Fake News Detection - Introduction

This project stands at the forefront of combating the escalating issue of misinformation in the media and on the internet, harnessing the power of data-driven insights and machine learning technologies. Central to the initiative is the LIAR dataset, an extensively curated repository of statements evaluated for their truthfulness across a spectrum from "true" to "pants on fire." This resource is instrumental in the endeavor to find patterns, indicators, and characteristics that signify fake news. Now, I embark on a mission to explore pivotal questions, such as the tendency of certain individuals or entities to disseminate misinformation, the role of specific contexts or platforms in amplifying fake news, and the prevalence of misinformation within discussions on various subjects.

Objective: I aim to answer questions such as:

- Are certain individuals or entities more inclined to propagate misinformation?
- Is misinformation more rampant in discussions about certain subjects, and does political affiliation influence the spread of false information?
- Is it possible to reliably detect whether a politically charged statement is true or false using models trained on similar statements.

Methodology: My approach combines exploratory data analysis (EDA) with advanced machine learning techniques. The EDA phase involves a deep dive into the LIAR dataset, enabling the formulation of hypotheses about the nature of fake news. Through this analysis, we can uncover the relationships between the veracity of statements and various factors like speaker identities, subject matters, and political leanings.

In the machine learning phase, I'll leverage models such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Decision Trees, and XGBoost. These models are trained to classify statements based on their truthfulness, employing text data that has been preprocessed through TF-IDF vectorization and standard scaling to optimize model performance.

Liar Training Dataset

We're first going to look at the training data of the dataset to form a hypothesis.

The overall question I'm trying to answer in this section is that are there certain indicators to detect fake news; such as certain people being more likely to spread false information, certain contexts where fake news is spread (like some podcasts, or some speakers, etc.), are some subjects more prone to having misinformation spread by a certain political party?

While exploring the data, I'll form more questions from that and building visualizations to show the data and explain it.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.svm import SVC
import re
import string
```

Exploratory Data Analysis

While exploring the training data information, we'll want to identify any relationships that are found in the data.

Questions to look into:

- Are there any relationships between how misinformation is spread?
- Do certain people have a pattern of spreading misinformation?
- Do political parties spread misinformation among different subjects?
- Can we quantify the label column as either true or false?

Focusing the Data

In order to simplify the LIAR dataset for the machine learning models, I'll map the truth identifiers to a "Truth Index" that only has two values instead of six. True to half-true will map to a 1 or true, while barely-true to false will map to a 0 or false.

In [ ]:
```
# Open the csv test file and label the columns (Names from the README File)
liar = pd.read_csv("train.tsv", sep='\t', header=None,
                           names=["ID", "Label", "Statement", "Subject", "Speaker",
                                  "State Info", "Party Affiliation", "Barely True
                                 "Half True Count", "Mostly True Count", "Pants on

test = pd.read_csv("test.tsv", sep='\t', header=None, names=["ID", "Label", "Statem
                                                             "State Info", "Party
                                                             "Half True Count", "M

hierarchy = {
    'true': 1,
    'mostly-true': 1,
    'half-true': 1,
    'barely-true': 0,
```

```
    'false': 0,
    'pants-fire': 0
}

liar["Truth Index"] = liar["Label"].map(hierarchy)
test["Truth Index"] = test["Label"].map(hierarchy)

liar
```

Out[ ]:

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
0	2635.json	false	Says the Annies List political group supports ...	abortion	dwayne-bohac	State representative	Texas
1	10540.json	half-true	When did the decline of coal start? It started...	energy,history,job-accomplishments	scott-surovell	State delegate	Virginia
2	324.json	mostly-true	Hillary Clinton agrees with John McCain "by vo...	foreign-policy	barack-obama	President	Illinois
3	1123.json	false	Health care reform legislation is likely to ma...	health-care	blog-posting	NaN	NaN
4	9028.json	half-true	The economic turnaround started at the end of ...	economy,jobs	charlie-crist	NaN	Florida
...	...	...	...	...	...	...	...
10235	5473.json	mostly-true	There are a larger number of shark attacks in ...	animals,elections	aclu-florida	NaN	Florida
10236	3408.json	mostly-true	Democrats have now become the party of the [At...	elections	alan-powell	NaN	Georgia
10237	3959.json	half-true	Says an alternative to Social Security that op...	retirement,social-security	herman-cain	NaN	Georgia

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
10238	2253.json	false	On lifting the U.S. Cuban embargo and allowing...	florida,foreign-policy	jeff-greene	NaN	Florida
10239	1155.json	pants-fire	The Department of Veterans Affairs has a manua...	health-care,veterans	michael-steele	chairman of the Republican National Committee	Marylanc

10240 rows × 15 columns

In [ ]:
```
# list all possible speakers
speakers = liar.groupby("Speaker").size().reset_index(name="count").sort_values(by=
speakers[speakers["count"] >= 75]
```

Out[ ]:

	Speaker	count
182	barack-obama	488
774	donald-trump	273
1112	hillary-clinton	239
1952	mitt-romney	176
2493	scott-walker	149
1416	john-mccain	148
2329	rick-perry	142
409	chain-email	142
1743	marco-rubio	117
2332	rick-scott	115
2649	ted-cruz	93
215	bernie-s	88
447	chris-christie	78
905	facebook-posts	78

Are some speakers more likely to spread misinformation/fake news than others?

The first idea I'll explore is the relationship between a speaker and how often their statements turn out to be true or false.

- For looking at the relationships initially, I'll be focusing on three different people: former president Barack Obama, George Allen, and former president Donald Trump.
- The x-axis will be the different labels (true to pants-fire) and the y-axis will be the amount of statements that fall under each label

In [ ]:
```
# look at the specific speaker Barack Obama
# Does he seem to say what is normally one label or the other (are his statements m
liar_dataset_obama = liar.loc[liar["Speaker"] == "barack-obama"]
# liar_dataset_obama.sort_values(by="ID", ascending=False)
# Group by ID to focus on the labels of the articles. Not the subjects
lds_obama_grouped = liar_dataset_obama.groupby(["Label", "Truth Index"]).size().res
lds_obama_grouped
```

Out[ ]:

	Label	Truth Index	count
2	half-true	1	124
3	mostly-true	1	130
5	true	1	103
0	barely-true	0	56
1	false	0	67
4	pants-fire	0	8

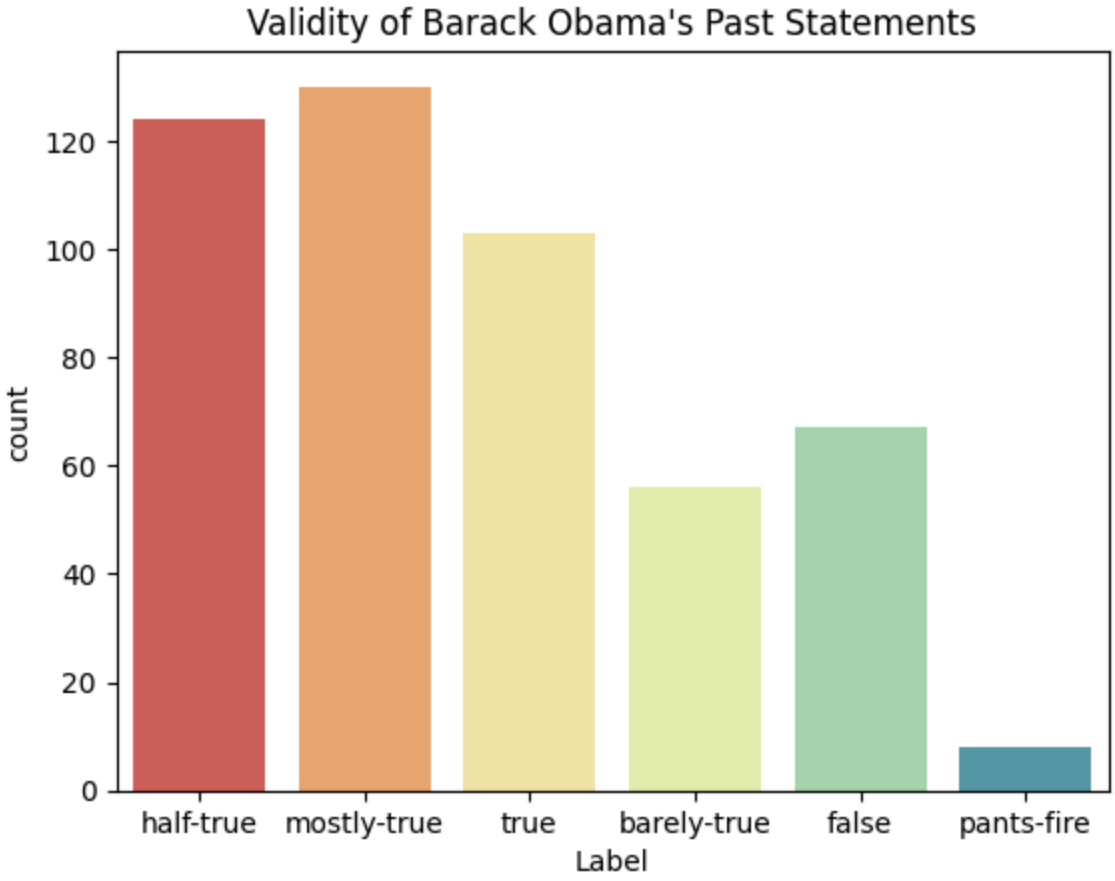
Creating the Visualisations

- By making these graphs, we will be able to study the relationship of individuals and how often they make truthful statements.

In [ ]:
```
# Create a visualization showing how Barack Obama's statements have been labeled
obama_barplot = sns.barplot(data=lds_obama_grouped, x="Label", y="count", palette="
# sns.color_palette("Spectral", as_cmap=True)
obama_barplot.set_title("Validity of Barack Obama's Past Statements")
obama_barplot
```

