Introduction

This notebook aims to create machine learning models that accurately classify the sentiment of tweets related to different products, and is intended for data scientists and business professionals interested in quickly and autonomously analyzing customer feedback on social media.

The notebook is organized into several sections covering data collection, preprocessing, exploratory data analysis, feature extraction, machine learning models, model evaluation, and business implications.

The business problem that this notebook addresses is helping companies understand the sentiment of their customers towards their products or services by analyzing social media data. By analyzing customer feedback on platforms such as Twitter, companies can gain insights into customer opinions, identify areas for improvement, and make data-driven decisions to enhance customer satisfaction and loyalty.

In [1]:
```
# Import necessary libraries for the project
import matplotlib.pyplot as plt # Visualizations
import tensorflow as tf # Neural Networks
import seaborn as sns # Visualizations
import pandas as pd # Data Manipulation
import numpy as np # Linear Algebra
import re # Text Processing

# Set random seeds to increase reproducibility of results
np.random.seed(42)
tf.random.set_seed(42)

# Sets verbose arguments of all models to that output is hidden in final notebook
global_verbose = 0

# Sets the visualization style to ggplot for consistency and aesthetics
plt.style.use("ggplot")
```

```
2023-03-28 15:37:10.067864: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized
with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical opera
tions:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Data Processing

Loading in data

Here, df2 is a signifficantly larger dataset that we will extract some relevant information from to hopefully bolster performance. I will try to filter for tweets which are associated with products.

In [2]:
```
# Load the product_tweet_prediction dataset as a Pandas dataframe
raw_df = pd.read_csv("Data/product_tweet_prediction.csv", encoding="ISO-8859-1")

# Load the train dataset as a Pandas dataframe, drop any rows with missing values
raw_df2 = pd.read_csv("Data/train.csv").dropna()
```

In [3]:
```
# Define a regular expression pattern to match product and service names
prod_servs = r"sxsw|iphone|ipad|apple|google|android|ipod|service|xbox|twitter|playstation|spotify|itunes|amazon|starbu

# Use regular expression matching to determine whether a tweet contains a product or service name
prods = raw_df2.text.str.contains(r"(?:" + prod_servs + r")\b", flags=re.IGNORECASE)

# Filter the tweets that contain product or service names and select the "text" and "sentiment" columns
extra = raw_df2[prods][["text", "sentiment"]].reset_index(drop=True)

# Rename the "text" column to "tweet_text" for clarity
extra.rename(columns={"text":"tweet_text"}, inplace=True)

# Map the "sentiment" column values to 0, 1, or 2 based on negative, positive, or neutral sentiment respectively
extra.sentiment = extra.sentiment.map({"negative":0, "positive":1, "neutral":2})

# Store the resulting dataframe in the "extra" variable
extra
```

Out[3]:

	tweet_text	sentiment
0	the free fillin` app on my ipod is fun, im add...	1
1	I'm going home now. Have you seen my new twitt...	1
2	:visiting my friendster and facebook	2
3	Then you should check out http://twittersucks...	2
4	mannnn..... _ got an iphone!!! im jealous.... ...	0
...	...	...
1052	that`s cuz you`re cruising the twitter #night...	2
1053	Twitter is slow!	0
1054	This is a much better tool than some I have co...	1
1055	my cousins moved there like 2 years ago and ...	0
1056	Scream just played on my iPod. First thing tha...	2

1057 rows × 2 columns

In [4]:
```
# Rename the "is_there_an_emotion_directed_at_a_brand_or_product" column to "sentiment" for clarity
raw_df.rename(columns={"is_there_an_emotion_directed_at_a_brand_or_product" : "sentiment"}, inplace=True)
```

In [5]:
```
# Drop all rows where the sentiment column contains the value "I can't tell"
raw_df = raw_df[raw_df['sentiment'] != "I can't tell"]

# Factorize the "sentiment" column to convert categorical values to numerical values
raw_df.sentiment = raw_df.sentiment.factorize()[0]

# Select only the "tweet_text" and "sentiment" columns from the dataframe
raw_df = raw_df[["tweet_text", "sentiment"]]

# Store the resulting dataframe in the "df" variable
raw_df
```

Out[5]:

	tweet_text	sentiment
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	0
1	@jessedee Know about @fludapp ? Awesome iPad/i...	1
2	@swonderlin Can not wait for #iPad 2 also. The...	1
3	@sxsw I hope this year's festival isn't as cra...	0
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	1
...	...	...
9088	Ipad everywhere. #SXSW {link}	1
9089	Wave, buzz... RT @mention We interrupt your re...	2
9090	Google's Zeiger, a physician never reported po...	2
9091	Some Verizon iPhone customers complained their...	2
9092	ï¿½Ïï¿½ï¿½ï¿½ï¿½ï¿½_ï¿½ï¿½Ê ï¿½Î ï¿½Ò ï¿½ï¿½ï...	2

8937 rows × 2 columns

In [6]:
```
# Concatenate the "df" and "extra" dataframes vertically using pd.concat()
raw_df = pd.concat([raw_df, extra], axis=0)

# Count the number of tweets in each sentiment category and calculate their proportions using the normalize parameter
raw_df.sentiment.value_counts(True)
```

Out[6]:
```
2    0.587452
1    0.330398
0    0.082149
Name: sentiment, dtype: float64
```

Class Imbalance

There is a massive class imbalance that will prove challenging when training models. I will undersample the majority class (neutral, 2) to address this initially.

In [7]:
```
neuts = raw_df[raw_df.sentiment == 2]

neuts = neuts.iloc[: int(len(neuts) / 2.5)]
raw_df = raw_df.drop(neuts.index).reset_index(drop=True)
print(raw_df.sentiment.value_counts(True))
print(len(raw_df))
```

```
2    0.462216
1    0.441903
0    0.095881
Name: sentiment, dtype: float64
7040
```

Text Preproccessing

In [8]:
```
# Import necessary libraries for text preprocessing
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize


# Create stopwords list to remove and remove several words from that list so that they are included
stops = stopwords.words("english")
for word in ["not", "no", "why", "should"]:
    stops.remove(word)
stops.append("amp")


# Define a list of regular expression patterns to replace with empty string
re_subs = [
    (r"\d+", ""), # remove digits
    (r"\*{2,}", " cuss "), # replace repeated * with cuss word
    (r"https?://\S+|www\.\S+", ""), # remove urls
    (r"[^\w\s]", " "), # remove punctuation
    (r"[^\x00-\x7F]+", ""), # Remove non-ASCII characters
    (prod_servs, "product"), # replace speciffic occurences of products with the word priduct to help model generalize
    (r"\b(mention|link|rt|quot)\b", ""), # remove words with very high frequencies in all classes
    (r"\b(" + "|".join(stops) + r")\b\s*", ""), # remove stopwords
    (r"\b\w\b", ""), # remove single letter words
    (r"(\bproduct+\b)(\s+\1)+", r"\1") # Replace repititions of product
    ]

lemr = WordNetLemmatizer()

# Define a function to preprocess a given input string
def preprocess_text(input_string):
    # Convert input string to lowercase
    input_string = str(input_string).lower()

    # Replace regular expression patterns in input string with empty string
    for pattern, repl in re_subs:
        input_string = re.sub(pattern, repl, input_string)

    # input_string = re.sub(r'(\bproduct+\b)(\s+\1)+', r'\1', input_string)

    # Tokenize the input string into words
    tokens = word_tokenize(input_string)

    # Lemmatize each word in the input string, including verbs
    words = [lemr.lemmatize(word, pos="v") for word in tokens]

    # Join the words back together into a single string and return it
    return " ".join(words)
```

In [9]:
```
df = raw_df.copy()

df.tweet_text = df.tweet_text.apply(preprocess_text) # Apply the preprocessing function defined above
df
```

Out[9]:

	tweet_text	sentiment
0	wesley product hrs tweet rise_austin dead need...	0
1	jessedee know fludapp awesome product app like...	1
2	swonderlin not wait product also should sale p...	1
3	product hope year festival crashy year product...	0
4	sxtxstate great stuff fri product marissa maye...	1
...	...	...
7035	ha look product game harmless fun despite auth...	1
7036	someone horse btw hubb feel product name party	2
7037	cuz cruise product nightshift	2
7038	much better tool come across product karma ste...	1
7039	scream play product first thing come mind bear...	2

7040 rows × 2 columns

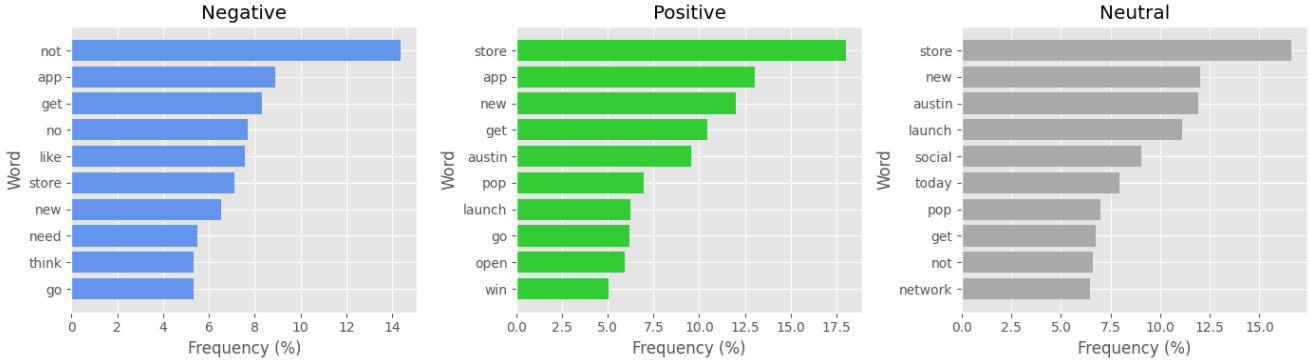
Data Visualizations

In [10]:
```
# Show process of preparing raw tweets for model
from keras.preprocessing.text import Tokenizer

# Raw tweet example
tweet = "I love using my new iPad! Best #christmas gift ever!"

# Apply preprocessing steps
processed_tweet = preprocess_text(tweet)

# Tokenize and vectorize
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(df.tweet_text)
sequenced_tweet = tokenizer.texts_to_sequences([processed_tweet])

