Explanation of code:

Importing several Python libraries commonly used for data analysis, natural language processing, and data visualization.

- numpy (imported as np): A library for numerical computations in Python.
- pandas (imported as pd): A library for data manipulation and analysis. It provides data structures and functions for efficient handling of structured data.
- re: The built-in Python module for regular expressions, used for pattern matching and text manipulation.
- nltk: The Natural Language Toolkit is a library for natural language processing tasks, such as tokenization, stemming, and part-of-speech tagging.
- matplotlib.pyplot (imported as plt): A library for creating static, animated, and interactive visualizations in Python. The %matplotlib inline command is a Jupyter Notebook magic command that allows plots to be displayed directly in the notebook.

These libraries are often used together for analyzing textual data, preprocessing text, and visualizing the results.

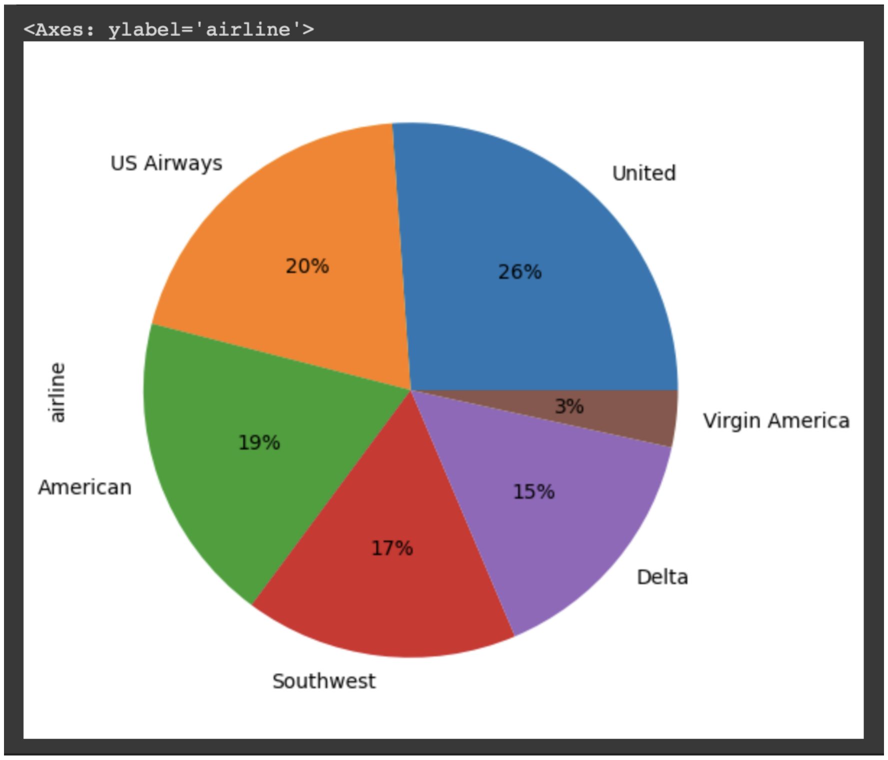
The code “airline_tweets = pd.read_csv('Tweets.csv')” reads a CSV file named "Tweets.csv" and assigns the data to the variable airline_tweets. It assumes that the CSV file is located in the current working directory.

To access the column names of the “airline_tweets” DataFrame, you can use the columns attribute.