```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\2825073065.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  obama_barplot = sns.barplot(data=lds_obama_grouped, x="Label", y="count", palette
="Spectral")
```

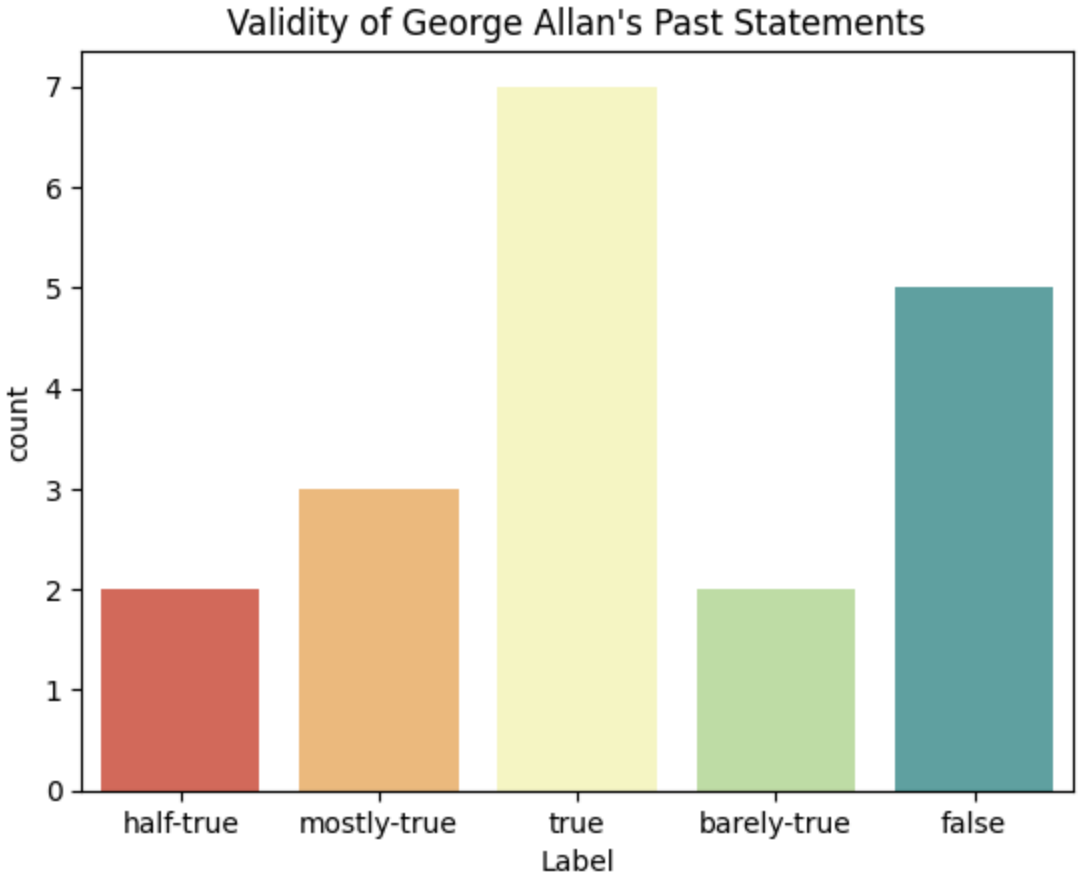
Out[ ]:
```
<Axes: title={'center': "Validity of Barack Obama's Past Statements"}, xlabel='Lab
el', ylabel='count'>
```



In [ ]:
```
# Create a visualization showing how George Allen's statements have been labeled
# speaker = liar_dataset.groupby(["Speaker", "Statement", "Speaker Job Title", "Tru
# speaker_statements = liar_dataset.groupby(["Speaker", "Statement",
#                                    "Truth Index", "Label"]).size().reset_index(nam
george_allen_dataset = liar.loc[liar["Speaker"] == "george-allen"]

george_allen_grouped = george_allen_dataset.groupby(["Label", "Truth Index"]).size(

george_allen_barplot = sns.barplot(data=george_allen_grouped, x="Label", y="count",
george_allen_barplot.set_title("Validity of George Allan's Past Statements")
george_allen_barplot
```

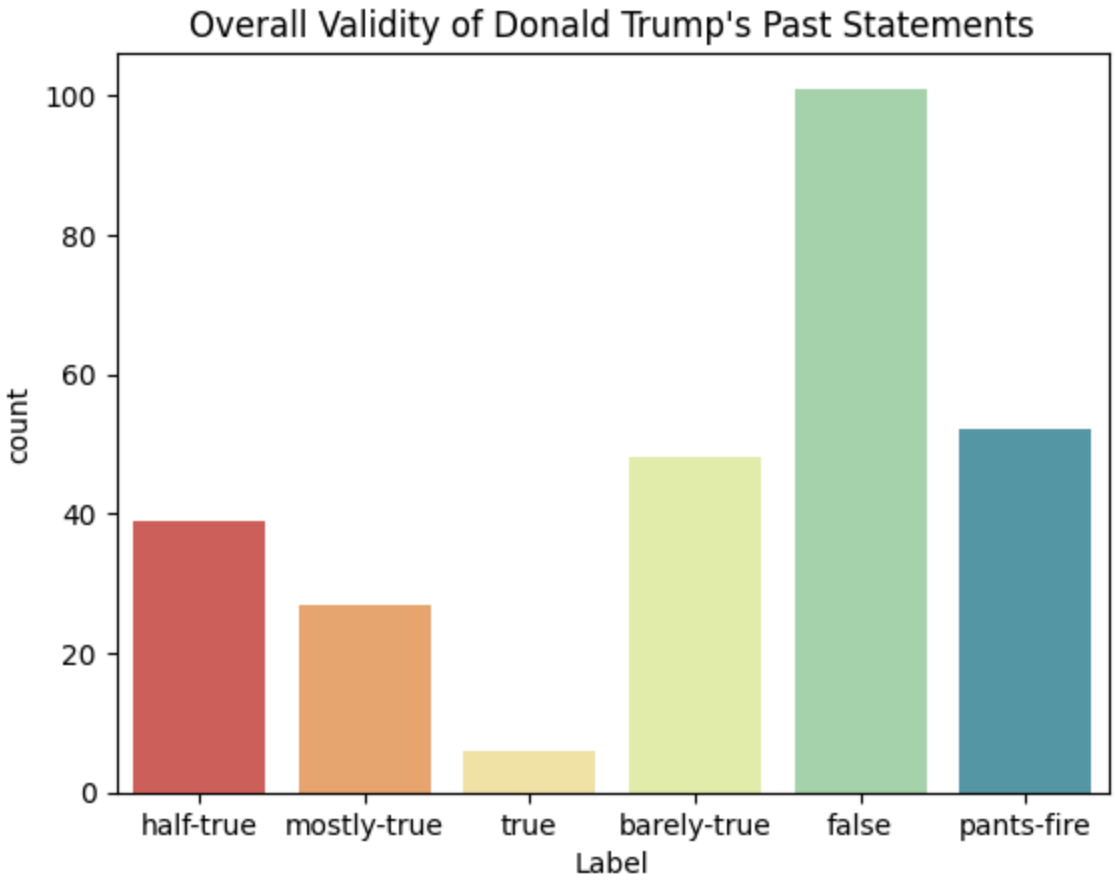
```
<ipython-input-9-e2444408ed97>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  george_allen_barplot = sns.barplot(data=george_allen_grouped, x="Label", y="coun
t", palette="Spectral")
```

Out[ ]:
```
<Axes: title={'center': "Validity of George Allan's Past Statements"}, xlabel='Lab
el', ylabel='count'>
```



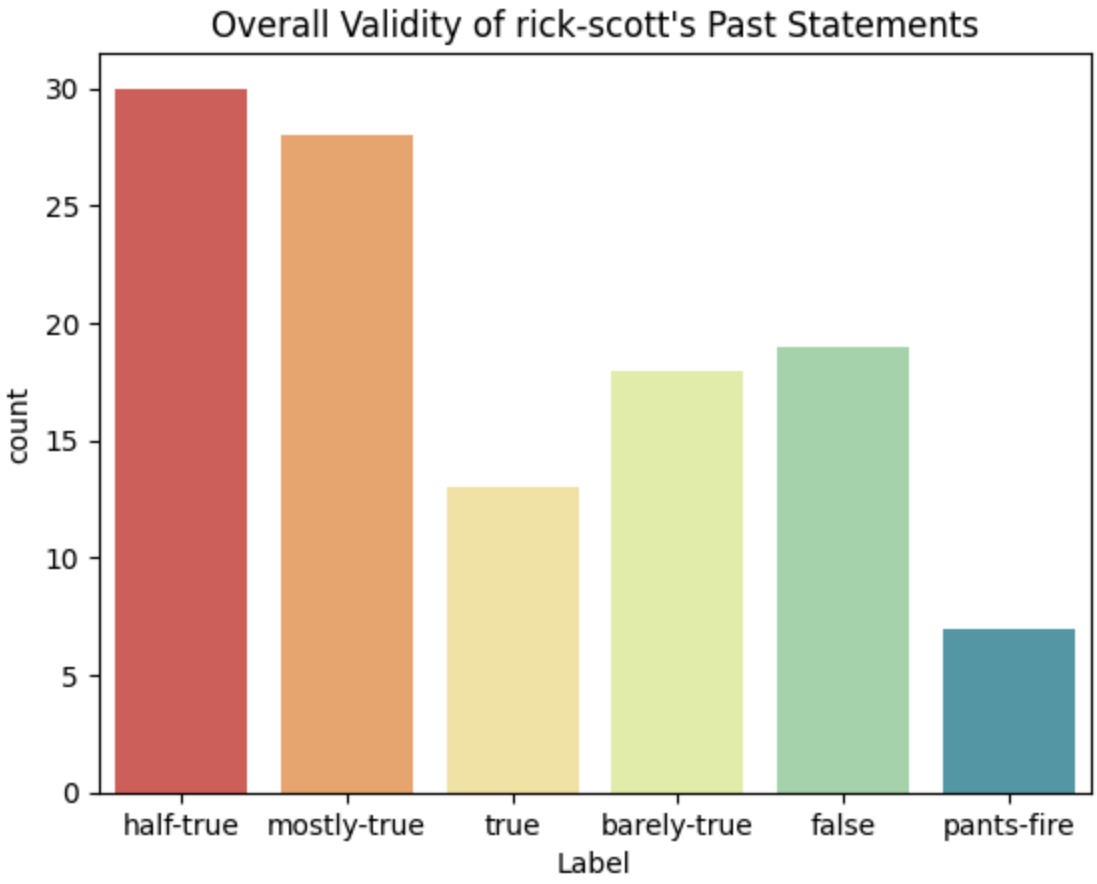
In [ ]:
```
donald_trump_dataset = liar.loc[(liar["Speaker"] == "donald-trump")]