# Print the steps
print(tweet)
print(processed_tweet)
print(sequenced_tweet)
```

```
I love using my new iPad! Best #christmas gift ever!
love use new product best christmas gift ever
[[46, 25, 3, 1, 67, 1138, 789, 125]]
```

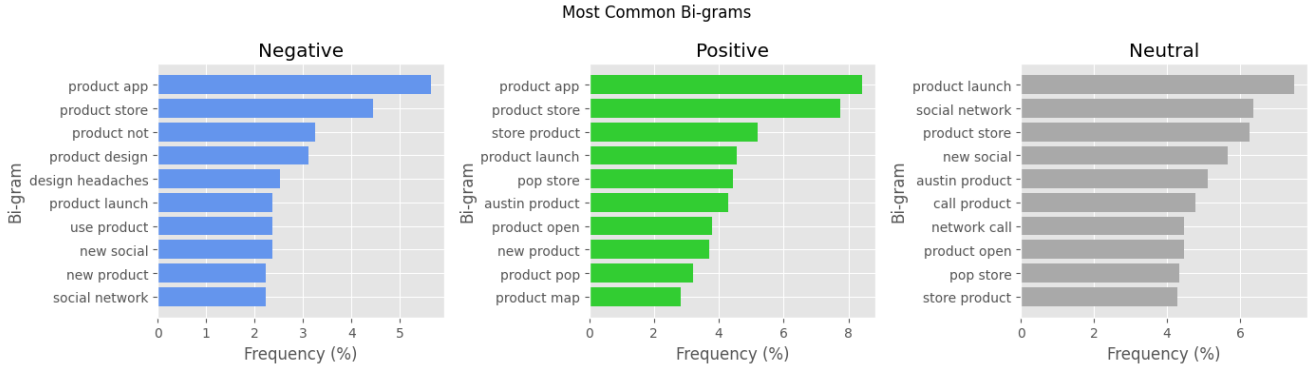
In [11]:
```
# Create the figure and three axes for each class
fig, axes = plt.subplots(1, 3, figsize=(14, 4), tight_layout=True)
# fig.suptitle("Most Common Words")

# Create arrays of sentiment names and colors to represent them
senitments = ["Negative", "Positive", "Neutral"]
colors = ["cornflowerblue", "limegreen", "darkgrey"]

# Loop over the sentiment categories 0, 1, and 2
for sentiment in range(3):
    # Select the tweet_text for the current sentiment category
    texts = df.tweet_text[df.sentiment == sentiment]

    # Split the text into words and count the frequency of each word
    word_frequency = texts.str.split(expand=True).stack().value_counts().reset_index().head(11)[1:]
    print(word_frequency.iloc[5])
    # Rename the columns of the word_frequency dataframe
    word_frequency.columns = ["Word", "Frequency"]

    # Normalize the word frequency
    word_frequency.Frequency /= len(texts) / 100

    # Create bar plot
    ax = axes[sentiment]

    ax.barh(word_frequency["Word"], word_frequency["Frequency"], color=colors[sentiment])
    ax.set_ylabel('Word')
    ax.set_xlabel('Frequency (%)')
    ax.set_title(senitments[sentiment])
    ax.invert_yaxis()
```

```
index    store
0           48
Name: 6, dtype: object
index    pop
0        217
Name: 6, dtype: object
index    today
0          259
Name: 6, dtype: object
```

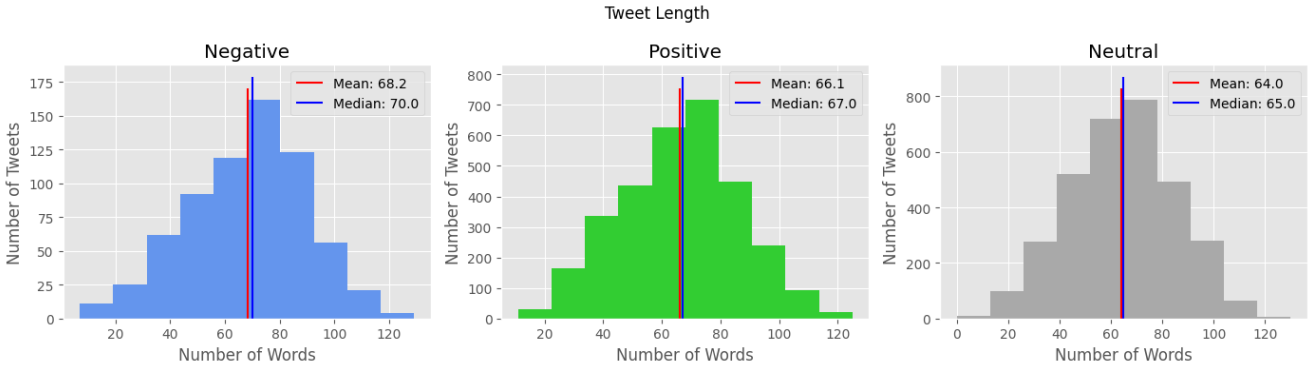


In [12]:
```
from sklearn.feature_extraction.text import CountVectorizer

def get_top_ngrams(data, n, top_n):
    # Create a count vectorizer object
    vectorizer = CountVectorizer(ngram_range=(n,n))

    # Fit and transform the data
    X = vectorizer.fit_transform(data)

    # Get the feature names (n-grams)
    feature_names = vectorizer.get_feature_names_out()

    # Get the frequency count of each n-gram
    count_values = X.toarray().sum(axis=0)

    # Create a dictionary with the n-grams and their frequency counts
    ngram_freq = pd.DataFrame(zip(feature_names, count_values), columns=[f"{n}-gram", "frequency"])

    return ngram_freq.sort_values(by="frequency", ascending=False).head(top_n)
```

In [13]:
```
# Create a figure with three subplots, with a title.
fig, axes = plt.subplots(1, 3, figsize=(14, 4), tight_layout=True)
fig.suptitle("Most Common Bi-grams")

n = 2

# Loop through the three different sentiments.
for sentiment in range(3):
    # Select the tweet texts with the current sentiment.
    sentences = df.tweet_text[df.sentiment == sentiment]

    # Get the top 10 bi-grams and their frequencies, and normalize the frequencies to a scale of 0 to 100.
    freqs = get_top_ngrams(sentences, n, 10)
    freqs.frequency /= len(sentences) / 100

    # Select the corresponding subplot and create a horizontal bar chart of the bi-grams and their frequencies.
    ax = axes[sentiment]
    ax.barh(freqs[f"{n}-gram"], freqs["frequency"], color=colors[sentiment])

    # Set the y-label, x-label, and title for the current subplot.
    ax.set_ylabel('Bi-gram')
    ax.set_xlabel('Frequency (%)')
    ax.set_title(senitments[sentiment])
    ax.invert_yaxis()
```



In [14]:
```
fig, axes = plt.subplots(1, 3, figsize=(14, 4), tight_layout=True)
fig.suptitle("Tweet Length")

# Add a new column to the dataframe "df" that contains the length of each tweet
df['length'] = df['tweet_text'].apply(len)

# Create a histogram of tweet lengths, grouped by sentiment, with a specified figure size

for sentiment in range(3):
    lengths = df.length[df.sentiment == sentiment]

    ax = axes[sentiment]

    ax.hist(lengths, color=colors[sentiment])
    ax.set_ylabel('Number of Tweets')
    ax.set_xlabel('Number of Words')
    ax.vlines(lengths.mean(), 0, ax.get_ylim()[1], colors="red", label=f"Mean: {lengths.mean():.1f}")
    ax.vlines(lengths.median(), 0, ax.get_ylim()[1], colors="blue", label=f"Median: {lengths.median()}")
    ax.legend()
    ax.set_title(senitments[sentiment])
```



Binary Models

In [15]:
```
# Import the `train_test_split` function from the scikit-learn library.
from sklearn.model_selection import train_test_split

# Filter the DataFrame to only include positive and negative sentiments (binary classification).
binary_df = df[df.sentiment < 2]

# Split the filtered DataFrame into feature (X) and target (y) variables.
X = binary_df.tweet_text
y = binary_df.sentiment

# Split the feature and target variables into training, validation, and test sets, with a 80-10-10 split ratio.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

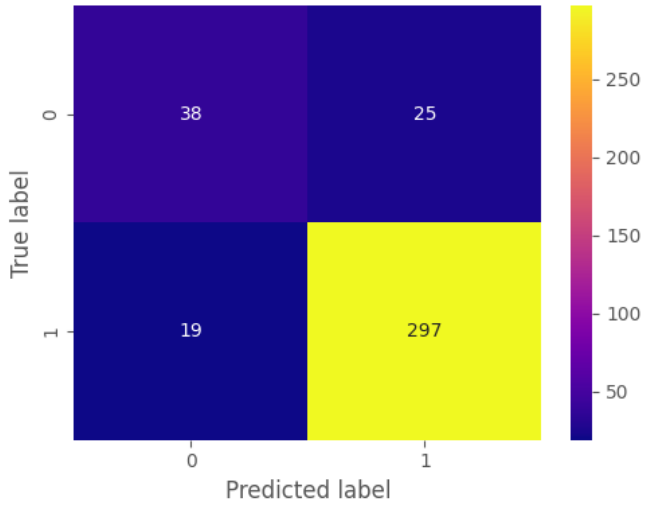
Metric evaluation function

In [16]:
```
# Import necessary libraries for model evaluation
from sklearn.metrics import classification_report, confusion_matrix

# Define a function to print classification metrics and confusion matrix
def metrics(y_true, y_pred):
    fig, axes = plt.subplots(ncols=2, figsize=(10,4))

    # Display the classification report, which includes precision, recall, f1-score, and support
    metrics = classification_report(y_true, y_pred)
    ax=axes[1]
    ax.axis("off")
    ax.text(0.5, 0.25, metrics, size=12, ha="center", transform=ax.transAxes)

    ax = axes[0]
    # Create the confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Create confusion matrix plot, which shows the counts of true positives, false positives, true negatives, and fal
    sns.heatmap(cm, annot=True, fmt="d", cmap="plasma", ax=ax)
    ax.set_ylabel("True label")
    ax.set_xlabel("Predicted label")

    fig.tight_layout()
```

Logistic Regression

In [17]:
```
# Import necessary libraries for building a machine learning model
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# Define a pipeline for the machine learning model using TfidfVectorizer and LogisticRegression
lr_pipe = Pipeline([
    ("vec", TfidfVectorizer(max_features=500000, ngram_range=(1,5))),
    ("clf", LogisticRegression(class_weight="balanced", random_state=42))
])

# # Define a grid of hyperparameters to search over using GridSearchCV
# param_grid = {
#     "vec__ngram_range" : [(1,2)],
#     "clf__C" : [2, 3, 4],
#     "clf__class_weight" : ["balanced", None]
# }

# # Perform a grid search over the hyperparameter grid using GridSearchCV
# grid_search = GridSearchCV(pipe, param_grid, cv=5, n_jobs=-1, scoring="f1_macro")
# grid_search.fit(X_train, y_train)

# # Print the best hyperparameters found by GridSearchCV
# grid_search.best_params_
```

In [18]:
```
best_params = {
    'clf__C': 10,
    'clf__class_weight': 'balanced',
    'vec__ngram_range': (1, 2)
    }

# Set the best hyperparameters found by GridSearchCV on the pipeline
lr_pipe.set_params(**best_params)

# Fit the pipeline on the training data
lr_pipe.fit(X_train, y_train)

lr_preds = lr_pipe.predict(X_test)

# Print the classification metrics and confusion matrix for the validation and test sets
metrics(y_test, lr_preds)
```



	precision	recall	f1-score	support
0	0.67	0.60	0.63	63
1	0.92	0.94	0.93	316
accuracy			0.88	379
macro avg	0.79	0.77	0.78	379
weighted avg	0.88	0.88	0.88	379

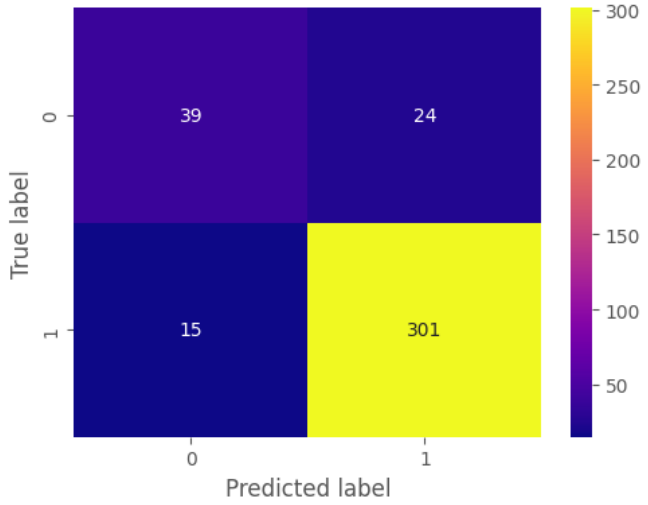
Evaluation

This model performed very well. 88% accuracy while successfully predicting a majority of the minority class resulting in a solid f1 score of ~0.80.

