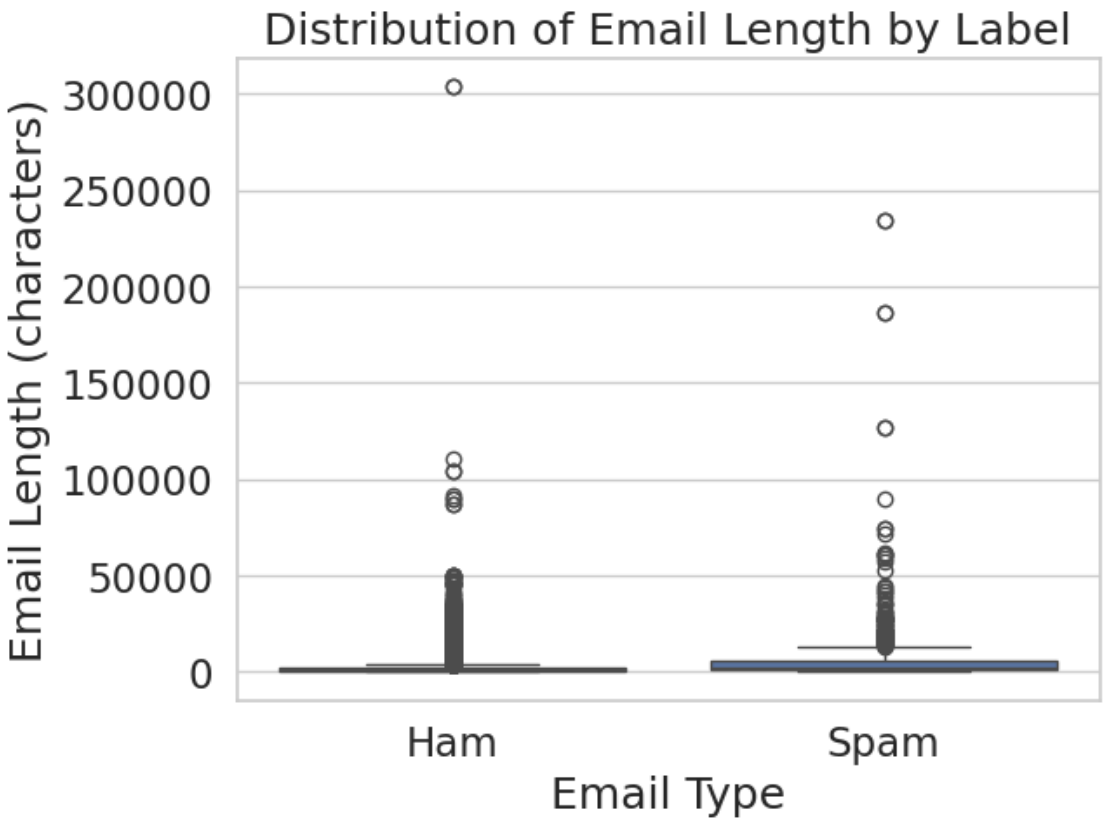
