:.4f}')
print(f'Classification Report: \n{classification_report(y_test, y_pred_lr)}')
```

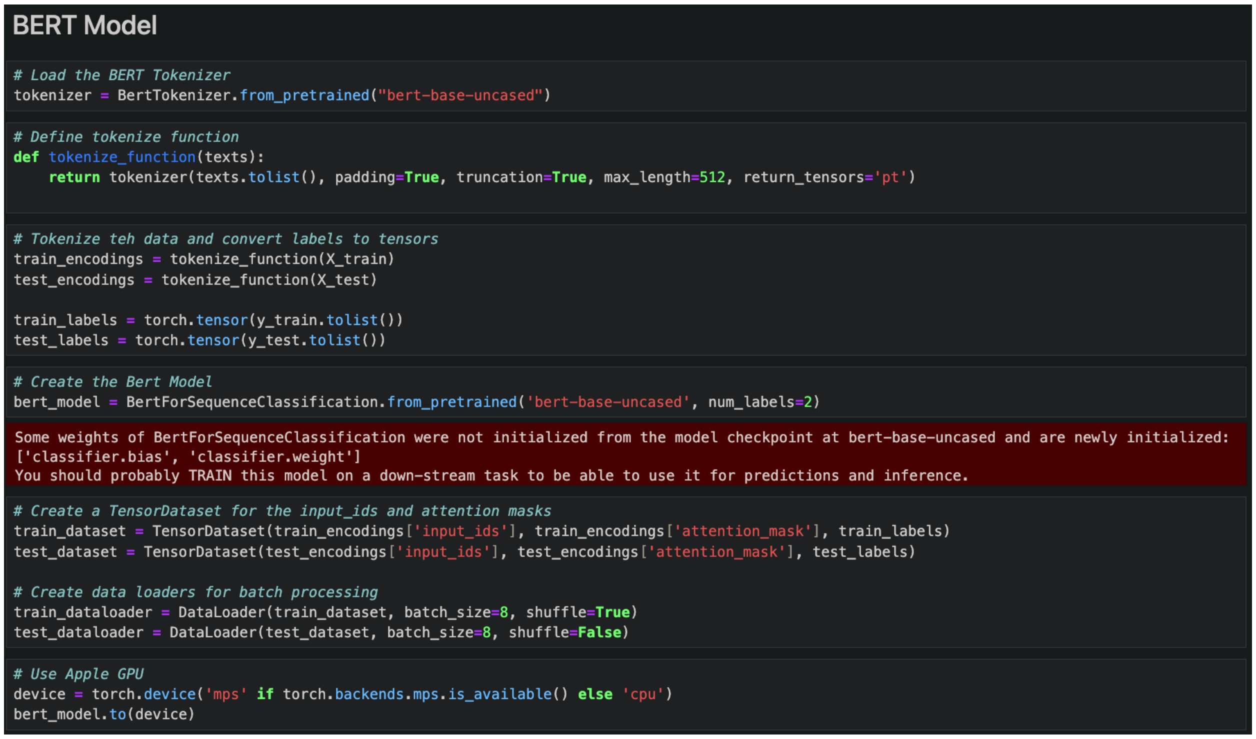
```
Accuracy: 0.9775
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2008
           1       0.98      0.96      0.97      1280

    accuracy                           0.98      3288
   macro avg       0.98      0.97      0.98      3288
weighted avg       0.98      0.98      0.98      3288
```

```python
feature_names = vectorizer.get_feature_names_out()
top_coefs = lr_model.coef_[0].argsort()[::-1][:20]
print([feature_names[i] for i in top_coefs])
```

```
['remove', 'click', 'sightings', 'free', 'money', 'site', 'email', 'save', 'life', 'removed', 'reply', 'hello', 'software', 'offer', 'want',
'viagra', 'best', 'online', 'quality', 'account']
```

```python
# Confusion Matrix
lr_cm = confusion_matrix(y_test, y_pred_lr)
lr_cm
```

BERT Model Creation



BERT Model

```python
# Load the BERT Tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

```python
# Define tokenize function
def tokenize_function(texts):
    return tokenizer(texts.tolist(), padding=True, truncation=True, max_length=512, return_tensors='pt')
```