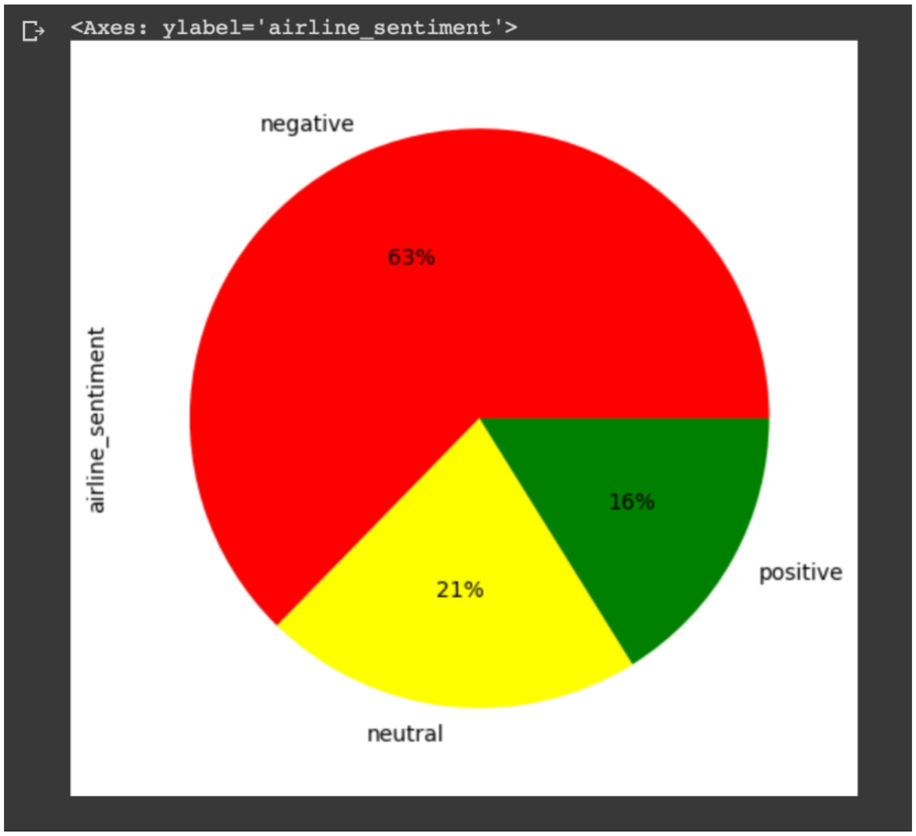
The code “airline_tweets.head()” is used to display the first few rows of the airline_tweets DataFrame. The” head()” method in pandas returns the specified number of rows from the beginning of a DataFrame. By default, it returns the first five rows.

1. **airline_tweets.columns:** This line retrieves the column names of the airline_tweets DataFrame. It returns a pandas Index object containing the column names.
2. **airline = airline_tweets[['text','airline_sentiment']]:** This line creates a new DataFrame called airline by selecting only the 'text' and 'airline_sentiment' columns from the airline_tweets DataFrame. The double brackets [['text','airline_sentiment']] are used to specify a list of column names.

3. **airline["airline_sentiment"].value_counts():** This line calculates the count of each unique value in the 'airline_sentiment' column of the airline DataFrame. It returns a Series object with the sentiment values as the index and their corresponding counts as the values.
4. **plot_size = plt.rcParams["figure.figsize"]:** This line retrieves the current size of the figure in matplotlib. The plt.rcParams object holds the default parameters for creating plots, and "figure.figsize" specifies the figure size as a tuple of width and height.
5. **print(plot_size[0]) and print(plot_size[1]):** These lines print the current width and height of the figure, respectively, using indexing on the plot_size tuple.
6. **plot_size[0] = 8 and plot_size[1] = 6:** These lines update the width and height of the figure by modifying the values in the plot_size tuple.
7. **plt.rcParams["figure.figsize"] = plot_size:** This line sets the figure size in matplotlib to the updated plot_size.
8. **airline_tweets.airline.value_counts().plot(kind='pie', autopct='%1.0f%%'):** This line creates a pie chart using the value_counts() function on the 'airline' column of the airline_tweets DataFrame. It counts the occurrences of each unique airline and plots the percentages as slices of the pie chart. The kind='pie' argument specifies the chart type as a pie chart, and autopct='%1.0f%%' displays the percentage value on each slice.


```
<Axes: ylabel='airline'>
```

airline_tweets.airline_sentiment.value_counts().plot(kind='pie', autopct='%1.0f%%', colors=["red", "yellow", "green"]): This line creates a pie chart of the counts of each sentiment value in the 'airline_sentiment' column of the airline_tweets DataFrame. The kind='pie' argument specifies the chart type as a pie chart, autopct='%1.0f%%' displays the percentage value on each slice, and colors=["red", "yellow", "green"] specifies the colors of each slice.