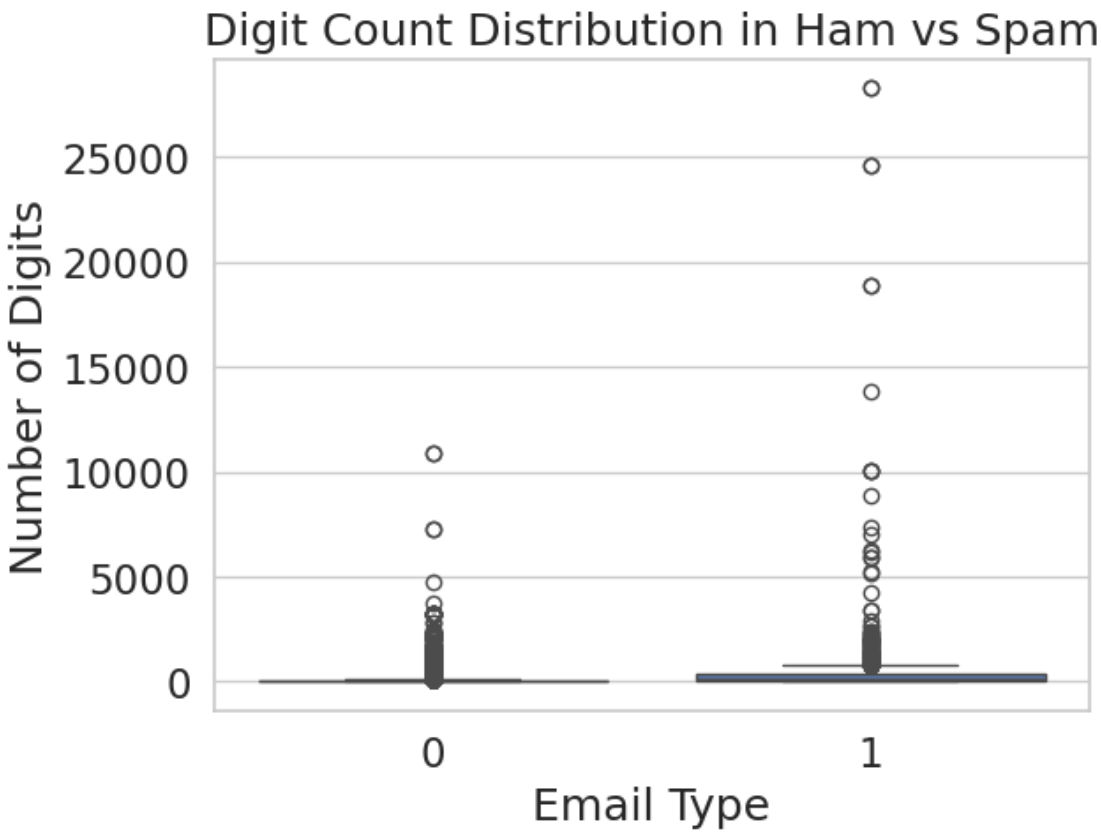
## 0.1 Question 1a