donald_trump_grouped = donald_trump_dataset.groupby(["Label", "Truth Index"]).size(

donald_trump_barplot = sns.barplot(data=donald_trump_grouped, x="Label", y="count",
donald_trump_barplot.set_title("Overall Validity of Donald Trump's Past Statements"

donald_trump_barplot
```

Out[ ]:
```
<Axes: title={'center': "Overall Validity of Donald Trump's Past Statements"}, xla
bel='Label', ylabel='count'>
```



In [ ]:
```
# Create a function to create visualizations for more speakers
def analyze_truth_by_individual(speaker, liar_dataset):
  dataset = liar_dataset.loc[(liar_dataset["Speaker"] == speaker)]

  dataset_grouped = dataset.groupby(["Label", "Truth Index"]).size().reset_index(na

  barplot = sns.barplot(data=dataset_grouped, x="Label", y="count", palette="Spectr
  barplot.set_title("Overall Validity of " + speaker + "'s Past Statements")

  barplot
```

In [ ]:
```
analyze_truth_by_individual("rick-scott", liar)
# analyze_truth_by_individual("hillary-clinton", liar)
# analyze_truth_by_individual("marco-rubio", liar)
```



Creating a honesty-dishonesty ratio and plotting of other individuals

Now that we have looked at the truthfulness of different individuals, we can make a scatterplot of individuals and where they lie in terms of a ratio of honesty and dishonesty to all statements.

- Those with a higher ratio would be considered as more honest than those who are not
- Some individuals tend to have more statements put out compared to others.It's crucial to consider not just the absolute quantity of truth versus lies spoken by an individual, but also the ratio of truth to lies when making our scatterplot.

By looking at the ratio of honesty and dishonesty to all statements, we can better analyze behaviors among people, and treat those with higher ratios as more trustworthy compared to others. For example, if person A has a higher number of truthful statements compared to person B, but person A tends to have a lower ratio of truths to lies compared to B, we can infer that A would have a higher propensity to lie than B.

The X axis will be the count of lies from an indivudal, and the Y axis will be the count of truthful statements from the individual.

How do we interpret the scatterplot?

- We are going to focus on individuals with more than 50 statements out. It can be assumed that the people with more than 50 statements out have a larger platform, than those who do not.
- I've made a graph that will show the proportions of lies to truths. In the scatterplot, we have individuals that seem to lie more often than tell the truth.
    - An example is former president Donald Trump, shown as a light yellow-green point. In the graphs above, he is shown to have a high rate of making false statements, and as shown in the scatterplot, he has a lower true to false statements.
- If you are to draw a line on the scatterplot with a slope of 1, an individual with equal amounts of lies and truths would fall on it. A person who falls above that line would have more truthful statements, and we can consider this person to be more honest than someone who falls below that line.

In [ ]:
```
# get the speakers and add number of statements, separated by truth index
speakers_stmts_by_truth_index = liar.groupby(["Speaker", "Truth Index"]).size().res
speakers_stmts_by_truth_index

speakers_stmt_count = liar.groupby('Speaker').size().reset_index(name='count')
speakers_stmt_count
```

Out[ ]:

	Speaker	count
0	18-percent-american-public	2
1	60-plus-association	2
2	AARP	1
3	Arizona-Citizens-Defense-League	2
4	Ballesteros	1
...	...	...
2905	yvette-mcgee-brown	3
2906	zack-space	1
2907	zell-miller	3
2908	zephyr-teachout	1
2909	zoe-lofgren	1

2910 rows × 2 columns

In [ ]:
```
truthful_stmts = speakers_stmts_by_truth_index[speakers_stmts_by_truth_index["Truth
false_stmts = speakers_stmts_by_truth_index[speakers_stmts_by_truth_index["Truth In
```

```
# Step 2: Calculate the counts for truthful and false statements
count_truthful = truthful_stmts.groupby("Speaker")["count"].sum().reset_index(name=
count_false = false_stmts.groupby("Speaker")["count"].sum().reset_index(name="False

# Step 3: Merge the counts to get the ratio
# First, merge count_truthful and count_false on the "Speaker" column
merged_counts = pd.merge(count_truthful, count_false, on="Speaker", how="outer")

# clean the data - wherever there's a NaN value, it should be 0
merged_counts['True Count'] = merged_counts['True Count'].fillna(0)
merged_counts['False Count'] = merged_counts['False Count'].fillna(0)

# Calculate the ratio of truthful statements to false statements
merged_counts["ratio"] = merged_counts["True Count"] / (merged_counts["False Count"

# # Now, add this ratio to speakers_stmt_count
# # Assuming speakers_stmt_count has a "Speaker" column that matches with the "Spea
# speakers_stmt_count = pd.merge(speakers_stmt_count, merged_counts[["Speaker", "ra

# # remove the speakers that have only a few statements
merged_counts.sort_values(by="ratio")
```

Out[ ]:

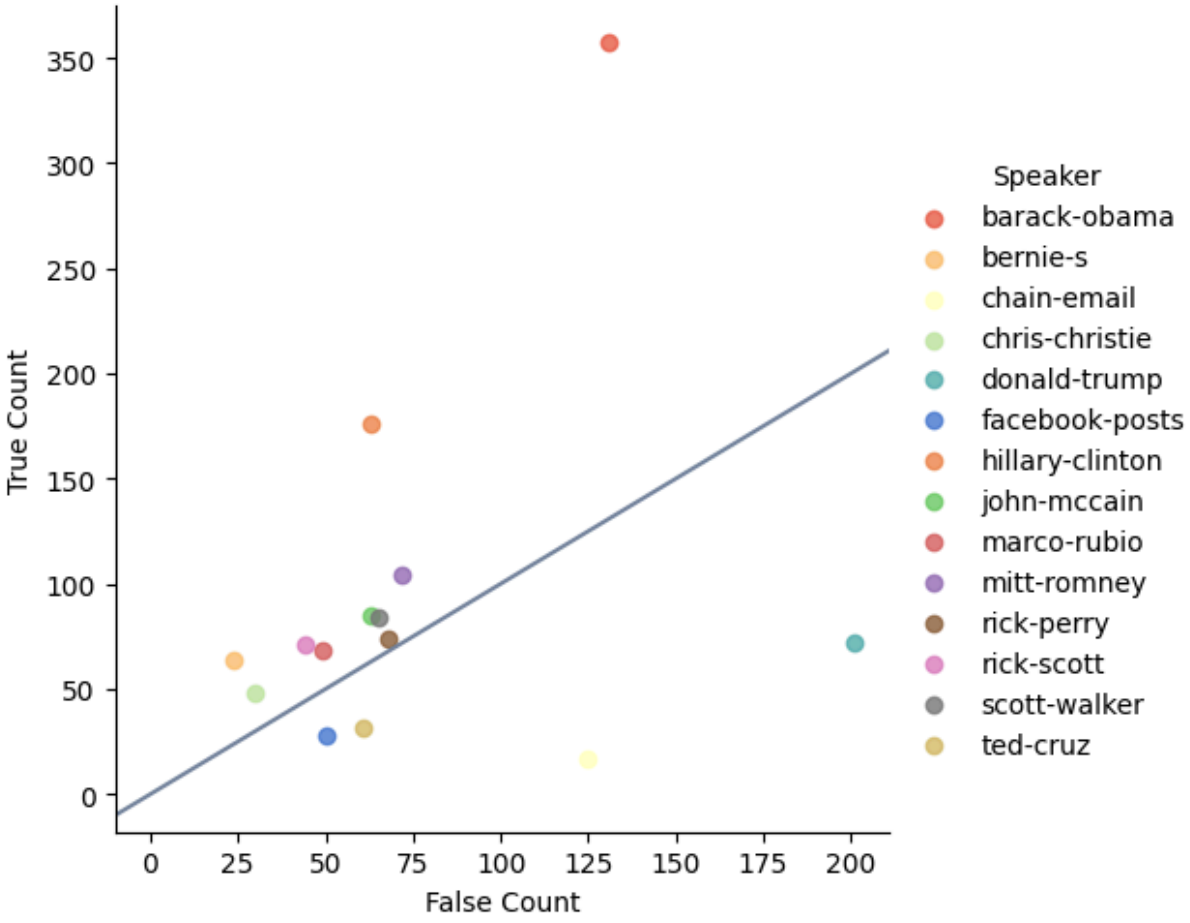
	Speaker	True Count	False Count	ratio
2909	zack-space	0.0	1.0	0.0
2317	georgia-state-senators	0.0	1.0	0.0
2316	georgia-state-road-and-tollway-authority	0.0	1.0	0.0
2315	georgia-gun-owners	0.0	1.0	0.0
2314	georgia-department-transportation	0.0	1.0	0.0
...	...	...	...	...
1843	texas-public-policy-foundation	1.0	0.0	1.0
1842	texas-petition	1.0	0.0	1.0
1841	texas-organizing-project	1.0	0.0	1.0
1839	texas-house-democratic-caucus	1.0	0.0	1.0
1454	pam-davidson	1.0	0.0	1.0

2910 rows × 4 columns

In [ ]:

```
# Create a merged palette with colors from spectral and muted color palettes
palette1 = sns.color_palette("Spectral", 5)
palette2 = sns.color_palette("muted", 10)
palette3 = sns.color_palette("pastel", 5)
merged_palette = palette1 + palette2 + palette3

palette = sns.color_palette(palette=merged_palette, n_colors=20)

# The x axis will be the ratio and the y axis will be the True Count. people who ar
```

```
seaborn_plot = sns.lmplot(data=merged_counts[(merged_counts["True Count"] + merged_
           x="False Count", y="True Count", palette=merged_palette, hue="Speaker")

plt.axline((0, 0), slope=1, color='#73849e', linestyle='-')