SVM - SVC

In [19]:
```
# Import necessary libraries for building a support vector machine model
from sklearn.svm import SVC

# Define a pipeline for the support vector machine model using TfidfVectorizer and SVC
svm_pipe = Pipeline([
    ("vec", TfidfVectorizer(max_features=int(5e6))),
    ("clf", SVC(kernel="linear", random_state=42, probability=True))
])

# Define a grid of hyperparameters to search over using GridSearchCV
# param_grid = {
#     "vec__ngram_range" : [(1,2), (1,3)],
#     "clf__kernel" : ["linear", "rbf"],
#     "clf__gamma" : ["scale", "auto"],
#     "clf__class_weight" : ["balanced", None]
# }

# # Perform a grid search over the hyperparameter grid using GridSearchCV
# gsearch = GridSearchCV(svm_pipe, param_grid, cv=5, n_jobs=-1, scoring="f1_macro")
# gsearch.fit(X_train, y_train)

# # Print the best hyperparameters found by GridSearchCV
# gsearch.best_params_
```

In [20]:
```
best_params = {
    'clf__class_weight': 'balanced',
    'clf__gamma': 'scale',
    'clf__kernel': 'linear',
    'vec__ngram_range': (1, 2)
    }

# Set best params and evaluate performance as with LR
svm_pipe.set_params(**best_params)
svm_pipe.fit(X_train, y_train)

svm_preds = svm_pipe.predict(X_test)

metrics(y_test, svm_preds)
```



	precision	recall	f1-score	support
0	0.72	0.62	0.67	63
1	0.93	0.95	0.94	316
accuracy			0.90	379
macro avg	0.82	0.79	0.80	379
weighted avg	0.89	0.90	0.89	379

Evaluation

This model performed almost identically to the Logistic Regression model. It would be interesting to see if the tweets they are misclassifying are the same, or they are having trouble with separate issues.

In [21]:
```
test = pd.DataFrame({
    "text" : X_test,
    "sentiment" : y_test,
    "lr_preds" : lr_preds,
    "svm_preds" : svm_preds
})
test["original_text"] = raw_df.loc[test.index, "tweet_text"]

incorrect = test[(test.sentiment != test.lr_preds) | (test.sentiment != test.svm_preds)]

pd.set_option('display.max_colwidth', None)

incorrect.head()
```

Out[21]:

	text	sentiment	lr_preds	svm_preds	original_text
1705	product bing panel like world expensive seo consultation product	1	0	0	This Google/Bing Q&A panel is like the world's most expensive SEO consultation. #SXSW
6604	open product account little confuse really get product seem much better	1	0	0	Just opened a facebook account, I`m a little confused I don`t really get it. Twitter seems much better
4058	best thing hear weekend product give product money japan relief need product	1	0	0	RT @mention RT @mention Best thing I've heard this weekend at #SXSW "I gave my iPad 2 money to #Japan relief. I don't need an iPad
4395	vs vs product things heat product offer check reward title lbs	1	0	1	So @mention vs @mention vs @mention at #SXSW. Things are heating up! "Google offers check-in rewards & titles" {link} - #lbs
229	need find better product stream follow inane product tweet surely not southby	0	1	1	I need to find a better #SXSW stream to follow. The inane iPad 2 tweets are surely not what SouthBy is about

The model's training performance improves with each epoch until epoch 5, where the validation loss increases while the accuracy and precision metrics do not improve. This suggests that the model may have started to overfit the training data. We can also see that the model almost entirely predicted the majority class.

Next steps include adjusting the model's hyperparameters and changing the architecture of the model to prevent overfitting.

Neural Networks

Class Weight Calculation

A weighted class distribution is created in order to combat the massive class imbalance. Otherwise, the models will likely try to simply predict the majority class.

In [22]:
```
# Import necessary libraries for computing class weights
from sklearn.utils import compute_class_weight

# Compute class weights for the training data
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(y), y=y)

# Create a dictionary mapping class indices to class weights
class_weight_dict = dict(enumerate(class_weights))

# Print the resulting class weight dictionary
class_weight_dict
```

Out[22]:
```
{0: 2.8044444444444445, 1: 0.6084860173577628}
```

Generate Keras Sequences

Creating a Tokenizer object and fitting it on the training data (X_train) using the fit_on_texts() method will generate a word index that can be used to convert text to sequences of token weDs.

Converting to sequences of token weDs using the texts_to_sequences() method converts each word in the text to its corresponding token weD using the word index generated during training.

Padding the sequences with zeros will ensure they all have the same length (maxlen) using the pad_sequences() function. This is necessary because the model expects inputs of a fixed length, and so sequences that are shorter than maxlen are padded with zeros at the end, while sequences that are longer than maxlen are truncated.

In [23]:
```
# Import necessary libraries for text preprocessing
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define the maximum length of the input sequences and the number of words to keep
maxlen = 130
num_words = 10000

# Create a tokenizer and fit it on the training data
tokenizer = Tokenizer(num_words=num_words)
tokenizer.fit_on_texts(X_train)

# Convert the training, validation, and test data to sequences of token IDs using the tokenizer
train_sequences = tokenizer.texts_to_sequences(X_train)
val_sequences = tokenizer.texts_to_sequences(X_val)
test_sequences = tokenizer.texts_to_sequences(X_test)

# Pad the sequences with zeros to ensure they all have the same length
train_data = pad_sequences(train_sequences, maxlen=maxlen)
val_data = pad_sequences(val_sequences, maxlen=maxlen)
test_data = pad_sequences(test_sequences, maxlen=maxlen)
```

Evaluation Function

In [24]:
```
def analyze_model(_history, _model, _test_data=test_data, y_true=y_test):
    fig, axes = plt.subplots(2, 2, figsize=(10,8))

    # Generate predictions on the test data
    y_pred = _model.predict(_test_data)

    if y_pred.shape[1] == 1:
        y_pred = list((y_pred > 0.5).astype(int))
    else:
        y_pred = np.argmax(y_pred, axis=1)
        y_true = np.argmax(y_true, axis=1)

    # Print model metrics report
    metrics = classification_report(y_true, y_pred)
    ax=axes[1][1]
    ax.axis("off")
    ax.text(0.5, 0.25, metrics, size=12, ha="center", transform=ax.transAxes)

    # Generate a confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Plot the confusion matrix
    ax=axes[1][0]
    sns.heatmap(cm, annot=True, fmt="d", cmap="plasma", ax=ax)
    ax.set_ylabel("True label")
    ax.set_xlabel("Predicted label")

    # Plot the training and validation accuracy
    ax = axes[0][0]

    ax.plot(_history.history["accuracy"])
    ax.plot(_history.history["val_accuracy"])
    ax.set_title("Model Accuracy")
    ax.set_ylabel("Accuracy")
    ax.set_xlabel("Epoch")
    ax.legend(["Train", "Val"])

    # Plot the training and validation loss
    ax = axes[0][1]
    ax.plot(_history.history["loss"])
    ax.plot(_history.history["val_loss"])
    ax.set_title("Model Loss")
    ax.set_ylabel("Loss")
    ax.set_xlabel("Epoch")
    ax.legend(["Train", "Val"])

    fig.tight_layout()
```

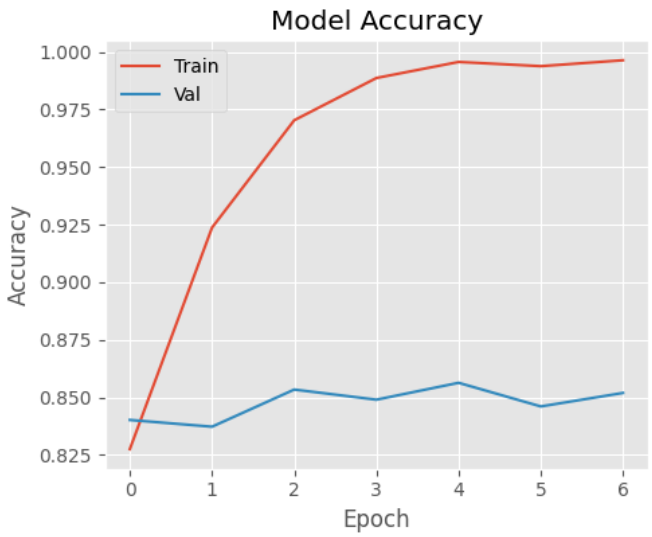
Base Model

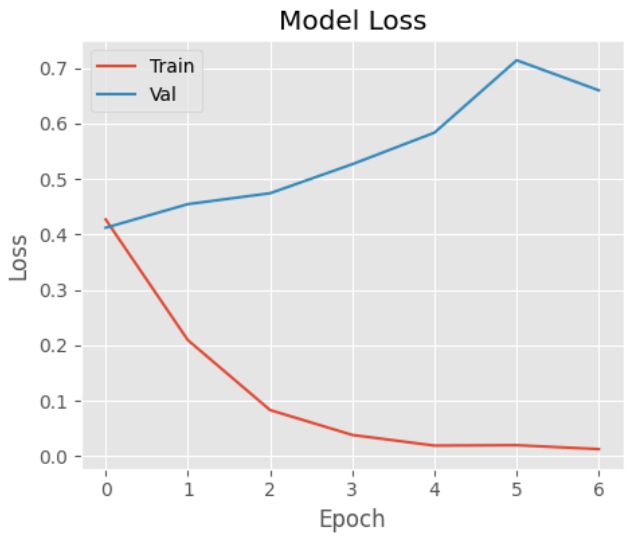
My base NN model will have a classic architecture of an embedding layer to capture semantic and syntactic relationships between words, followed by an LSTM layer to remember information from previous time steps. The embedding layer takes the input words and maps them to dense vectors of fixed size, while the LSTM layer processes the sequences of words and extracts meaningful features from them.