Generate your visualization in the cell below.

```
In [11]: import matplotlib.pyplot as plt
         import seaborn as sns

         # Add a new feature: email length
         train['email_length'] = train['email'].apply(len)

         # Plot the distribution
         sns.boxplot(x='spam', y='email_length', data=train)
         plt.xticks([0, 1], ['Ham', 'Spam'])
         plt.title('Distribution of Email Length by Label')
         plt.xlabel('Email Type')
         plt.ylabel('Email Length (characters)')
         plt.show()
```

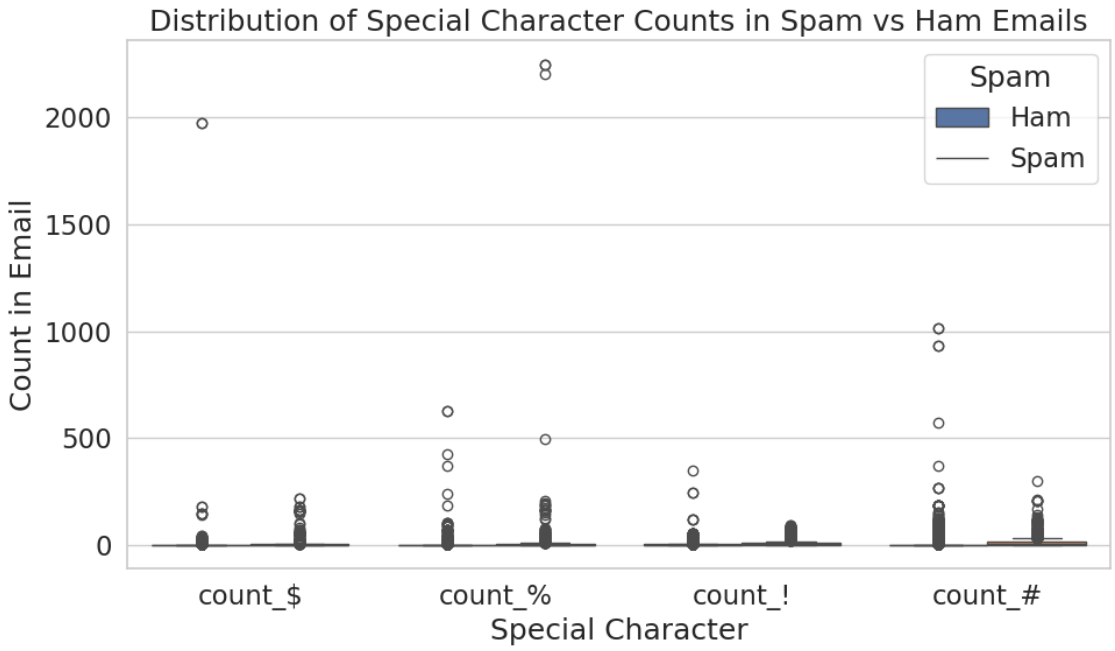


In [12]:
```
import re

# Count digits in each email
train['digit_count'] = train['email'].apply(lambda x: len(re.findall(r'\d', x)))

sns.boxplot(x='spam', y='digit_count', data=train)
plt.title('Digit Count Distribution in Ham vs Spam')
plt.xlabel('Email Type')
plt.ylabel('Number of Digits')
plt.show()
```




In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define special characters to check
```

```
special_chars = ['$', '%', '!', '#']

# Create a new feature: count of special characters in each email
for char in special_chars:
    train[f'count_{char}'] = train['email'].apply(lambda x: x.count(char))

# Melt the data to long format for easier plotting with seaborn
special_counts = train[[f'count_{char}' for char in special_chars] + ['spam']]
long_format = special_counts.melt(id_vars='spam', var_name='char', value_name='count')

# Plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=long_format, x='char', y='count', hue='spam')
plt.title('Distribution of Special Character Counts in Spam vs Ham Emails')
plt.xlabel('Special Character')
plt.ylabel('Count in Email')
plt.legend(title='Spam', labels=['Ham', 'Spam'])
plt.tight_layout()
plt.show()

# === Summary Statistics for Special Characters by Spam Label ===
summary_stats = {}
for char in special_chars:
    group_stats = train.groupby('spam')[f'count_{char}'].describe()
    summary_stats[char] = group_stats