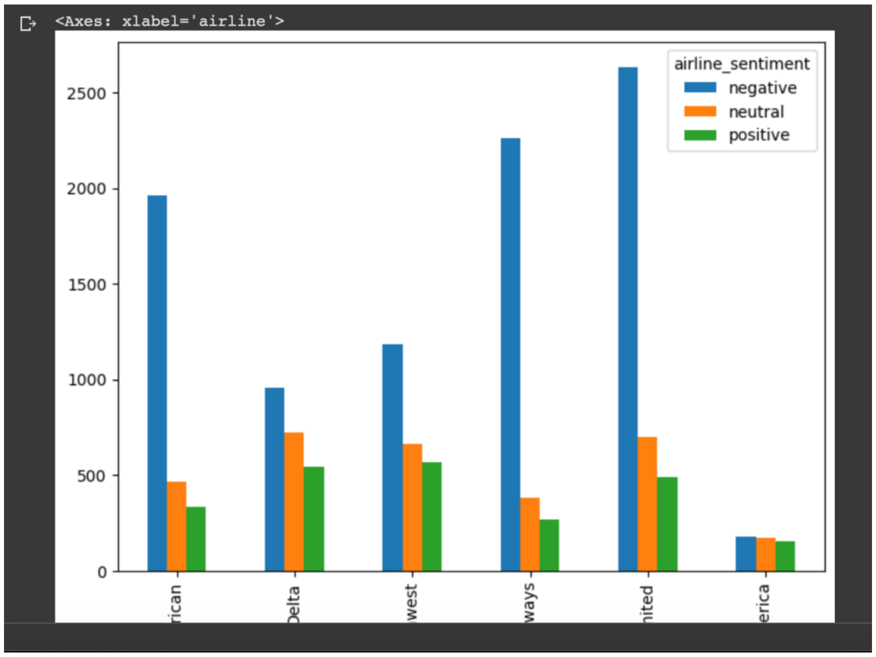


```
<Axes: ylabel='airline_sentiment'>
```

airline_sentiment = airline_tweets.groupby(['airline', 'airline_sentiment']).airline_sentiment.count().unstack(): This line groups the airline_tweets DataFrame by the 'airline' and 'airline_sentiment' columns and counts the number of occurrences for each group. The unstack() function then reshapes the data from a stacked Series to a DataFrame with 'airline_sentiment' as the columns and 'airline' as the index. The resulting DataFrame is stored in airline_sentiment.