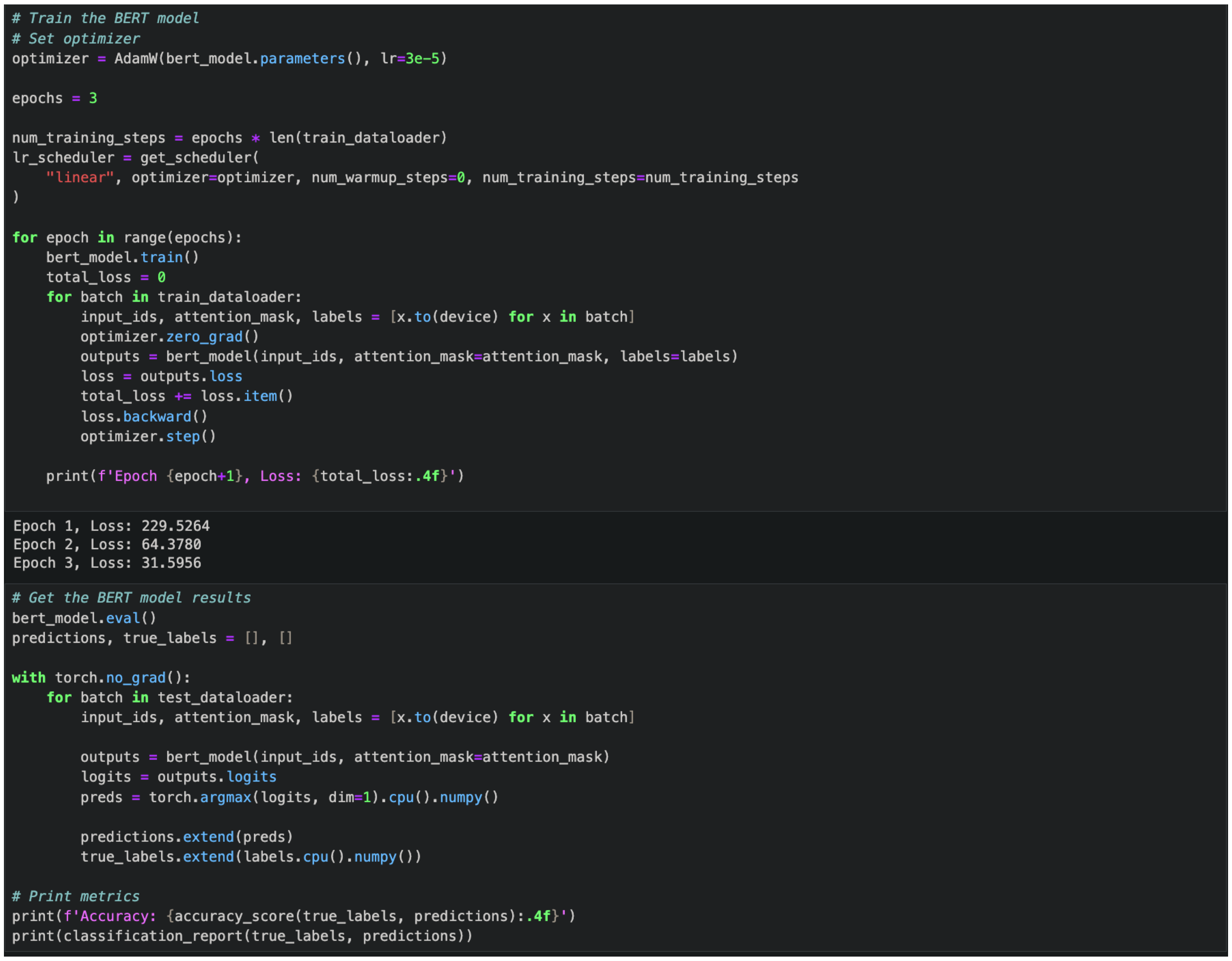
```python
# Tokenize teh data and convert labels to tensors
train_encodings = tokenize_function(X_train)
test_encodings = tokenize_function(X_test)

train_labels = torch.tensor(y_train.tolist())
test_labels = torch.tensor(y_test.tolist())
```

```python
# Create the Bert Model
bert_model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```python
# Create a TensorDataset for the input_ids and attention masks
train_dataset = TensorDataset(train_encodings['input_ids'], train_encodings['attention_mask'], train_labels)
test_dataset = TensorDataset(test_encodings['input_ids'], test_encodings['attention_mask'], test_labels)

# Create data loaders for batch processing
train_dataloader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=8, shuffle=False)
```

```python
# Use Apple GPU
device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
bert_model.to(device)
```



```
# Train the BERT model
# Set optimizer
optimizer = AdamW(bert_model.parameters(), lr=3e-5)

epochs = 3

num_training_steps = epochs * len(train_dataloader)
lr_scheduler = get_scheduler(
    "linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=num_training_steps
)

for epoch in range(epochs):
    bert_model.train()
    total_loss = 0
    for batch in train_dataloader:
        input_ids, attention_mask, labels = [x.to(device) for x in batch]
        optimizer.zero_grad()
        outputs = bert_model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch+1}, Loss: {total_loss:.4f}')
```

```
Epoch 1, Loss: 229.5264
Epoch 2, Loss: 64.3780
Epoch 3, Loss: 31.5956
```

```
# Get the BERT model results
bert_model.eval()
predictions, true_labels = [], []

with torch.no_grad():
    for batch in test_dataloader:
        input_ids, attention_mask, labels = [x.to(device) for x in batch]

        outputs = bert_model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()

        predictions.extend(preds)
        true_labels.extend(labels.cpu().numpy())

# Print metrics
print(f'Accuracy: {accuracy_score(true_labels, predictions):.4f}')
print(classification_report(true_labels, predictions))
```

Ten Questions an audience might ask:

1. How does the BERT model compare to Logistic Regression in terms of accuracy and performance?
2. What is the primary difference between the Logistic Regression and the BERT models, why did you use both?
3. What specific features from the email data did you use to train the model?
4. How does the model handle new phishing techniques that it has never seen before?
5. Can the model be integrated into existing email systems? If so, how?
6. How does the model balance false positives and false negatives? Are there trade-offs in these rates?
7. What challenges did you encounter while working with this dataset?
8. How do the models handle variations in the language used in phishing emails, such as slang or different writing styles?
9. Can this model be deployed for real-time phishing detection?
10. How could the performance of these models be improved?