# Combine into one big table
summary_df = pd.concat(summary_stats, axis=1)
summary_df.columns.names = ['char', 'stat']

# Display the summary table
summary_df
```




Out[13]:

| char | $ | | | | | | | | % | \ |
|---|---|---|---|---|---|---|---|---|---|---|
| stat | count | mean | std | min | 25% | 50% | 75% | max | count | |
| spam | | | | | | | | | | |
| 0 | 5595.0 | 1.397855 | 37.797170 | 0.0 | 0.0 | 0.0 | 0.0 | 1977.0 | 5595.0 | |
| 1 | 1918.0 | 3.973931 | 17.496951 | 0.0 | 0.0 | 0.0 | 2.0 | 218.0 | 1918.0 | |

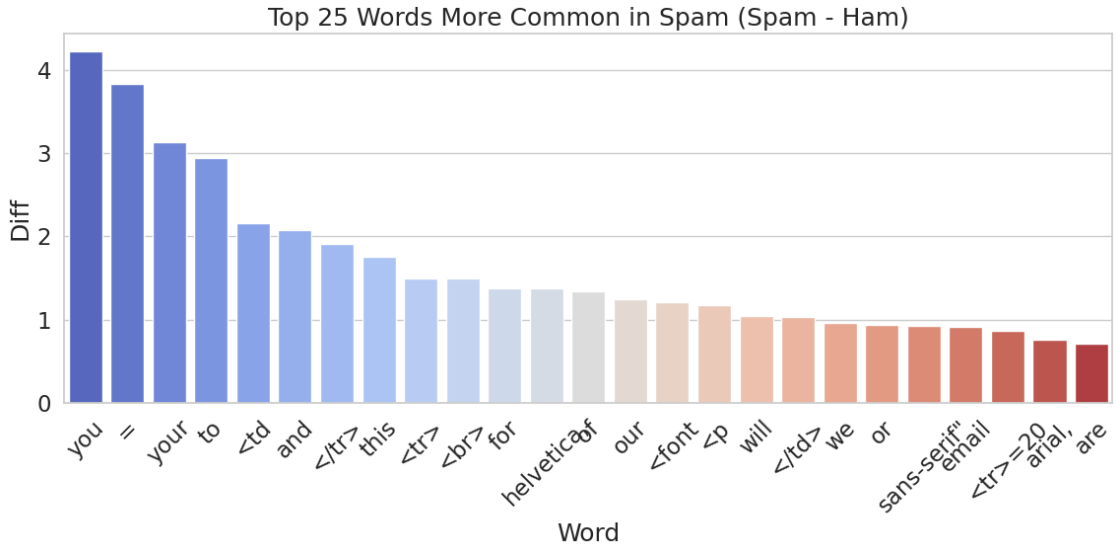
| char | … | ! | | | # | | | | | \ |
|---|---|---|---|---|---|---|---|---|---|---|
| stat | mean | … | 75% | max | count | mean | std | min | 25% | 50% |
| spam | | … | | | | | | | | |
| 0 | 1.550492 | … | 1.0 | 345.0 | 5595.0 | 5.695800 | 34.702300 | 0.0 | 0.0 | 0.0 |
| 1 | 8.919708 | … | 7.0 | 92.0 | 1918.0 | 10.400417 | 22.099416 | 0.0 | 0.0 | 1.0 |

| char | | |
|---|---|---|
| stat | 75% | max |
| spam | | |
| 0 | 0.0 | 1015.0 |
| 1 | 12.0 | 296.0 |

[2 rows x 32 columns]

In [14]:
```
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
def calculate_frequency_difference(train_df, label_column='spam', text_column='email'):
    # Separate spam and ham
    spam_text = train_df[train_df[label_column] == 1][text_column].str.cat(sep=' ').lower()
    ham_text = train_df[train_df[label_column] == 0][text_column].str.cat(sep=' ').lower()

    # Tokenize
    spam_words = spam_text.split()
    ham_words = ham_text.split()

    # Count word frequencies
    spam_freq = Counter(spam_words)
    ham_freq = Counter(ham_words)
    all_words = set(spam_freq.keys()).union(ham_freq.keys())

    # Total number of emails
    total_spam = train_df[train_df[label_column] == 1].shape[0]
    total_ham = train_df[train_df[label_column] == 0].shape[0]

    rows = []
    for word in all_words:
        spam_prop = spam_freq[word] / total_spam
        ham_prop = ham_freq[word] / total_ham
        diff = spam_prop - ham_prop
        rows.append((word, spam_prop, ham_prop, diff))

    df_diff = pd.DataFrame(rows, columns=['Word', 'Spam_Prop', 'Ham_Prop', 'Diff'])

    # Only keep words that are more common in spam
    df_diff = df_diff[df_diff['Diff'] > 0]

    return df_diff.sort_values(by='Diff', ascending=False)

# Run and visualize
diff_df = calculate_frequency_difference(train)
top_words = diff_df.head(25)

plt.figure(figsize=(12, 6))
sns.barplot(data=top_words, x='Word', y='Diff', palette='coolwarm')
plt.title('Top 25 Words More Common in Spam (Spam - Ham)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Print the list of top 15 spam-heavy words
top_words_list = top_words['Word'].tolist()
print(top_words_list)
```