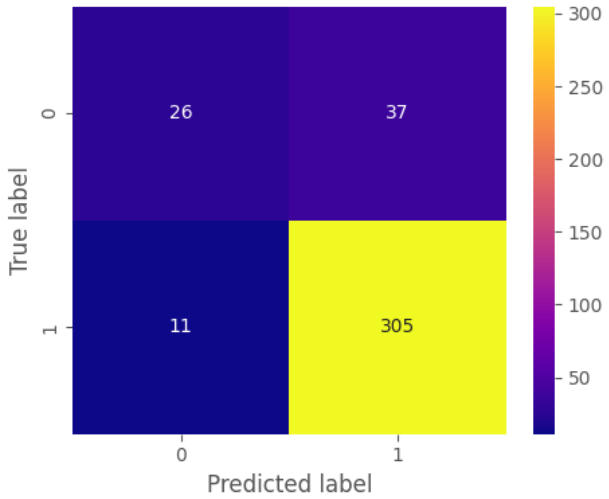
In [25]:
```
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Embedding, LSTM

# Create a Sequential model
model = Sequential()

# Add an Embedding layer that maps input sequences of token weDs to dense vectors of fixed size
model.add(Embedding(input_dim=num_words, output_dim=128))

# Add an LSTM layer with 64 units to process the sequences of embedded vectors
model.add(LSTM(64))

# Add a Dense output layer with a single unit and a sigmoid activation function to generate binary predictions
model.add(Dense(1, "sigmoid"))

# Compile the model using the Adam optimizer, binary cross-entropy loss, and track accuracy and precision as metrics
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy", "Precision"])

# Fit the model on the training data and validate on the validation data for 20 epochs with a batch size of 16
# Use EarlyStopping to prevent overfitting
history = model.fit(
    train_data,
    y_train,
    verbose=global_verbose,
    validation_data=(val_data, y_val),
    epochs=20,
    batch_size=16,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
    ],
)
```

In [26]:
```
analyze_model(history, model)
```

```
12/12 [==============================] - 1s 17ms/step
```







	precision	recall	f1-score	support
0	0.70	0.41	0.52	63
1	0.89	0.97	0.93	316
accuracy			0.87	379
macro avg	0.80	0.69	0.72	379
weighted avg	0.86	0.87	0.86	379

Evaluation

The model's training performance improves with each epoch until epoch 5, where the validation loss increases while the accuracy and precision metrics do not improve. This suggests that the model may have started to overfit the training data. We can also see that the model almost entirely predicted the majority class.

Next steps include adjusting the model's hyperparameters and changing the architecture of the model to prevent overfitting.

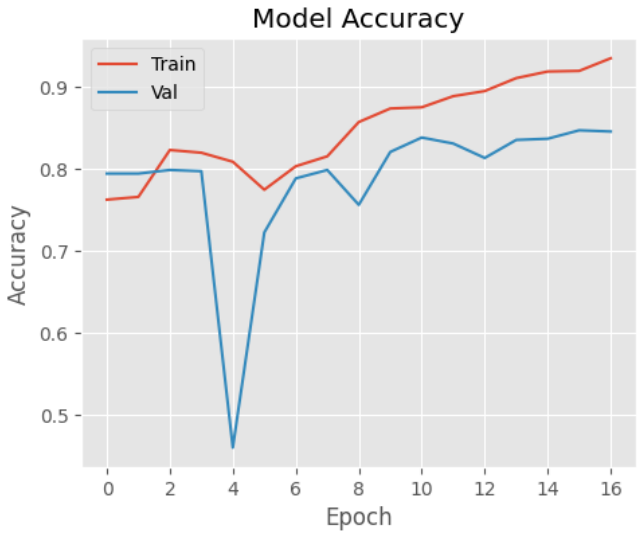
Add Regularization via Dropout

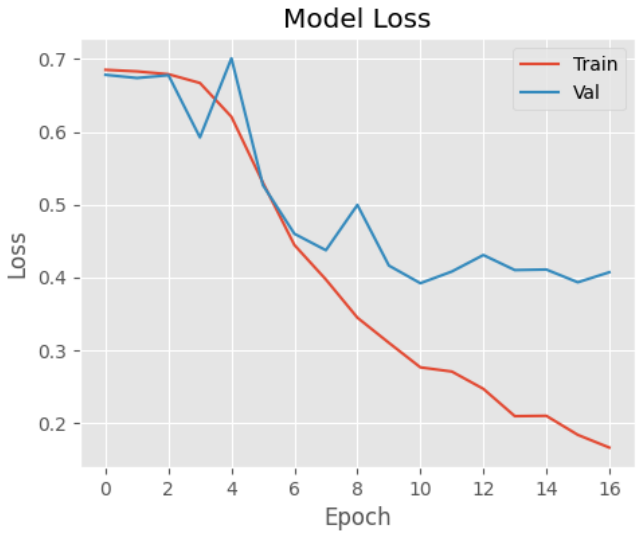
In the following model, we add a signifficant amount of dropout regularization and add an additional LSTM layer. These changes help the model to generalize better and prevent overfitting. Additionally, the model is now using the Adam optimizer with a lower learning rate of 1e-4 and includes a class weight dictionary to account for the class imbalance in the data.

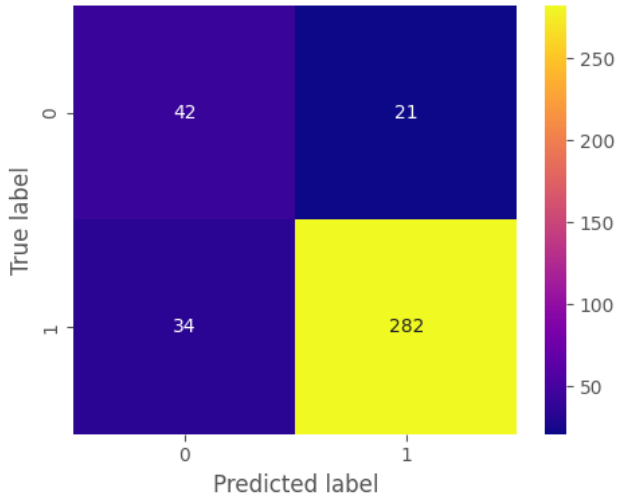
In [27]:
```
from tensorflow.keras.optimizers import Adam
from keras.layers import Dropout


model_2 = Sequential()
model_2.add(Embedding(input_dim=num_words, output_dim=128))

# Add a dropout layer to the model
model_2.add(Dropout(0.5))

# Add an LSTM layer with 64 units and 0.5 dropout, returning sequences so for the following LSTM layer
model_2.add(LSTM(64, dropout=0.5, return_sequences=True))

# Add another LSTM layer with 32 units and 0.3 dropout, not returning sequences
model_2.add(LSTM(32, dropout=0.5))

model_2.add(Dense(1, "sigmoid"))

model_2.compile(optimizer=Adam(learning_rate=1e-4), loss="binary_crossentropy", metrics=["accuracy", "Precision"])
history = model_2.fit(
    train_data, y_train,
    verbose=global_verbose,
    validation_data=(val_data, y_val),
    epochs=25,
    batch_size=16,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        ],
    class_weight=class_weight_dict
)
```

In [28]:
```
analyze_model(history, model_2)
```

```
12/12 [==============================] - 1s 46ms/step
```







	precision	recall	f1-score	support
0	0.55	0.67	0.60	63
1	0.93	0.89	0.91	316
accuracy			0.85	379
macro avg	0.74	0.78	0.76	379
weighted avg	0.87	0.85	0.86	379

Evaluation

Though the model's accuracy only modestly increased, the precision, recall, and f1 scores all improved greatly. This is because the model began to predict the minority class. The model begins to overfit at around the 8th epoch

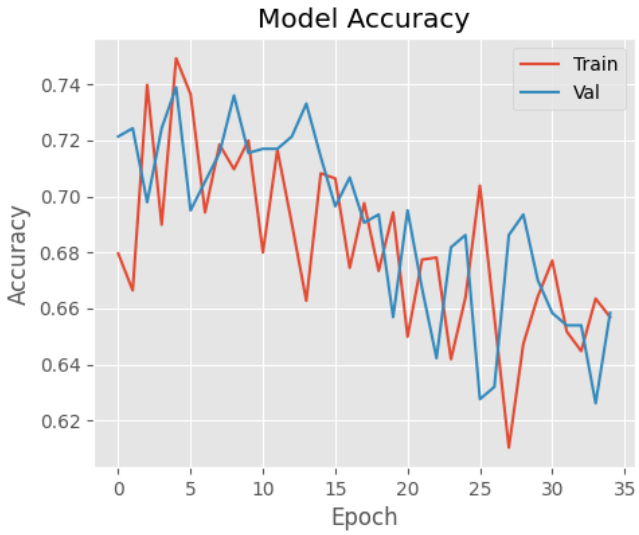
Glove Embedding

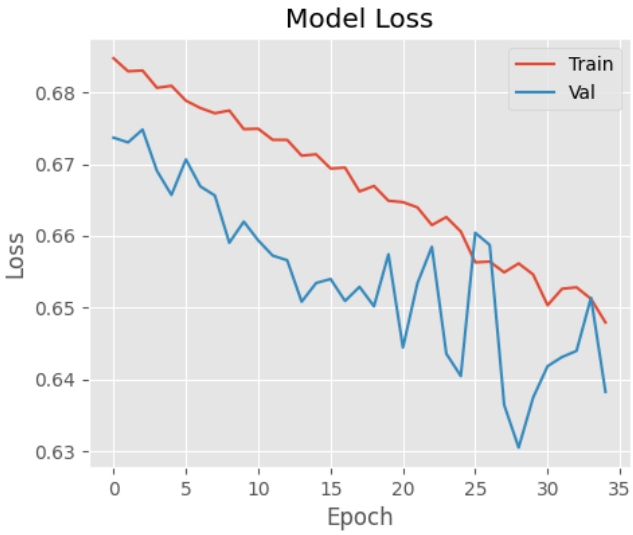
the purpose of using the GloVe embedding is to utilize pre-trained word embeddings that were trained on a large corpus of Twitter data. By doing so, we can leverage the semantic information captured in these embeddings to improve the performance of our model. wen this case, we used a pre-trained GloVe embedding with 100 dimensions, and add it to our model as the first layer. Additionally, we set the trainable parameter to False so that the embedding weights are not updated during training, and are kept fixed to the pre-trained values. The rest of the model is similar to the previous one, except for the use of a different embedding layer.