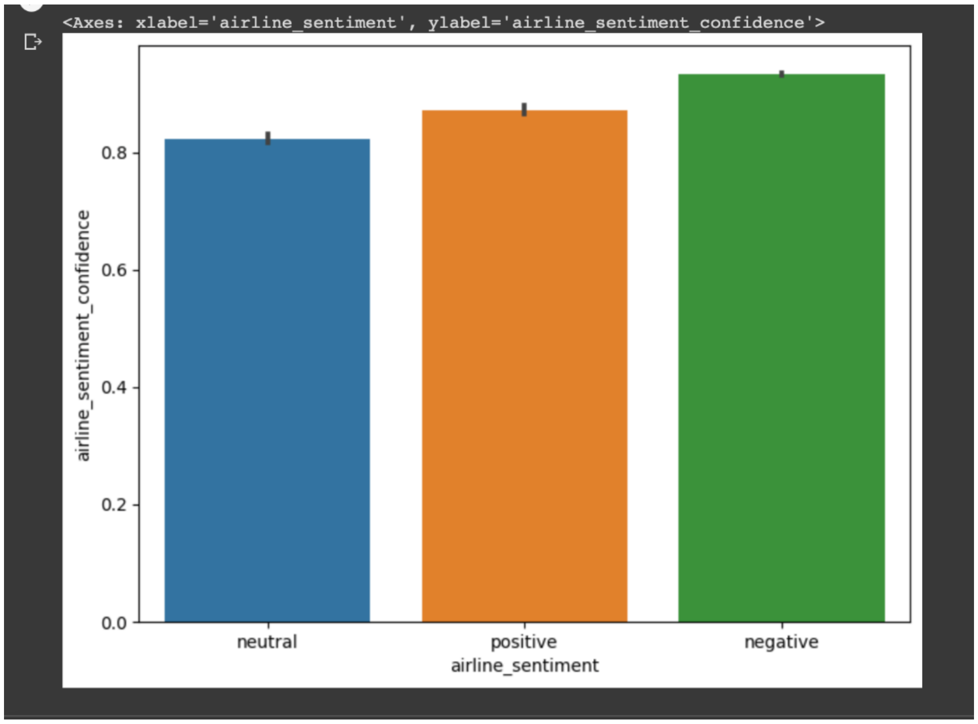
airline_sentiment.plot(kind='bar'): This line creates a bar chart of the counts of each sentiment value for each airline in the airline_sentiment DataFrame.

```
<Axes: xlabel='airline'>
```



import seaborn as sns and sns.barplot(x='airline_sentiment', y='airline_sentiment_confidence' , data=airline_tweets): These lines create a bar chart using seaborn library with the 'airline_sentiment' column on the x-axis, the 'airline_sentiment_confidence' column on the y-axis, and the data source as the airline_tweets DataFrame.

```
<Axes: xlabel='airline_sentiment', ylabel='airline_sentiment_confidence'>
```



1. **features = airline_tweets.iloc[:, 10].values and labels = airline_tweets.iloc[:, 1].values:** These lines extract the 'text' and 'airline_sentiment' columns from the airline_tweets DataFrame and store them in the features and labels variables, respectively.
2. **processed_features = []:** This line initializes an empty list to store the processed text data.
3. The following code block is a loop that processes each sentence in the features list using regular expressions and other string operations. Here's a breakdown of what each operation does:

- re.sub(r'\W', ' ', str(features[sentence])): removes all non-word characters.
- re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_feature): removes all single characters.
- re.sub(r'\^[a-zA-Z]\s+', ' ', processed_feature): removes single characters from the start of the sentence.
- re.sub(r'\s+', ' ', processed_feature, flags=re.I): substitutes multiple spaces with a single space.
- re.sub(r'^b\s+', '', processed_feature): removes the 'b' prefix at the start of the sentence.
- processed_feature.lower(): converts the sentence to lowercase.

8. **from nltk.corpus import stopwords:** This line imports the stopwords module from the Natural Language Toolkit (NLTK) library.
9. **from sklearn.feature_extraction.text import TfidfVectorizer:** This line imports the TfidfVectorizer class from the scikit-learn library.
10. **vectorizer = TfidfVectorizer(max_features=2500, min_df=7, max_df=0.8, stop_words=('english')):** This line creates an instance of the TfidfVectorizer class from scikit-learn. This vectorizer is used to convert the processed text data into a numerical representation using the TF-IDF (Term Frequency-Inverse Document Frequency) scheme. The parameters are:
    a. **max_features=2500:** It specifies the maximum number of features (unique words) to be used. Only the top 2500 most frequent words will be considered.
    b. **min_df=7:** It sets the minimum threshold for the word frequency. Words that appear in fewer than 7 documents will be ignored.
    c. **max_df=0.8:** It sets the maximum threshold for the word frequency. Words that appear in more than 80% of the documents will be ignored.
    d. **stop_words=('english'):** It specifies that common English stopwords should be removed from the text data during the vectorization process.