# The ratios of true to false statements made by individuals with more than 50 stat
plt.show()
```



Separating items with multiple subjects

Some of the subjects have multiple subjects that fall under multiple, already mentioned, subjects in the dataset. When we want to analyze how different subjects tend to have misinformation potentially spread by political parties, they must be separated.

- This will get only one subject per entry (e.g. a row with "alcohol, children, crime" as the subject will become 3 rows, so we can easily and more thoroughly group by subject)
- By not separating, an article that would be tied to the alcohol subject wouldn't be counted if its subject is "alcohol, children, and crime". It will be counted as its own subject.

In [ ]:
```
# This will separate each subject with multiple subjects into their own column (one
# Note: some of the subjects seem to have multiple subjects related to it
# Open the csv test file and label the columns (Names from the README File)
```

```
liar_dataset_exploded = pd.read_csv("train.tsv", sep='\t', header=None,
                            names=["ID", "Label", "Statement", "Subject", "Speaker",
                                   "State Info", "Party Affiliation", "Barely True C
                                   "Half True Count", "Mostly True Count", "Pants on

hierarchy = {
    'true': 1,
    'mostly-true': 1,
    'half-true': 1,
    'barely-true': 0,
    'false': 0,
    'pants-fire': 0
}

liar_dataset_exploded["Truth Index"] = liar_dataset_exploded["Label"].map(hierarchy

# Make it so that the liar data set is exploded by subject - i.e. Each row will hav
# The statements with multiple subjects will be repeated but with a different subje
liar_dataset_exploded["Subject"] = liar_dataset_exploded["Subject"].str.split(",")
liar_dataset_exploded = liar_dataset_exploded.explode("Subject")
liar_dataset_exploded['Subject'] = liar_dataset_exploded['Subject'].str.strip()
liar_dataset_exploded
```

Out[ ]:

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
0	2635.json	false	Says the Annies List political group supports ...	abortion	dwayne-bohac	State representative	Texas
1	10540.json	half-true	When did the decline of coal start? It started...	energy	scott-surovell	State delegate	Virginia
1	10540.json	half-true	When did the decline of coal start? It started...	history	scott-surovell	State delegate	Virginia
1	10540.json	half-true	When did the decline of coal start? It started...	job-accomplishments	scott-surovell	State delegate	Virginia
2	324.json	mostly-true	Hillary Clinton agrees with John McCain "by vo...	foreign-policy	barack-obama	President	Illinois
...	...	...	...	...	...	...	...
10237	3959.json	half-true	Says an alternative to Social Security that op...	social-security	herman-cain	NaN	Georgia
10238	2253.json	false	On lifting the U.S. Cuban embargo and allowing...	florida	jeff-greene	NaN	Florida
10238	2253.json	false	On lifting the U.S. Cuban embargo and allowing...	foreign-policy	jeff-greene	NaN	Florida

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
10239	1155.json	pants-fire	The Department of Veterans Affairs has a manua...	health-care	michael-steele	chairman of the Republican National Committee	Maryland
10239	1155.json	pants-fire	The Department of Veterans Affairs has a manua...	veterans	michael-steele	chairman of the Republican National Committee	Maryland

22205 rows × 15 columns

Studying the trends of statements of individuals based on their political party

Affiliations Objective: Looking into whether some parties have a trend of spreading more misinformation compared to others.

- There are more than 20 parties mentioned in the dataset. Some of these organizations have hundreds or thousands of statements made from individuals affiliated with those parties; however, there are some organizations that have only one or two statements. It'll be better to focus on just the top five parties with the most statements since they might have a larger influence compared to others.
- We can find the Political Parties with the most statements by "counting" the number of statements based on the "Political Affiliation" column and taking the top five. For the statements that do not fall in the top five, we will group them as "Other". There should be six different parties shown on the graph.

The reasoning to look at party affiliations is that if a certain affiliation has a trend of making mostly to completely false statements, we would know to treat the statements made by individuals of that party with more scrutiny when deciding to mark a statement as false.

What we can learn from the graph below:

The parties with the most statements made overall are Democrats and Republicans - The two prominent American political parties. Individuals with no party affiliation end up with the third highest count.

- Republicans tend to have a highest count of statements labeled as "pants-fire", "false", "barely true", or "half-true" compared to other parties' statements under the same label. Democrats seem to have a highest count of "mostly-true" statements compared to other political affiliations with the same label.
- Individuals with no affiliations seem to be distributed more evenly throughout the graph. However, the three labels with the highest number of unaffiliated speakers are "false", "half-true", and "mostly-true".
- Democrats' statements tend to be labeled as "mostly-true" and "half-true", or "true" across the graph, as compared to "barely-true", "false", and "pants on fire".

By looking at how statements' truthfulness compare between each party, we can identify where our model would need to focus on more when identifying fake news. If a certain party makes a higher number of false statements compared to other parties, then we could assume that there is a higher chance of that party making false statements in the future.

In [ ]:
```
# get the entries where there is a political affiliation associated with the indivi
liar_subset = liar[
    (liar["Party Affiliation"].isnull() == False)]
liar_subset.groupby("Party Affiliation").size().reset_index(name="count")


# Step 1: Identify the top 4 party affiliations based on count
top_affiliations = liar_subset["Party Affiliation"].value_counts().index[:5]

# Step 2: Group the rest into "Other" and map every entry based on whether they fal
liar_subset['Party Affiliation'] = liar_subset['Party Affiliation'].apply(lambda x:

# filter by party, label, and truth index, and count the values that show up for ea
filtered_liar_dataset = liar_subset.groupby(['Party Affiliation',
                                             'Label', "Truth Index"]).size().reset_

# filtered_liar_dataset
# How do certain subjects and their validity relate to one another
color_palette = {'republican': 'red', 'democrat': 'blue', 'none': 'purple', 'indepe
desired_order = ['true', 'mostly-true', 'half-true', 'barely-true', 'false', 'pants