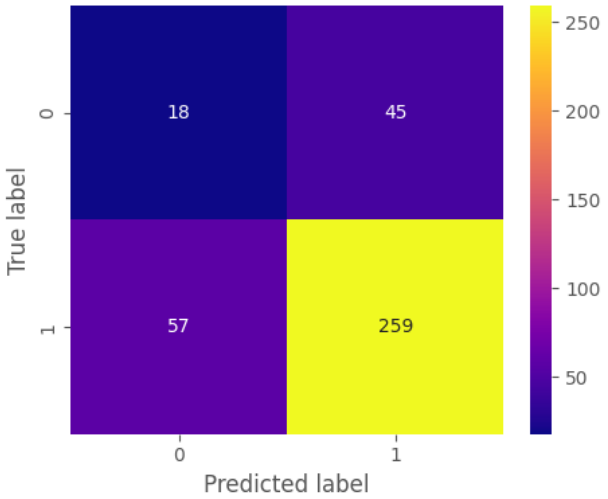
In [29]:
```
import gensim.downloader as api

embedding = api.load("glove-twitter-100") # Load GloVe Twitter 100 pre-trained word embeddings using Gensim APwe

model_glove = Sequential()
model_glove.add(Embedding(input_dim=len(embedding.index_to_key), output_dim=100, weights=[embedding.vectors], trainabl
model_glove.add(Dropout(0.3))
model_glove.add(LSTM(96, return_sequences=True))
model_glove.add(LSTM(64))
model_glove.add(Dense(1, "sigmoid"))

model_glove.compile(optimizer=Adam(learning_rate=1e-5), loss="binary_crossentropy", metrics=["accuracy", "Precision"])
history = model_glove.fit(
    train_data,
    y_train,
    verbose=global_verbose,
    validation_data=(val_data, y_val),
    epochs=40,
    batch_size=16,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        ],
    class_weight=class_weight_dict
)
```

In [30]:
```
analyze_model(history, model_glove)
```

```
12/12 [==============================] - 2s 44ms/step
```







	precision	recall	f1-score	support
0	0.24	0.29	0.26	63
1	0.85	0.82	0.84	316
accuracy			0.73	379
macro avg	0.55	0.55	0.55	379
weighted avg	0.75	0.73	0.74	379

Evaluation

This did not work at all. The model seems to predict the majority class in all circumstances, regardless of changes to architecture, optimizer, or regularization. This adds further evidence to my assumption that this dataset has a lot of noise.

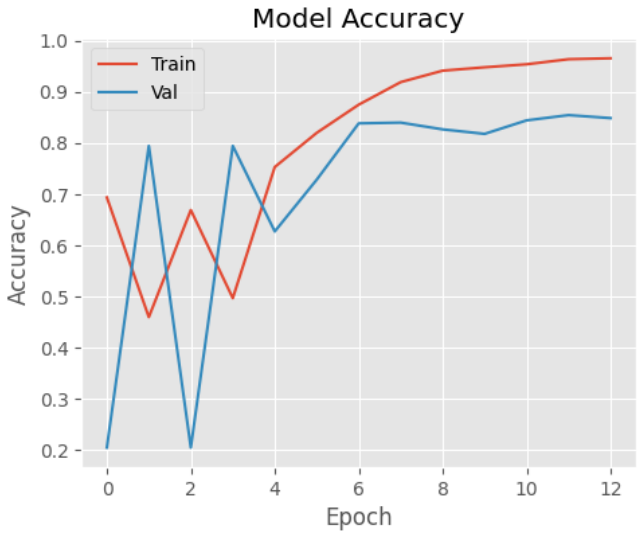
Complex Model From from Cristóbal Colón-Ruiz, wesabel Segura-Bedmar

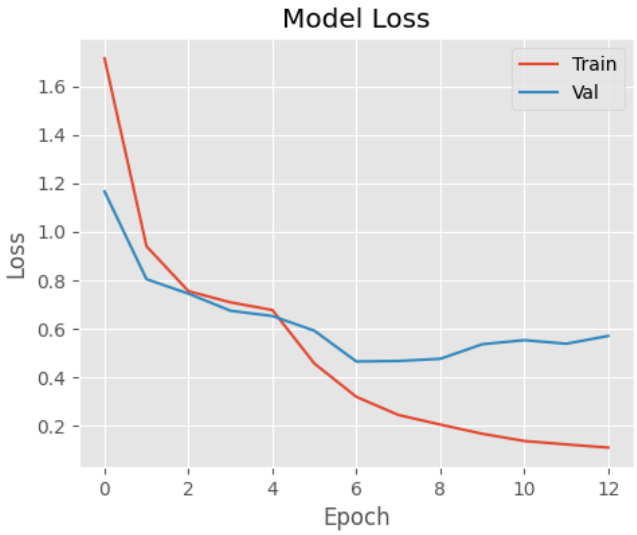
This architecture is based on an architecture from this paper (https://www.sciencedirect.com/science/article/pii/S1532046420301672) comparing deep learning architectures for sentiment analysis on drug reviews. This architecture is a scaled down version of one of their top models in the papaer.

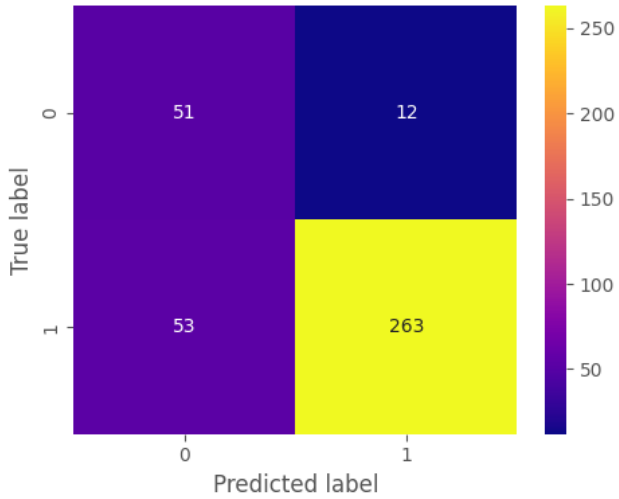
In [31]:
```
from keras.models import Model
from keras.layers import SpatialDropout1D, Conv1D, MaxPooling1D, concatenate, Input, Flatten, Bidirectional
from keras.regularizers import l2

# Model Layers
input_layer = Input(shape=(maxlen,))
embedding_layer = Embedding(num_words, 128, input_length=maxlen)(input_layer)

dropout_layer = SpatialDropout1D(0.4)(embedding_layer) # Overfitting mitigation

biltsm = Bidirectional(LSTM(96, return_sequences=True))(dropout_layer)

# Add Conv1D Layers with MaxPooling for different kernel sizes to capture n-grams
conv1 = MaxPooling1D()(Conv1D(filters=32, kernel_size=2, activation="relu")(biltsm)) # 2 grams
conv2 = MaxPooling1D()(Conv1D(filters=32, kernel_size=3, activation="relu")(biltsm)) # 3 grams
conv3 = MaxPooling1D()(Conv1D(filters=32, kernel_size=5, activation="relu")(biltsm)) # 5 grams
convs = concatenate([conv1, conv2, conv3], axis=1)

dropout_layer_2 = SpatialDropout1D(0.4)(convs) # Overfitting mitigation

dense = Dense(96, activation="relu", kernel_regularizer=l2())(Flatten()(dropout_layer_2))
output = Dense(1, activation="sigmoid")(dense)

# Create Model
model_complex = Model(inputs=input_layer, outputs=output)
model_complex.compile(optimizer=Adam(learning_rate=1e-4), loss="binary_crossentropy", metrics=["accuracy", "Precision"

# Train and Evaluate
history = model_complex.fit(
    train_data, y_train,
    verbose=global_verbose,
    epochs = 25,
    batch_size = 16,
    validation_data = (val_data, y_val),
    class_weight=class_weight_dict,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        ],
    )
```

In [32]:
```
analyze_model(history, model_complex)
```

```
12/12 [==============================] - 2s 59ms/step
```







	precision	recall	f1-score	support
0	0.49	0.81	0.61	63
1	0.96	0.83	0.89	316
accuracy			0.83	379
macro avg	0.72	0.82	0.75	379
weighted avg	0.88	0.83	0.84	379

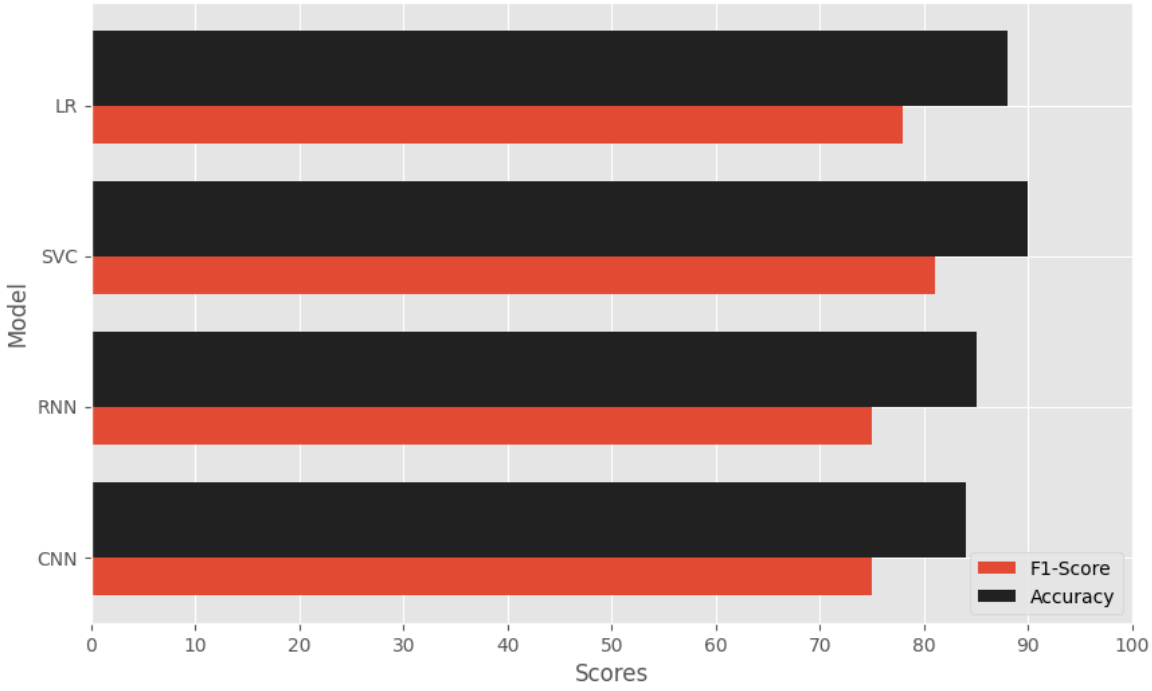
This model performed quite well! Despite setting the random seeds and states, we still get a degree of randomness when training these models. This model seems to be behind the second model in terms of performance, so we will use that model as the best model to extend to binary.