11. **processed_features = vectorizer.fit_transform(processed_features).toarray():** This line applies the fit_transform() method of the TfidfVectorizer to the processed_features list. It performs the vectorization process and converts the processed text data into a numeric matrix representation. The resulting matrix is stored in processed_features.
12. **from sklearn.model_selection import train_test_split:** This line imports the train_test_split function from scikit-learn. This function is used to split the dataset into training and testing sets for machine learning model evaluation.
13. **X_train, X_test, y_train, y_test = train_test_split(processed_features, labels, test_size=0.2, random_state=42):** This line splits the processed_features matrix and the labels array into training and testing sets. The training set (X_train and y_train) will be used to train a machine learning model, while the testing set (X_test and y_test) will be used to evaluate the model's performance. The test_size parameter is set to 0.2, indicating that 20% of the data will be used for testing. The random_state parameter is set to 42 to ensure reproducibility of the split, meaning the same split will be obtained if the code is run again.

Using Random Forest:

a Random Forest Classifier model is trained on the features (X_train) and labels (y_train) data. Here's an explanation of the code:

1. **from sklearn.ensemble import RandomForestClassifier:** This line imports the RandomForestClassifier class from the sklearn.ensemble module. Random Forest is an ensemble learning method that combines multiple decision trees to make predictions.
2. **text_classifier = RandomForestClassifier(n_estimators=200, random_state=42):** This line creates an instance of the RandomForestClassifier class with 200 estimators (number of decision trees) and a random state of 42. The random state is set for reproducibility, ensuring that the results can be replicated.
3. **text_classifier.fit(X_train, y_train):** This line trains the random forest classifier using the fit() method. It takes the training features (X_train) and corresponding labels (y_train) as input and builds the model.
4. **predictions = text_classifier.predict(X_test):** This line uses the trained classifier to make predictions on the test features (X_test). The predict() method assigns a predicted label to each test sample.

5. **from sklearn.metrics import confusion_matrix, accuracy_score:** This line imports the confusion_matrix and accuracy_score functions from the sklearn.metrics module. These functions are used to evaluate the performance of the classifier.
6. **print(confusion_matrix(y_test, predictions)):** This line prints the confusion matrix, which is a table that summarizes the performance of a classification model. It shows the number of true positive, true negative, false positive, and false negative predictions.
7. **print('accuracy score', accuracy_score(y_test, predictions)):** This line calculates and prints the accuracy score, which measures the proportion of correctly predicted labels among all the test samples.

The code segment demonstrates the training of a Random Forest Classifier model, making predictions on the test data, and evaluating the model's performance using the confusion matrix and accuracy score.

Using KNN Algorithm:

K-Nearest Neighbors (KNN) Classifier model is trained on the features (X_train) and labels (y_train) data. Here's an explanation of the code:

1. **from sklearn.neighbors import KNeighborsClassifier:** This line imports the KNeighborsClassifier class from the sklearn.neighbors module. KNN is a type of instance-based learning algorithm that classifies new instances based on the majority class of their neighboring instances.
2. **knn = KNeighborsClassifier(n_neighbors=200):** This line creates an instance of the KNeighborsClassifier class with n_neighbors set to 200. This parameter determines the number of neighbors used for classification.
3. **knn.fit(X_train, y_train):** This line trains the KNN classifier using the fit() method. It takes the training features (X_train) and corresponding labels (y_train) as input and builds the model.
4. **y_pred = knn.predict(X_test):** This line uses the trained classifier to make predictions on the test features (X_test). The predict() method assigns a predicted label to each test sample.

5. **print(confusion_matrix(y_test, y_pred)):** This line prints the confusion matrix, which summarizes the performance of the classifier by showing the number of true positive, true negative, false positive, and false negative predictions.
6. **print("Accuracy:", accuracy_score(y_test, y_pred)):** This line calculates and prints the accuracy score, which measures the proportion of correctly predicted labels among all the test samples.

The code segment demonstrates the training of a K-Nearest Neighbors Classifier model, making predictions on the test data, and evaluating the model's performance using the confusion matrix and accuracy score.