liar_barplot = sns.barplot(data=filtered_liar_dataset, x="Label", y="count", palett
liar_barplot.set_title("Validity of Statements Based on Political Statements on Sub
liar_barplot
liar_subset
```

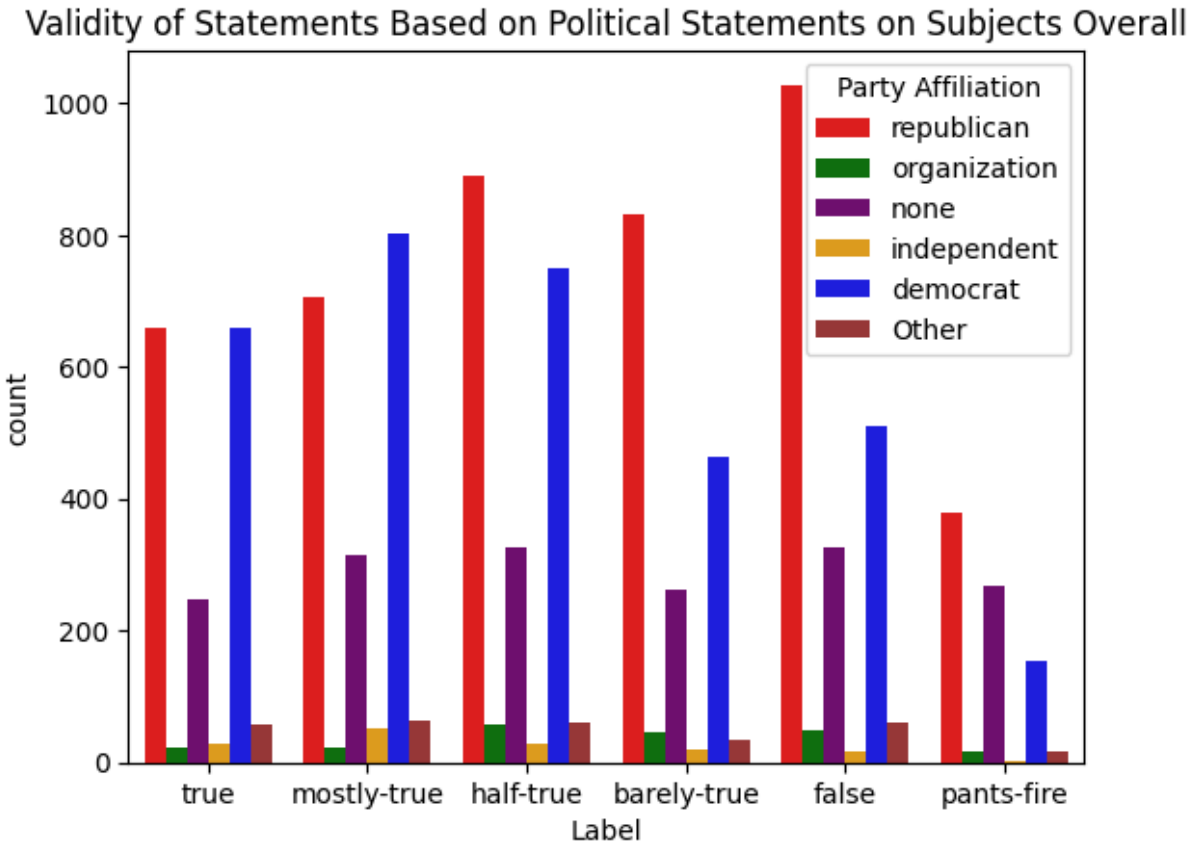
```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\2862982539.py:11: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  liar_subset['Party Affiliation'] = liar_subset['Party Affiliation'].apply(lambda
x: x if (x in top_affiliations) else 'Other')
```

Out[ ]:

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
0	2635.json	false	Says the Annies List political group supports ...	abortion	dwayne-bohac	State representative	Texas
1	10540.json	half-true	When did the decline of coal start? It started...	energy,history,job-accomplishments	scott-surovell	State delegate	Virginia
2	324.json	mostly-true	Hillary Clinton agrees with John McCain "by vo...	foreign-policy	barack-obama	President	Illinois
3	1123.json	false	Health care reform legislation is likely to ma...	health-care	blog-posting	NaN	NaN
4	9028.json	half-true	The economic turnaround started at the end of ...	economy,jobs	charlie-crist	NaN	Florida
...	...	...	...	...	...	...	...
10235	5473.json	mostly-true	There are a larger number of shark attacks in ...	animals,elections	aclu-florida	NaN	Florida
10236	3408.json	mostly-true	Democrats have now become the party of the [At...	elections	alan-powell	NaN	Georgia
10237	3959.json	half-true	Says an alternative to Social Security that op...	retirement,social-security	herman-cain	NaN	Georgia

	ID	Label	Statement	Subject	Speaker	Speaker Job Title	State Info
10238	2253.json	false	On lifting the U.S. Cuban embargo and allowing...	florida,foreign-policy	jeff-greene	NaN	Florida
10239	1155.json	pants-fire	The Department of Veterans Affairs has a manua...	health-care,veterans	michael-steele	chairman of the Republican National Committee	Marylanc

10238 rows × 15 columns



Among certain subjects, which party spreads the most misinformation?

What we can learn from the graphs below:

The parties with the most statements made overall are Democrats and Republicans - The two most prominent American political parties. Individuals with no party affiliation had the third highest count.

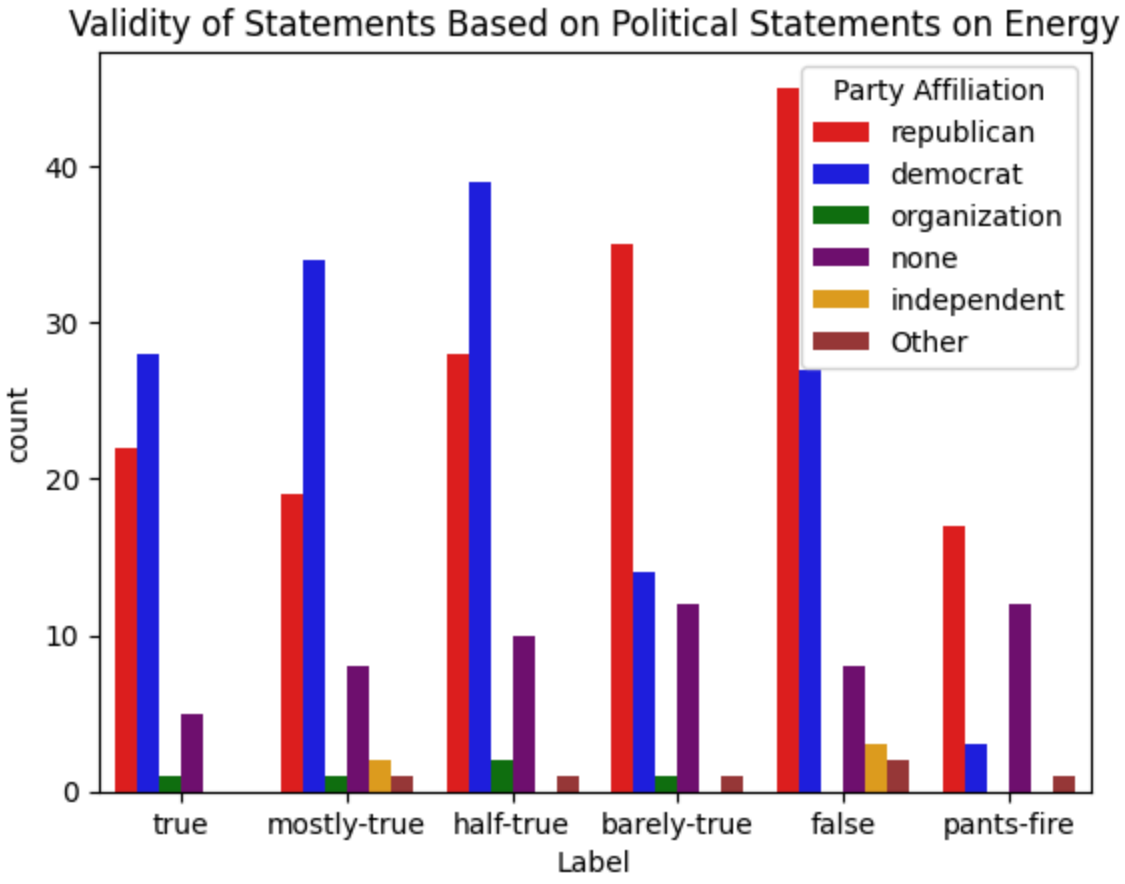
- Republicans tend to have a highest count of statements labeled as "pants-fire", "false", "barely true", or "half-true" compared to other parties' statements under the same label. Democrats seem to have a highest count of "mostly-true" statements compared to other political affiliations with the same label.
- Individuals with no affiliations seem to be distributed more evenly throughout the graph. However, the three labels with the highest number of unaffiliated speakers are "false", "half-true", and "mostly-true".
- Democrats' statements tend to be labeled as "mostly-true" and "half-true", or "true" across the graph, as compared to "barely-true", "false", and "pants on fire".
- By looking at how statements' truthfulness compare between each party, we can identify where our model would need to focus on more when identifying fake news. If a certain party makes a higher number of false statements compared to other parties, then we could assume that there is a higher chance of that party making false statements in the future.

In [ ]:
```
# find individuals where the party affiliation is not null
liar_subset = liar_dataset_exploded [
    (liar_dataset_exploded["Party Affiliation"].isnull() == False)]
# energy_liar_subset

# find the top 5 affiliations in this dataset
top_affiliations = liar_subset["Party Affiliation"].value_counts().index[:5]   # th
top_affiliations

# define the color palette in the top affiliations
color_palette = {'republican': 'red', 'democrat': 'blue', 'none': 'purple', 'indepe
```

In [ ]:
```
energy_liar_subset = liar_subset[liar_subset["Subject"] == "energy"]
# # Step 2: Group the rest into "Other"
energy_liar_subset['Party Affiliation'] = energy_liar_subset['Party Affiliation'].a

# # filter by subject, party, label, and truth index
filtered_liar_dataset = energy_liar_subset.groupby(['Subject', 'Party Affiliation',

# energy_liar_dataset = filtered_liar_dataset[filtered_liar_dataset["Subject"] == "
# # How do certain subjects and their validity relate to one another
desired_order = ['true', 'mostly-true', 'half-true', 'barely-true', 'false', 'pants

liar_barplot = sns.barplot(data=filtered_liar_dataset, x="Label", y="count", palett
liar_barplot.set_title("Validity of Statements Based on Political Statements on Ene
liar_barplot
```

```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\3031223662.py:3: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  energy_liar_subset['Party Affiliation'] = energy_liar_subset['Party Affiliation'].
apply(lambda x: x if (x in top_affiliations) else 'Other')
```

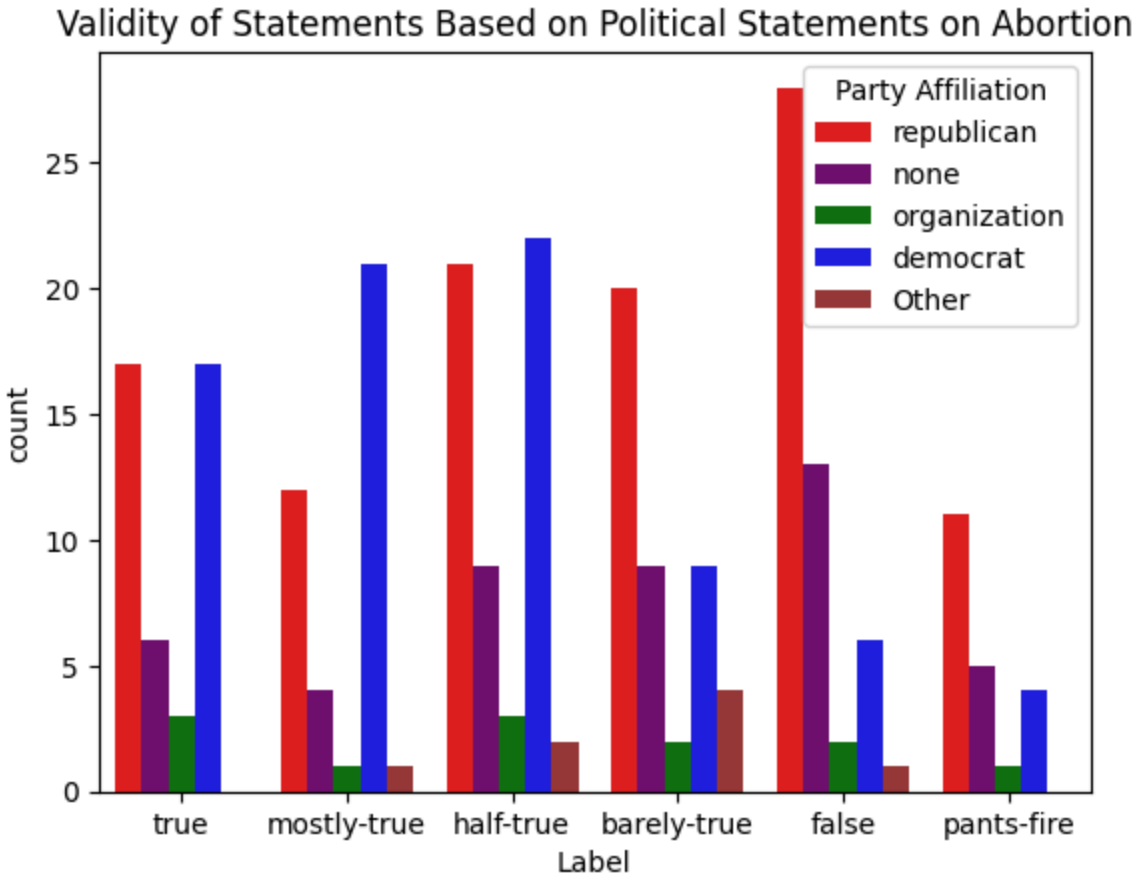
Out[ ]: <Axes: title={'center': 'Validity of Statements Based on Political Statements on E
nergy'}, xlabel='Label', ylabel='count'>



In [ ]:
```
abortion_liar_subset = liar_dataset_exploded[liar_dataset_exploded["Subject"] == "a