```
['you', '=', 'your', 'to', '<td', 'and', '</tr>', 'this', '<tr>', '<br>', 'for', 'helvetica,', 'of', 'ou
```

## 0.2 Question 1b

In two to three sentences, describe what you plotted and its implications with respect to your features.

I explored features like email length, digit count, special character usage, and word frequency to differentiate spam from ham emails. The analysis shows that spam emails are generally longer and contain more digits and special characters like $, %, and !. Additionally, words such as “email”, “will”, and common HTML tags like

, , and appear more frequently in spam emails. These patterns suggest that both structural elements and content-specific keywords are strong indicators of spam and should be incorporated into the model.

# 1 Question 4

Describe the process of improving your model. You should use at least 2-3 sentences each to address the following questions:

1. How did you find better features for your model?
2. What did you try that worked or didn't work?
3. What was surprising in your search for good features?

1. How did you find better features for your model? I used exploratory data analysis (EDA) to compare word frequencies between spam and non-spam emails. By calculating the frequency difference of each word normalized by class size, I identified which words appeared much more often in spam emails. I also included features like email length, digit count, and special character counts to capture structural patterns.
2. What did you try that worked or didn't work? One thing that worked well was selecting the top spammy words based on frequency difference — it significantly boosted accuracy. Adding length and digit/special character counts also helped. On the other hand, using raw text with a vectorizer didn't work because it wasn't allowed in the assignment and triggered errors.
3. What was surprising in your search for good features? I was surprised that HTML-related tokens like and font names like 'helvetica' were among the top spam indicators. It showed me that formatting patterns, not just content, can be strong spam signals. It also reminded me that seemingly unimportant metadata can be very informative.

## 2 Question 5: ROC Curve

In most cases, we won't be able to get 0 false positives and 0 false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover that they have cancer until it's too late. In contrast, a patient can receive another screening for a false positive.

Recall that logistic regression calculates the probability that an example belongs to a particular class. To classify an example, we say that an email is spam if our classifier gives it $\geq 0.5$ probability of being spam. However, **we can adjust that cutoff threshold**. We can say that an email is spam only if our classifier gives it $\geq 0.7$ probability of being spam, for example. This is how we can trade off false positives and false negatives.

The Receiver Operating Characteristic (ROC) curve shows this trade-off for each possible cutoff probability. In the cell below, plot an ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Lecture 23 may be helpful.

**Hint**: You'll want to use the `.predict_proba` method (documentation) for your classifier instead of `.predict` to get probabilities instead of binary predictions.

In [22]:
```
from sklearn.metrics import roc_curve, RocCurveDisplay

# Predict probabilities for the positive class (spam)
y_train_probs = grid.predict_proba(X_train_scaled)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_train_probs)