In [33]:
```
model_2.save("best_model.h5")
```

Binary Results Summary

In [34]:
```
fig, ax = plt.subplots(figsize=(10,6))

models = ["LR", "SVC", "RNN", "CNN"]
accuracies = [88,90,85,84]
f1s = [78,81,75,75]

for ls in [models, accuracies, f1s]:
    ls.reverse()


ax.barh(models, f1s, label="F1-Score", height=0.5)
ax.barh(models, accuracies, label="Accuracy", height=0.5, align="edge", color="#212121")
ax.legend()
ax.set_xlabel("Scores")
ax.set_ylabel("Model")
ticks = [10 * i for i in range(11)]
ax.set_xticks(ticks)
```

Out[34]:
```
[<matplotlib.axis.XTick at 0x16d46dfc0>,
 <matplotlib.axis.XTick at 0x16d46d330>,
 <matplotlib.axis.XTick at 0x177e077f0>,
 <matplotlib.axis.XTick at 0x16d35fc70>,
 <matplotlib.axis.XTick at 0x16d35dd80>,
 <matplotlib.axis.XTick at 0x16d35d0f0>,
 <matplotlib.axis.XTick at 0x16d51a5c0>,
 <matplotlib.axis.XTick at 0x16d51ad10>,
 <matplotlib.axis.XTick at 0x16d51acb0>,
 <matplotlib.axis.XTick at 0x16d51b370>,
 <matplotlib.axis.XTick at 0x16d51ae60>]
```



Multiclass Models

This code preprocesses the text data in `df.tweet_text` for use in a deep learning model by converting the text to sequences of token weDs and padding them to ensure they are all of the same length.

`to_categorical` is used to one-hot encode the labels df.sentiment to create y3.

The `maxlen` variable defines the maximum sequence length and the `num_words` variable defines the maximum number of words in the vocabulary.

A Tokenizer is created and fit on the training set to convert text data to sequences of token weDs. The `texts_to_sequences` method is then used to convert the text data in the train, validation, and test sets to sequences of token weDs.

Finally, the `pad_sequences` method is used to pad the sequences to ensure they are all of the same length. The resulting padded sequences are stored in `train_data3` , `val_data3` , and `test_data3` .

Again, we will start with simple Logisitc Regression and SVM models, using a grid search to find the optimal hyperparameters.

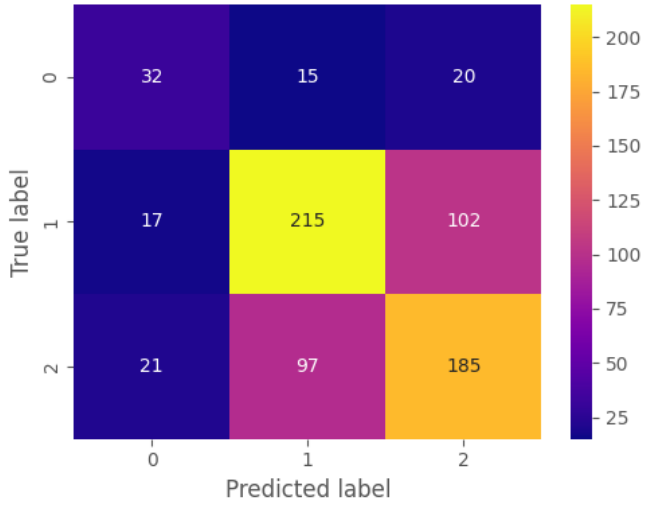
In [35]:
```
from keras.utils import to_categorical

# Extract features and labels from the dataframe
X3 = df.tweet_text
y3 = to_categorical(df.sentiment, num_classes=3)

# Split the dataset into train, validation, and test sets
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.1, random_state=42)
X_train3, X_val3, y_train3, y_val3 = train_test_split(X_train3, y_train3, test_size=0.2, random_state=42)

# Define the maximum sequence length and the maximum number of words in the vocabulary
maxlen = 130
num_words = 10000

# Create a tokenizer and fit it on the training set to convert text data to sequences of token weDs
tokenizer = Tokenizer(num_words=num_words)
tokenizer.fit_on_texts(X_train3)

# Convert the text data in the train, validation, and test sets to sequences of token weDs
train_sequences3 = tokenizer.texts_to_sequences(X_train3)
val_sequences3 = tokenizer.texts_to_sequences(X_val3)
test_sequences3 = tokenizer.texts_to_sequences(X_test3)

# Pad the sequences to ensure they are all of the same length
train_data3 = pad_sequences(train_sequences3, maxlen=maxlen)
val_data3 = pad_sequences(val_sequences3, maxlen=maxlen)
test_data3 = pad_sequences(test_sequences3, maxlen=maxlen)
```

Logistic Regression

In [36]:
```
pipe_lr_multi = Pipeline([
    ("vec", TfidfVectorizer(max_features=500000 )),
    ("clf", LogisticRegression(class_weight="balanced", solver="saga", max_iter=30000))
])

# param_grid = {
#     "vec__ngram_range" : [(1,2), (1,3)],
#     "clf__C" : [25, 26, 27]
# }

# gsearch = GridSearchCV(pipe_lr_multi, param_grid, cv=5, n_jobs=-1, scoring="f1_macro")
# gsearch.fit(X_train, y_train)

# gsearch.best_params_
```

In [37]:
```
best_params_ = {
    'clf__C': 26,
    'vec__ngram_range': (1, 2)
    }

pipe_lr_multi.set_params(**best_params_)
pipe_lr_multi.fit(X_train3, np.argmax(y_train3, axis=1))

# comb_y = np.concatenate((np.argmax(y_val3, axis=1).reshape(-1, 1), np.argmax(y_test3, axis=1).reshape(-1, 1)), axis=
metrics(np.argmax(y_test3, axis=1), pipe_lr_multi.predict(X_test3))
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reach
ed which means the coef_ did not converge
  warnings.warn(
```

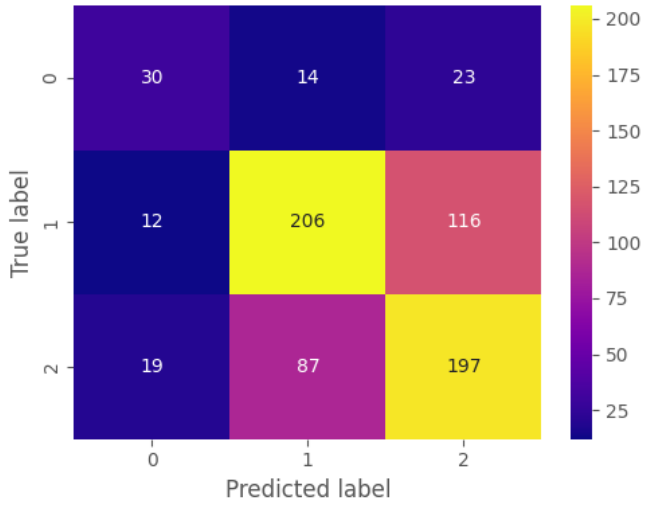


	precision	recall	f1-score	support
0	0.46	0.48	0.47	67
1	0.66	0.64	0.65	334
2	0.60	0.61	0.61	303
accuracy			0.61	704
macro avg	0.57	0.58	0.57	704
weighted avg	0.61	0.61	0.61	704

SVM - SVC

In [38]:
```
pipe_svm_multi = Pipeline([
    ("vec", TfidfVectorizer(max_features=int(5e6))),
    ("clf", SVC(class_weight="balanced", random_state=42))
])

# param_grid = {
#     "vec__ngram_range" : [(1,2), (1,3)],
#     "clf__kernel" : ["linear", "rbf"],
#     "clf__gamma" : ["scale", "auto"],
#     "clf__decision_function_shape" : ["ovr", "ovo"]
# }

# gsearch = GridSearchCV(pipe_svm_multi, param_grid, cv=5, n_jobs=-1, scoring="f1_macro")
# gsearch.fit(X_train, y_train)

# gsearch.best_params_
```

In [39]:
```
best_params_ = {
    'clf__decision_function_shape': 'ovr',
    'clf__gamma': 'scale',
    'clf__kernel': 'linear',
    'vec__ngram_range': (1, 2)
    }

pipe_svm_multi.set_params(**best_params_)
pipe_svm_multi.fit(X_train3, np.argmax(y_train3, axis=1))

svm_multi_preds = pipe_svm_multi.predict(X_test3)

metrics(np.argmax(y_test3, axis=1), svm_multi_preds)
```



	precision	recall	f1-score	support
0	0.49	0.45	0.47	67
1	0.67	0.62	0.64	334
2	0.59	0.65	0.62	303
accuracy			0.62	704
macro avg	0.58	0.57	0.58	704
weighted avg	0.62	0.62	0.61	704

In [40]:
```
errors = pd.DataFrame({
    "tweet" : X_test3,
    "sentiment" : np.argmax(y_test3, axis=1),
    "prediction" : svm_multi_preds
})

errors = errors[errors.sentiment != errors.prediction]
errors["original_tweet"] = raw_df.tweet_text.loc[errors.index]

errors
```

Out[40]:

	tweet	sentiment	prediction	original_tweet
177	course bing result good product seo product qagb	1	2	Of course Bing results are good they are Google's:) #SEO #SXSW #qagb
3612	go mock tech nerds come product stand hour line product remember horror movies	2	1	RT @mention I was going to mock the tech-nerds for coming to #sxsw only to stand in a 4-hour line for an iPad 2. Then I remembered me and horror movies.
1657	new social network may launch day product launch major new social network call product possibly today product	1	2	New Social Network may launch 2day! "Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxswï¿½Û
3534	hello holler gram come take look new first product app hollergram product	2	1	RT @mention Hello, Holler Gram! Come take a look at our new (and first!) iPad app {link} #hollergram #sxsw
1087	inside product chip bring browse boost best hungry product tip mekong river	1	2	Inside the iPad 2: chip brings 50% browsing boost: RT @mention Best Hungry at #sxsw Tip: Mekong River, g... {link}
...	...	...	...	...
1550	win free product webdoc com product	1	2	Win free iPad 2 from webdoc.com #sxsw RT
4814	product show product place hotpot allow post review place go search result map friends see others product	1	2	Google showing off google places with hotpot allows u to post review of place, goes in search results, maps for friends to see, others #sxsw
6350	wait give product someone product want head enter must present win	2	1	We can't wait to give an iPad to someone at #sxsw. Want in? Just head to www.pep.jobs/upc to enter. (must be present to win)
3214	delete product app	0	1	RT @mention Deleting the #sxsw iPhone app! {link}
6722	product officially best way advertise something	1	2	Twitter is officially the BEST way to advertise something.