# Step 2: Group the rest into "Other"
abortion_liar_subset['Party Affiliation'] = abortion_liar_subset['Party Affiliation

# filter by subject, party, label, and truth index
filtered_liar_dataset = abortion_liar_subset.groupby(['Subject', 'Party Affiliation

# # # How do certain subjects and their validity relate to one another
desired_order = ['true', 'mostly-true', 'half-true', 'barely-true', 'false', 'pants
liar_barplot = sns.barplot(data=filtered_liar_dataset, x="Label", y="count", palett
liar_barplot.set_title("Validity of Statements Based on Political Statements on Abo
liar_barplot
```

```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\438217718.py:4: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  abortion_liar_subset['Party Affiliation'] = abortion_liar_subset['Party Affiliatio
n'].apply(lambda x: x if (x in top_affiliations) else 'Other')
```

Out[ ]: <Axes: title={'center': 'Validity of Statements Based on Political Statements on A
bortion'}, xlabel='Label', ylabel='count'>

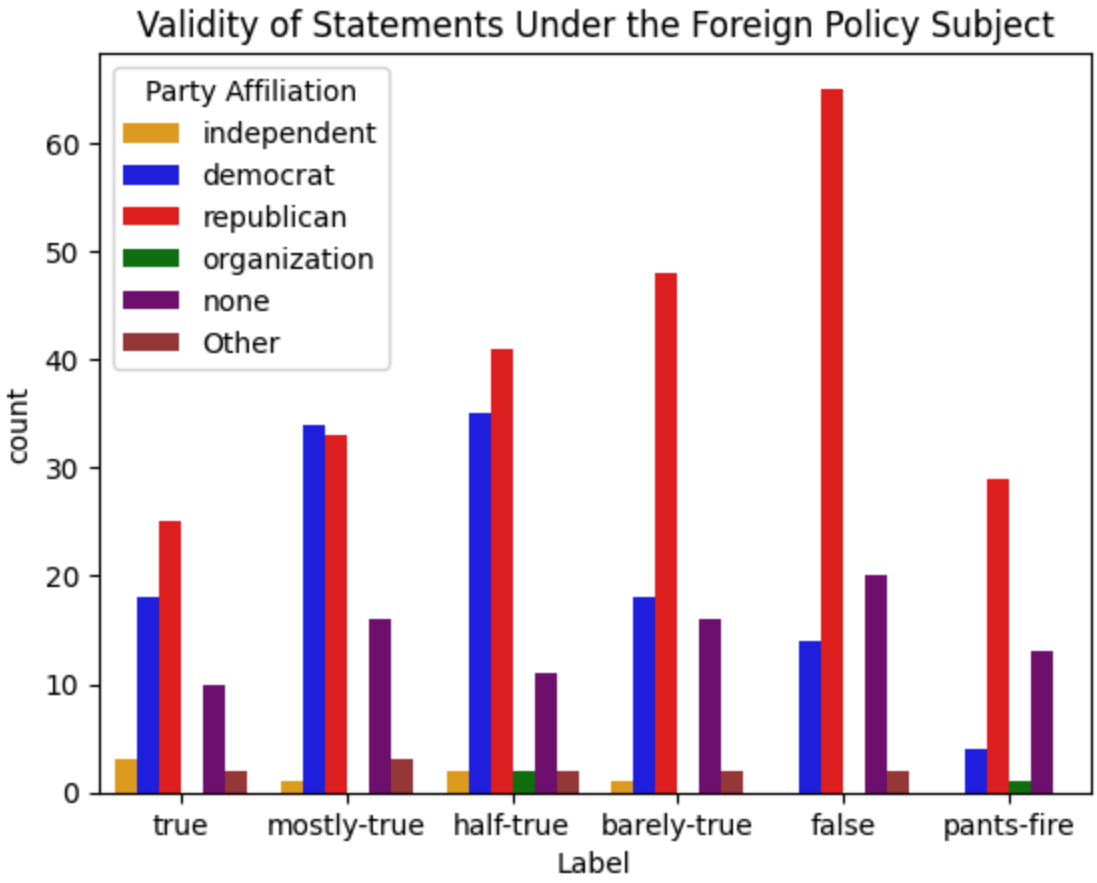


In [ ]:
```
# look at how the foreign policy subject statements match up between each party
foreign_policy_liar_subset = liar_dataset_exploded[liar_dataset_exploded["Subject"]

# Step 1: Identify the top 4 party affiliations based on count in this subset
# top_affiliations = alcohol_liar_subset["Party Affiliation"].value_counts().index[

# # # Step 2: Group the rest into "Other"
foreign_policy_liar_subset['Party Affiliation'] = foreign_policy_liar_subset['Party

# # # filter by subject, party, label, and truth index for the graph, and create a
filtered_liar_dataset = foreign_policy_liar_subset.groupby(['Subject', 'Party Affil

# # How do certain subjects and their validity relate to one another
desired_order = ['true', 'mostly-true', 'half-true', 'barely-true', 'false', 'pants
liar_barplot = sns.barplot(data=filtered_liar_dataset, x="Label", y="count", palett
```

```
liar_barplot.set_title("Validity of Statements Under the Foreign Policy Subject")
liar_barplot
```

```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\1885585244.py:8: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  foreign_policy_liar_subset['Party Affiliation'] = foreign_policy_liar_subset['Part
y Affiliation'].apply(lambda x: x if x in top_affiliations else 'Other')
```

Out[ ]: <Axes: title={'center': 'Validity of Statements Under the Foreign Policy Subject'}, xlabel='Label', ylabel='count'>



Assuming statements could either be only true or only false, how often do political parties make false statements?

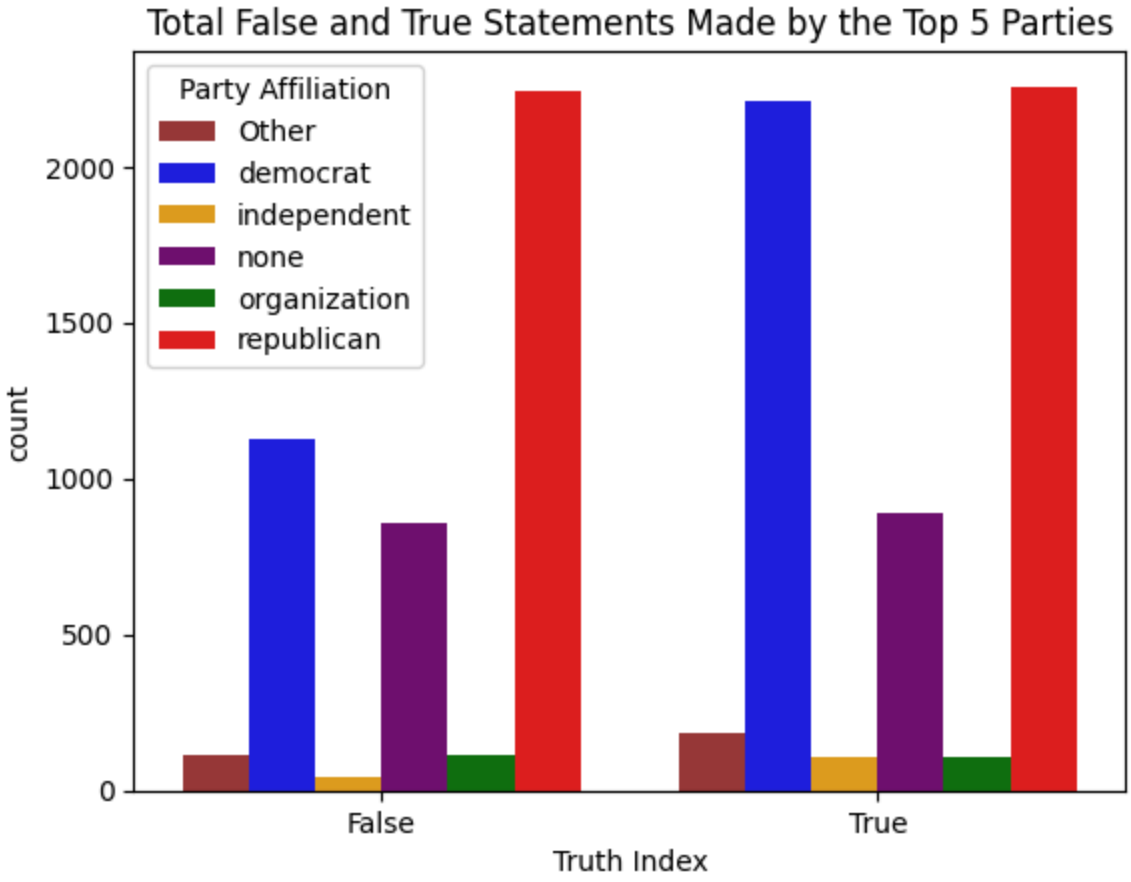
- We can assume that either statements can be only true or false.
- When grouping all the statements as "true" or "false" (true -> half-true = true and pants on fire -> barely true = false), we can look at how each of these parties line up in terms of trustworthiness of their statements.

In [ ]:
```
true_false_liar_subset = liar

# find the columns in top_affiliations
true_false_liar_subset['Party Affiliation'] = true_false_liar_subset['Party Affilia
true_false_liar_subset
true_false_liar_subset_grouped = liar.groupby(['Party Affiliation', "Truth Index"])

total_per_each_party = liar.groupby(['Party Affiliation']).size().reset_index(name=



# # How do certain subjects and their validity relate to one another
liar_barplot = sns.barplot(data=true_false_liar_subset_grouped, x="Truth Index", y=

# we will assume here that a statement can either be only false or true
liar_barplot.set_title("Total False and True Statements Made by the Top 5 Parties")
liar_barplot.set_xticklabels(['False', 'True'])
liar_barplot
true_false_liar_subset_grouped
# total_per_each_party
```