# Plot
RocCurveDisplay(fpr=fpr, tpr=tpr).plot()

# Add random guess diagonal
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.legend(loc='lower right')  # Optional: adds a legend
plt.title("ROC Curve for Logistic Regression")  # Optional: adds title
plt.grid(True)  # Optional: adds grid
plt.show()
```


ROC Curve for Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Random Guess

### 2.0.1 Question 6a

Pick at least **one** of the emails provided above to comment on. How would you classify the email (e.g., spam or ham), and does this align with the classification provided in the training data? What could be a reason someone would disagree with *your* classification of the email? In 2-3 sentences, explain your perspective and potential reasons for disagreement.

Example 1 (Index 5216): This email was labeled as spam (1) but reads like a personal message between friends or family. It includes casual language, storytelling, and no obvious spam indicators (no suspicious links, sales, or promotional tone). This suggests it may have been misclassified—possibly due to the sender's domain or past spam signals. It shows how false positives can arise when legitimate messages have formatting or metadata resembling spam.

### 2.0.2 Question 6b

As data scientists, we sometimes take the data to be a fixed "ground truth," establishing the "correct" classification of emails. However, as you might have seen above, some emails can be ambiguous; people may disagree about whether an email is actually spam or ham. How does the ambiguity in our labeled data (spam or ham) affect our understanding of the model's predictions and the way we measure/evaluate our model's performance?

Sometimes, emails aren't clearly spam or ham — they can be in a gray area. This means the labels in our dataset might not always be 100% correct. If an email is labeled as spam but feels more like ham (or vice versa), then our model might get "punished" for making a prediction that actually makes sense.

This makes it harder to fully trust metrics like accuracy, because the model might be making reasonable decisions based on confusing or mislabeled data. That's why, as data scientists, we should always think about where the data and labels came from, and remember that our model's performance depends on the quality of that data.

**Part ii** Please provide below the index of the email that you flipped classes (`email_idx`). Additionally, in 2-3 sentences, explain why you think the feature you chose to remove changed how your email was classified.

I chose to remove the word "bank" from the model's features. This word strongly signals spam because it's commonly used in scam or phishing emails. Once removed, the model no longer detected one of its key spam indicators, which caused the prediction to flip from spam to ham.

**Part i** In this context, do you think you could easily find a feature that could change an email's classification as you did in part a)? Why or why not?

I don't think it would be easy to find a single feature that changes the classification in a model with 1000 features. That's because in a bigger model, the prediction is usually based on a combination of many small signals rather than just one strong feature. In the smaller model, each feature has a more obvious influence, so removing one word can flip the prediction. But in the larger model, it's harder to trace the impact of any one feature.

**Part ii** Would you expect this new model to be more or less interpretable than `simple_model`?

**Note**: A model is considered interpretable if you can easily understand the reasoning behind its predictions and classifications. For example, the model we saw in part a), `simple_model`, is considered interpretable as we can identify which features contribute to an email's classification.

I would expect the larger model to be less interpretable than simple_model. With just a few features, it's easier to see how the model is making decisions, which makes it more transparent and easier to explain. When we add hundreds of features, it gets more complicated and harder to understand what's driving each prediction, even though the accuracy might improve.

### 2.0.3 Question 7c

Now, imagine you're a data scientist at Meta, developing a text classification model to decide whether to remove certain posts / comments on Facebook. In particular, you're primarily working on moderating the following categories of content: * Hate speech * Misinformation * Violence and incitement

Pick one of these types of content to focus on (or if you have another type you'd like to focus on, feel free to comment on that!). What content would fall under the category you've chosen? Refer to Facebook's Community Standards, which outline what is and isn't allowed on Facebook.

I chose to focus on violence and incitement. This category includes content that promotes or threatens physical harm, like posts encouraging riots, attacks on specific groups, or promoting armed conflict. According to Meta's Community Standards, this also includes threats that are credible, calls for organized violence, or praise of violent events. For example, a post saying "Let's bring our weapons and storm the courthouse tomorrow" would fall under this category.

#### 2.0.4 Question 7d

What are the stakes of misclassifying a post in the context of a social media platform? Comment on what a false positive and false negative means for the category of content you've chosen (hate speech, misinformation, or violence and incitement).

Misclassifying posts in this area can have serious consequences. A false positive—removing a post that isn't actually violent—could unfairly silence users or erase discussions about real events (e.g., reporting violence or expressing outrage about it). But a false negative—failing to detect actual incitement—can lead to real-world harm if it inspires attacks or puts people at risk. That's why it's so important that the model is precise and context-aware when flagging violent content.

### 2.0.5 Question 7e

As a data scientist, why might having an interpretable model be useful when moderating content online?

Having an interpretable model is really helpful when moderating content online because it allows us to understand why the model made a certain decision. This is especially important when we're dealing with sensitive categories like violence or hate speech. If a user appeals a content removal, we need to be able to explain what triggered the classification. It also helps build trust with users and allows policy teams to improve or adjust rules more easily. Without interpretability, it's like the model is making decisions in a black box, which can be risky and harder to improve.