271 rows × 4 columns

Neural Networks

Recalculate the class weights

In [41]:
```
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(df.sentiment), y=df.sentiment)
class_weight_dict = dict(enumerate(class_weights))


# No emotion toward brand or product    2
# Positive emotion                      1
# Negative emotion                      0

class_weight_dict
```

Out[41]:
```
{0: 3.476543209876543, 1: 0.754312654023358, 2: 0.7211636959639418}
```

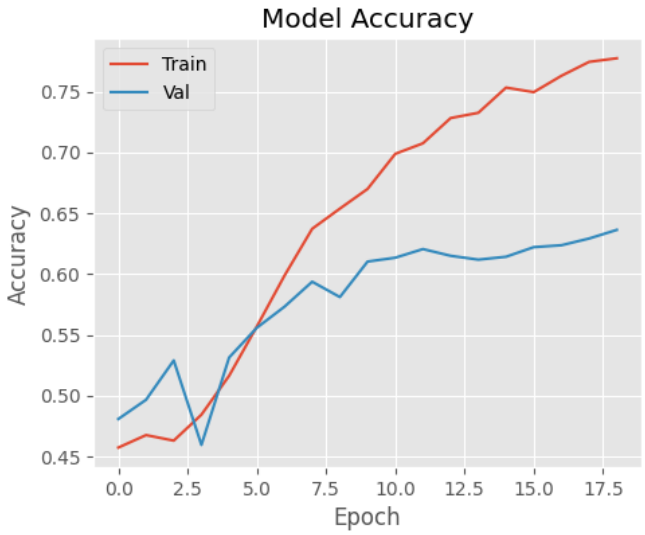
Retrying Best Binary Model

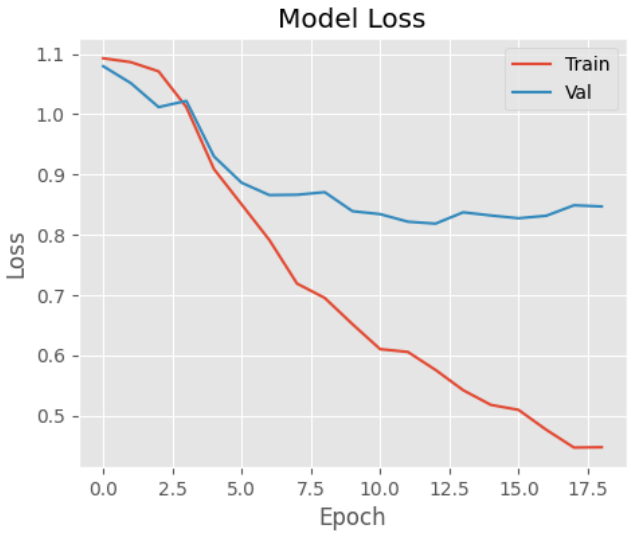
This is the exact same architecture as the previous `model_2` from the binary models. This will serve as a base model for the binary predictions.

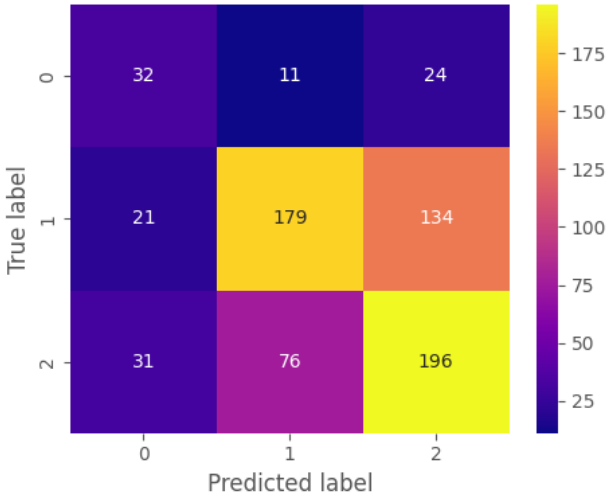
In [42]:
```
from tensorflow.keras.optimizers import Adam
from keras.layers import Dropout


multi_model_1 = Sequential()
multi_model_1.add(Embedding(input_dim=num_words, output_dim=128))

# Add a dropout layer to the model
multi_model_1.add(Dropout(0.5))

# Add an LSTM layer with 64 units and 0.5 dropout, returning sequences so for the following LSTM layer
multi_model_1.add(LSTM(64, dropout=0.5, return_sequences=True))

# Add another LSTM layer with 32 units and 0.3 dropout, not returning sequences
multi_model_1.add(LSTM(32, dropout=0.5))

multi_model_1.add(Dense(3, "softmax"))

multi_model_1.compile(optimizer=Adam(learning_rate=1e-4), loss="categorical_crossentropy", metrics=["accuracy", "Preci
history = multi_model_1.fit(
    train_data3, y_train3,
    verbose=global_verbose,
    validation_data=(val_data3, y_val3),
    epochs=30,
    batch_size=16,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        ],
    class_weight=class_weight_dict
)
```

In [43]:
```
analyze_model(history, multi_model_1, test_data3, y_test3)
```

```
22/22 [==============================] - 2s 53ms/step
```







	precision	recall	f1-score	support
0	0.38	0.48	0.42	67
1	0.67	0.54	0.60	334
2	0.55	0.65	0.60	303
accuracy			0.58	704
macro avg	0.54	0.55	0.54	704
weighted avg	0.59	0.58	0.58	704

Evaluation

The model performed signifficantly worse than the LR and SVM models, but did attempt to predict all three classes.

Extended Binary Model

Here we attempt to extend my best binary model to predict a third class.

Load Model Weights

In [44]:
```
from keras.models import load_model

binary_model = load_model("best_model.h5")

binary_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 128)         1280000

 dropout (Dropout)           (None, None, 128)         0

 lstm_1 (LSTM)               (None, None, 64)          49408

 lstm_2 (LSTM)               (None, 32)                12416

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,341,857
Trainable params: 1,341,857
Non-trainable params: 0
_________________________________________________________________
```

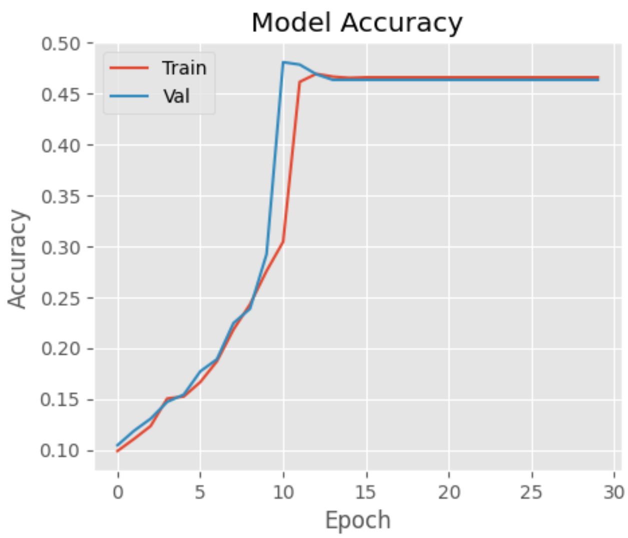
Adding new output

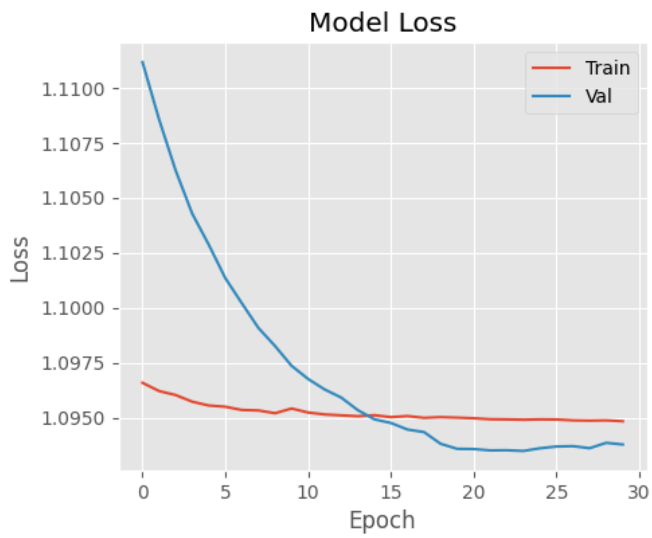
I adapt the model for a third class by freezing its current layers and adding additional layers to hopefully pick up on patterns of the third class. I add a dense layer with 10 neurons to hopefully discover multiclass patterns in the binary classification output.

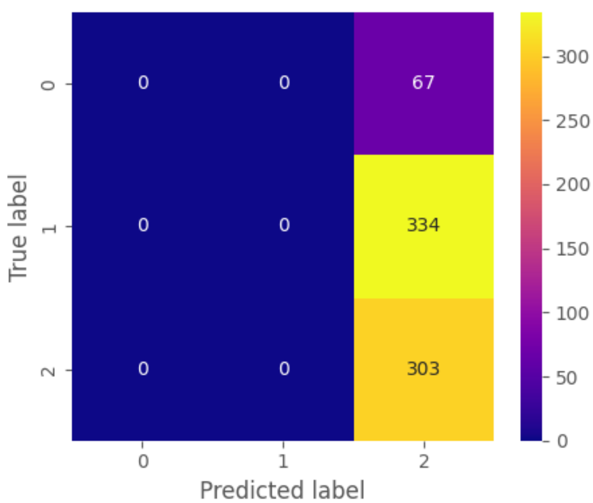
In [45]:
```
from keras.models import Model

# Freeze all layers of the binary classification model
for layer in binary_model.layers[:-1]:
    layer.trainable = False

# Add a new output layer with 3 classes using the output of the last layer of the binary model
new_output_layer = Dense(3, activation="softmax")(Dense(10, "relu")(binary_model.layers[-1].output))

# Create a new multi-class classification model with the binary model as its base layers and the new output layer
multi_class_model = Model(inputs=binary_model.inputs, outputs=new_output_layer)

# Compile the multi-class model with the Adam optimizer, categorical cross-entropy loss, and accuracy, precision, and
multi_class_model.compile(optimizer=Adam(learning_rate=1e-4), loss="categorical_crossentropy", metrics=["accuracy", "P

# Fit the multi-class model on the training data and validate on the validation data for 25 epochs with a batch size o
# Use EarlyStopping to prevent overfitting and restore the best weights of the model at each epoch
history = multi_class_model.fit(
    train_data3,
    y_train3,
    verbose=global_verbose,
    validation_data=(val_data3, y_val3),
    epochs=40,
    batch_size=16,
    callbacks=[EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True)],
    class_weight = class_weight_dict
)
```