```
C:\Users\Taha\AppData\Local\Temp\ipykernel_24496\2127651911.py:17: UserWarning: set_
ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks
() or using a FixedLocator.
  liar_barplot.set_xticklabels(['False', 'True'])
```

Out[ ]:

	Party Affiliation	Truth Index	count
1	Other	1	184
3	democrat	1	2209
5	independent	1	107
7	none	1	888
9	organization	1	107
11	republican	1	2257
0	Other	0	113
2	democrat	0	1127
4	independent	0	40
6	none	0	856
8	organization	0	112
10	republican	0	2240


Total False and True Statements Made by the Top 5 Parties
Party Affiliation
Other
democrat
independent
none
organization
republican
2000
1500
1000
500
0
count
False
True
Truth Index

Analysis of the results from above

There seems to be a higher count of Democrats and Republicans overall making statements; however, we can still see that there is a significant difference of false statements and true statements for Democrats, Independents, and "Others", showing that there is at least a slight higher chance that a statement made by one of these groups is true. Republicans and unaffiliated people, on the other hand, seem to have a trend of having a roughly almost equal number of false and true statements in the past. The only affiliation with more false statements than true statements made overall were organizations.

Can we look at words that are used more often in a certain subject compared to others?

- Can we build a Table or a scatterplot or Pie Chart for showing what words are used in different subjects we looked at?
- TF IDF on certain words being used? and where they might fall in validity
- How often do certain individuals appear in telling the truth among others? make clusters to find out on a scatterplot

Machine Learning Analysis

When it comes to picking the best features to train the models on, ideally we'll want the features that are filled in the most.

In [ ]:
```
liar.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10240 entries, 0 to 10239
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10240 non-null  object
 1   Label                10240 non-null  object
 2   Statement            10240 non-null  object
 3   Subject              10238 non-null  object
 4   Speaker              10238 non-null  object
 5   Speaker Job Title    7342 non-null   object
 6   State Info           8030 non-null   object
 7   Party Affiliation    10240 non-null  object
 8   Barely True Count    10238 non-null  float64
 9   False Count          10238 non-null  float64
 10  Half True Count      10238 non-null  float64
 11  Mostly True Count    10238 non-null  float64
 12  Pants on Fire Count  10238 non-null  float64
 13  Context/Location     10138 non-null  object
 14  Truth Index          10240 non-null  int64
dtypes: float64(5), int64(1), object(9)
memory usage: 1.2+ MB
```

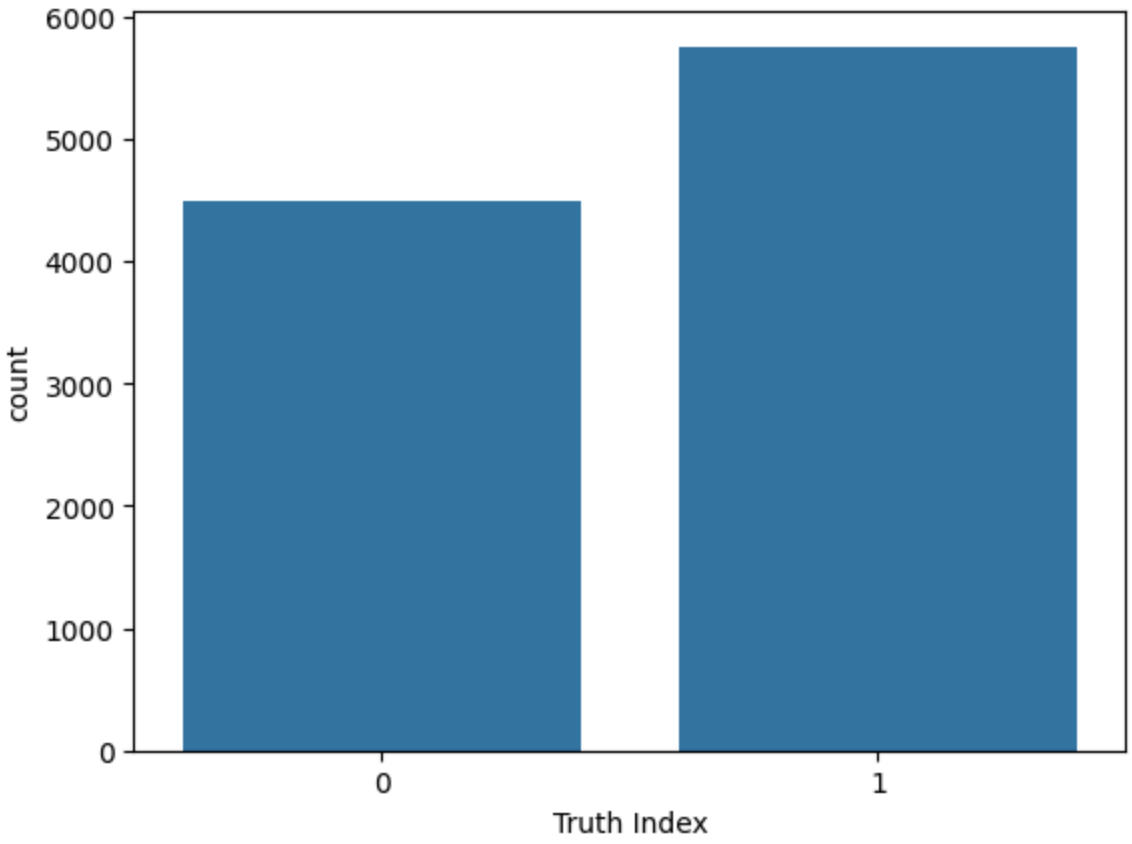
One final notable bit is that the dataset contains more true statements than false statements.

In [ ]:
```
liar['Truth Index'].value_counts()
```

Out[ ]:
```
Truth Index
1    5752
0    4488
Name: count, dtype: int64
```

In [ ]:
```
sns.countplot(data=liar, x='Truth Index')
```

Out[ ]:
```
<Axes: xlabel='Truth Index', ylabel='count'>
```



Data Pre-Training

Here I define the functions for the lemmatizer technique used in Homework 3 to truncate words into their root word (Ex: "doing" -> "do")

In [ ]:

```
import nltk

nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
stopwords = nltk.corpus.stopwords.words('english')

posMapping = {
    "N":'n',
    "V":'v',
    "J":'a',
    "R":'r'
}

def remove_url(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    res = re.sub(url_pattern, '', text)

    return res
```

```
def process(text, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()):
    text = remove_url(text)
    text = text.replace("'s", "")
    text = text.replace("'", "")
    for i in string.punctuation:
        text = text.replace(i, " ")

    tokenized = text.split()
    tags = nltk.pos_tag(tokenized)
    res = []
    for i in tags:
        if i[1][0] in posMapping:
            res.append(lemmatizer.lemmatize(i[0].lower(), pos=posMapping[i[1][0]]))
        else:
            res.append(lemmatizer.lemmatize(i[0].lower(), pos='n'))

    return res

def process_all(df, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()):
    df['Combined'] = df['Combined'].apply(lambda x: process(x, lemmatizer))
    return df
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Taha\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Taha\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Taha\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

First we must choose the features to train the model on. For this dataset, we'll use the Statement, Party Affiliation, and Subject features since they almost all filled in. If we included some other features such as Speaker we would end up with less entries to train the model on. We then use the functions defined above to lemmatize the words in the combined column.

Next, we split the datasets into two sets of training and testing sets for each dataset we have.

In [ ]:
```
features = ["Statement", "Party Affiliation", "Subject"]

liar["Combined"] = liar[features].apply(lambda x: ' '.join(x.astype(str)), axis=1)
test["Combined"] = test[features].apply(lambda x: ' '.join(x.astype(str)), axis=1)

processed_liar = process_all(liar, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()
processed_test = process_all(test, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()

x_train = processed_liar["Combined"]
```

```
y_train = processed_liar["Truth Index"]

x_test = processed_test["Combined"]
y_test = processed_test["Truth Index"]
```

Here, I apply TF-IDF vectorization to the training and test datasets from the liar set. This process not only uncovers the most relevant features of the data, but also allows the model to be trained by the vectorized data. I'll also use stopwords to ignore some irrelevant words.

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler

vectorization = TfidfVectorizer(min_df=2, tokenizer=(lambda x : x), stop_words=sort
xv_train = vectorization.fit_transform(x_train)
xv_test = vectorization.transform(x_test)
```

```
c:\Python311\Lib\site-packages\sklearn\feature_extraction\text.py:525: UserWarning:
The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\feature_extraction\text.py:408: UserWarning:
Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop wor
ds generated tokens ["'", 'b', 'c', 'e', 'f', 'g', 'h', 'j', 'l', 'n', 'p', 'r',
'u', 'v', 'w'] not in stop_words.
  warnings.warn(
```

Now we scale the training and test datasets to center the data and achieve a normal distribution.

In [ ]:
```
scaler = StandardScaler(with_mean=False)
xv_train = scaler.fit_transform(xv_train)
xv_test = scaler.transform(xv_test)
```

Printing the resulting features from the vectorization.

In [ ]:
```
feature_names = vectorization.get_feature_names_out()
print(feature_names)
```