In [46]:
```
analyze_model(history, multi_class_model, test_data3, y_test3)
```

```
22/22 [==============================] - 1s 22ms/step

/usr/local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```







	precision	recall	f1-score	support
0	0.00	0.00	0.00	67
1	0.00	0.00	0.00	334
2	0.43	1.00	0.60	303
accuracy			0.43	704
macro avg	0.14	0.33	0.20	704
weighted avg	0.19	0.43	0.26	704

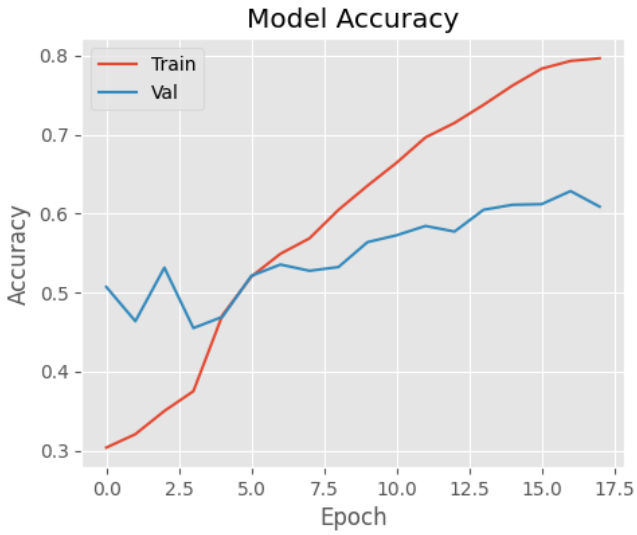
Evaluation

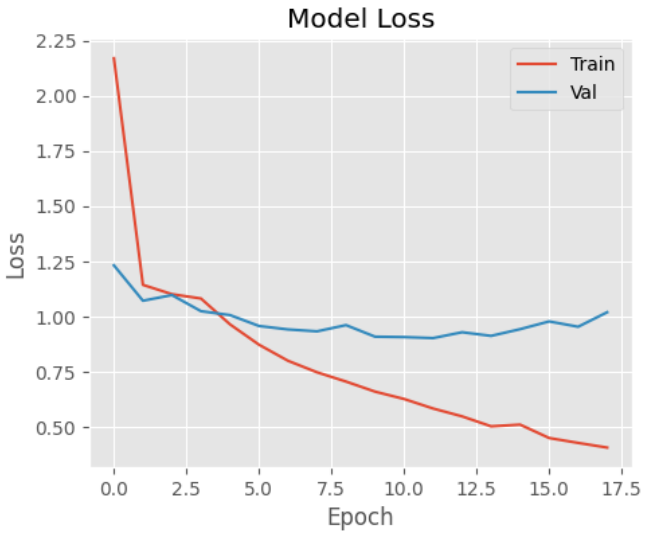
This did not work. The accuracy and f1 scores are signifficantly lower than the previous LR and SVM models. Despite having previously being trained on the binary classification task, it appear that the new final output layer has interpeted the binary output in such a way that the model only predict the Positive and Neutral classes. I have spent some time tinkering with the hyperparameters to no avail, so I think this route is a dud.

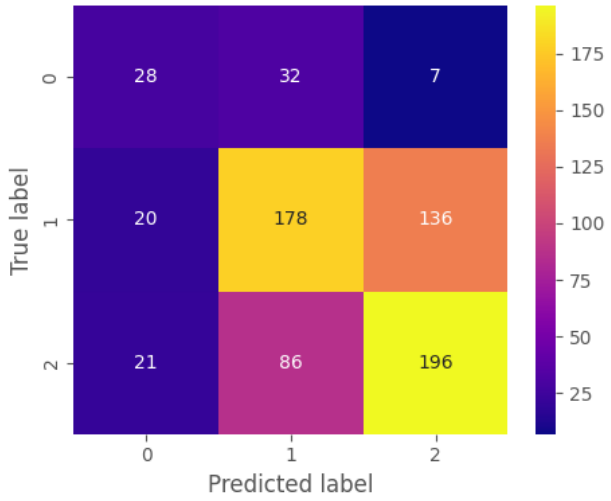
In [47]:
```
from keras.layers import SpatialDropout1D, Conv1D, MaxPooling1D, concatenate, Input, Flatten
from keras.regularizers import l2

# Model Layers
input_layer = Input(shape=(maxlen,))
embedding_layer = Embedding(num_words, 128, input_length=maxlen)(input_layer)

dropout_layer = SpatialDropout1D(0.5)(embedding_layer) # Overfitting mitigation

biltsm = Bidirectional(LSTM(192, dropout=0.2, return_sequences=True))(dropout_layer)

# Add Conv1D Layers with MaxPooling for different kernel sizes to capture n-grams
conv1 = MaxPooling1D()(Conv1D(filters=64, kernel_size=2, activation="relu")(biltsm)) # 2 grams
conv2 = MaxPooling1D()(Conv1D(filters=64, kernel_size=3, activation="relu")(biltsm)) # 3 grams
conv3 = MaxPooling1D()(Conv1D(filters=64, kernel_size=5, activation="relu")(biltsm)) # 5 grams
convs = concatenate([conv1, conv2, conv3], axis=1)

dropout_layer_2 = SpatialDropout1D(0.3)(convs) # Overfitting mitigation

dense = Dense(192, activation="relu", kernel_regularizer=l2())(Flatten()(dropout_layer_2))
output = Dense(3, activation="softmax")(dense)

# Create Model
model = Model(inputs=input_layer, outputs=output)
model.compile(optimizer=Adam(learning_rate=1e-4), loss="categorical_crossentropy", metrics=["accuracy", "Precision", "

# Train and Evaluate
history = model.fit(
    train_data3, y_train3,
    verbose=global_verbose,
    epochs = 25,
    batch_size = 16,
    validation_data = (val_data3, y_val3),
    class_weight=class_weight_dict,
    callbacks=[
        EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        ],
    )
```

In [48]:
```
analyze_model(history, model, test_data3, y_test3)
```

```
22/22 [==============================] - 2s 79ms/step
```





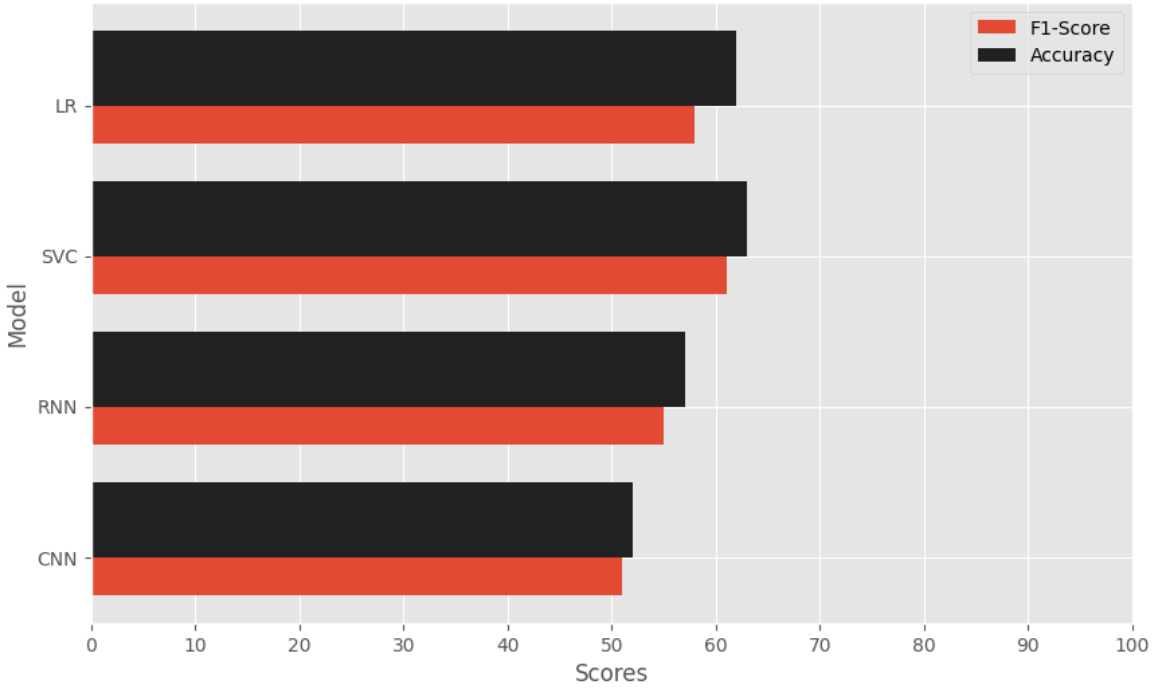


	precision	recall	f1-score	support
0	0.41	0.42	0.41	67
1	0.60	0.53	0.57	334
2	0.58	0.65	0.61	303
accuracy			0.57	704
macro avg	0.53	0.53	0.53	704
weighted avg	0.57	0.57	0.57	704

Multiclass Results Summary

In [49]:
```
fig, ax = plt.subplots(figsize=(10,6))

models = ["LR", "SVC", "RNN", "CNN"]
accuracies = [62, 63, 57, 52]
f1s = [58, 61, 55, 51]

for ls in [models, accuracies, f1s]:
    ls.reverse()


ax.barh(models, f1s, label="F1-Score", height=0.5)
ax.barh(models, accuracies, label="Accuracy", height=0.5, align="edge", color="#212121")
ax.legend()
ax.set_xlabel("Scores")
ax.set_ylabel("Model")
ticks = [10 * i for i in range(11)]
ax.set_xticks(ticks)
```

Out[49]:
```
[<matplotlib.axis.XTick at 0x16c5c0fa0>,
 <matplotlib.axis.XTick at 0x16c5c1000>,
 <matplotlib.axis.XTick at 0x1783bf940>,
 <matplotlib.axis.XTick at 0x16c5c80a0>,
 <matplotlib.axis.XTick at 0x16c5c86d0>,
 <matplotlib.axis.XTick at 0x16c5c8e20>,
 <matplotlib.axis.XTick at 0x16c5c9360>,
 <matplotlib.axis.XTick at 0x16c5c9450>,
 <matplotlib.axis.XTick at 0x16c5c9ba0>,
 <matplotlib.axis.XTick at 0x16c5ca2f0>,
 <matplotlib.axis.XTick at 0x16c5caa40>]
```



Conclusion

This notebook explores NLP sentiment analysis for tweets about various products using a pre-made dataset. The tweets were preprocessed and cleaned before training various machine learning models to classify the sentiment of the tweets.

Binary sentiment classification (positive and negative) achieved higher accuracy compared to the multi-class model. The difficulty in training models in this space is due to a number of factors. Firstly, tweets are a particularly difficult medium to perform NLP on because of the short length, use of slang, and intentional and unintentional grammatical and spelling errors. Second is the size of our dataset. After undersampling the majority class, we were left with less than 6k tweets. Many of the highest performing NLP models are trained on millions if not billions of datapoints. Finally, I believe that the dataset's labeling contains many errors and/or tweets that people may not label into any one category as they contain both positive and negative aspects.

While the results may be limited by the size and diversity of the dataset, sentiment analysis of tweets can provide valuable insights into customer opinions and attitudes towards products or services. This work demonstrates the potential for NLP sentiment analysis to be used in real-world applications for business decision-making.