```
['0' '00' '000' ... 'zombie' 'zone' '\x90']
```

Various Model Training

There are four main machine learning models I'll test:

1. Support Vector Machines (SVM) Classification
2. K-Nearest Neighbors (KNN) Classification
3. Decision Trees
4. XGBoost

Each of the models will use the TF-IDF vectorized training data as well as the training data for the output class to fit the data. Then, the model will continue on to train using the fitted data and the TF-IDF vectorized testing data. Finally, the model will be evaluated using the predicted classifications from the training and the testing data. The precision, recall and f1-scores will be printed, as well as the accuracy of the model. Furthermore, to supplement the precision and recall, a graph the confusion matrix for each model will be shown.

Baseline comparison

The most basic way to compare our trained models is to come up with a baseline model that is simple and can serve as a benchmark for the success of the models. For this particular dataset, the baseline comparison would be a model that always says that the test statement is true. As mentioned earlier, there are far more true statements than false statements in the dataset. If this model was used on the training data, it would have an accuracy of about 56%.

In that instance, our models would have to be able to achieve an accuracy significantly greater than this baseline of 56% to be seen as a viable solution.

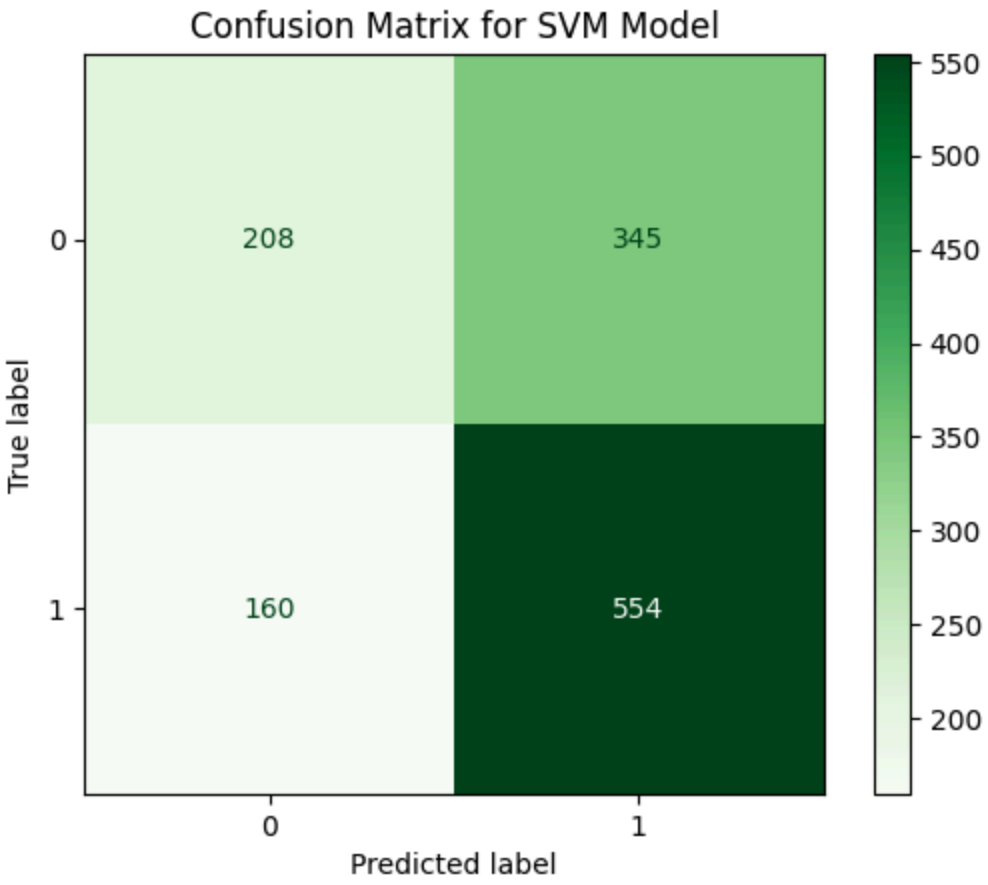
SVM Classification

In [ ]:
```
#Training SVM model
svm_model = SVC(kernel='rbf', C=1.0, random_state=42,probability=True)
svm_model.fit(xv_train, y_train)

y_pred = svm_model.predict(xv_test)

#Model evaluation
accuracy_1 = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_1)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.601420678768745
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.38      0.45       553
           1       0.62      0.78      0.69       714

    accuracy                           0.60      1267
   macro avg       0.59      0.58      0.57      1267
weighted avg       0.59      0.60      0.58      1267
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)

# Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Greens')
```

```
plt.title('Confusion Matrix for SVM Model')
plt.show()
```

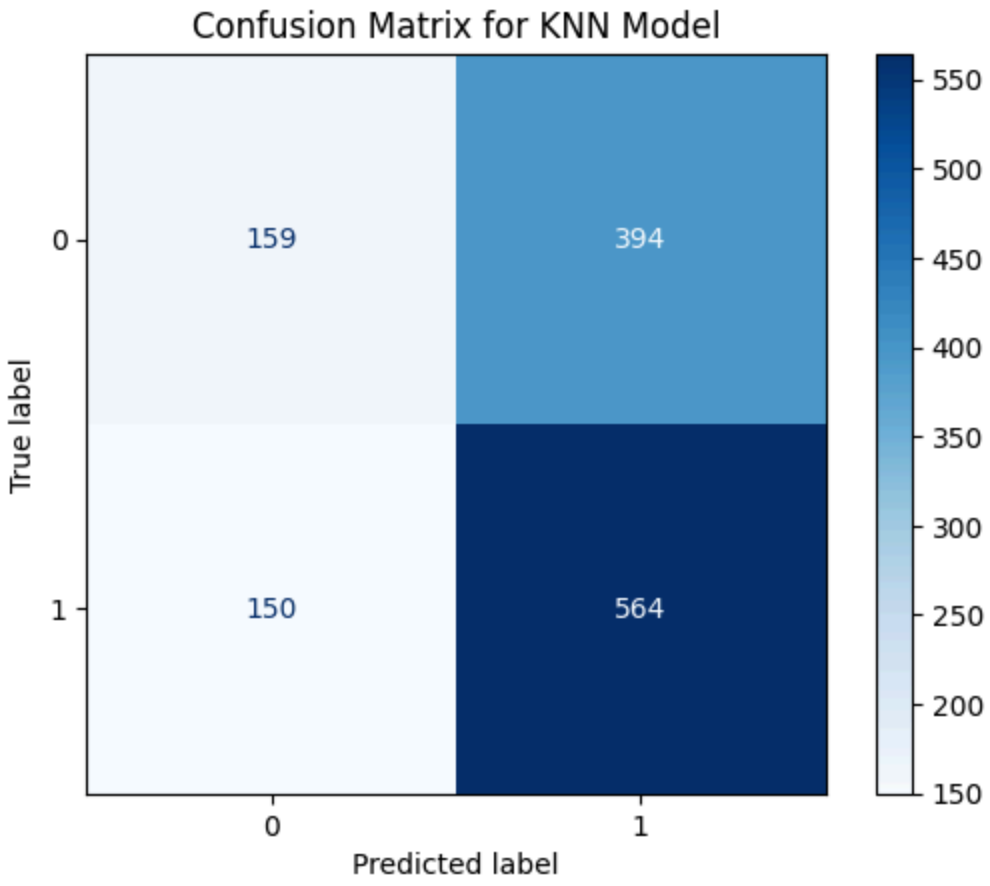


KNN Classification

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
# Train KNN classifier
k = 5  # number of neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(xv_train, y_train)
y_pred = knn_classifier.predict(xv_test)
#Model evaluation
accuracy_2 = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_2)
print("Classification Report:")
# Additional evaluation metrics
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5706393054459353
Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.29      0.37       553
           1       0.59      0.79      0.67       714

    accuracy                           0.57      1267
   macro avg       0.55      0.54      0.52      1267
weighted avg       0.56      0.57      0.54      1267
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)

# Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix for KNN Model')
plt.show()
```



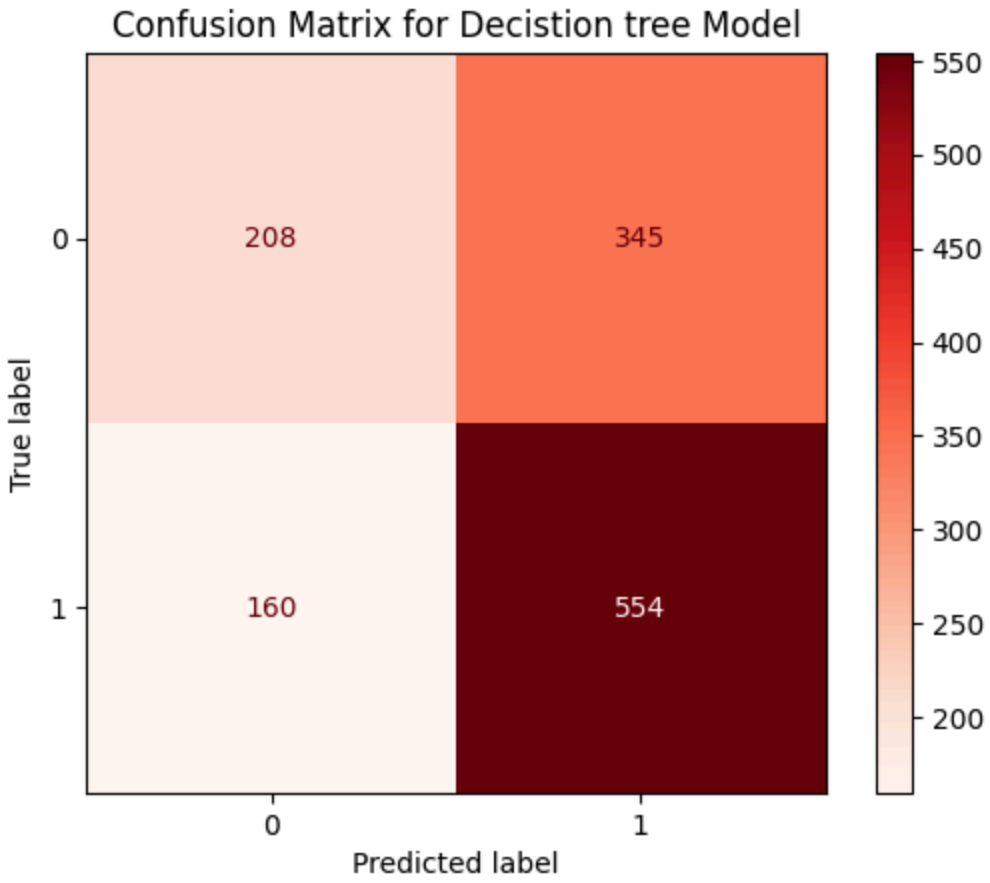
Decision Trees

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()
DT.fit(xv_train, y_train)
y_pred = DT.predict(xv_test)
```

```
# Step 5: Model evaluation
accuracy_3 = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_3)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5509076558800315
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.49      0.49       553
           1       0.60      0.60      0.60       714

    accuracy                           0.55      1267
   macro avg       0.54      0.54      0.54      1267
weighted avg       0.55      0.55      0.55      1267
```

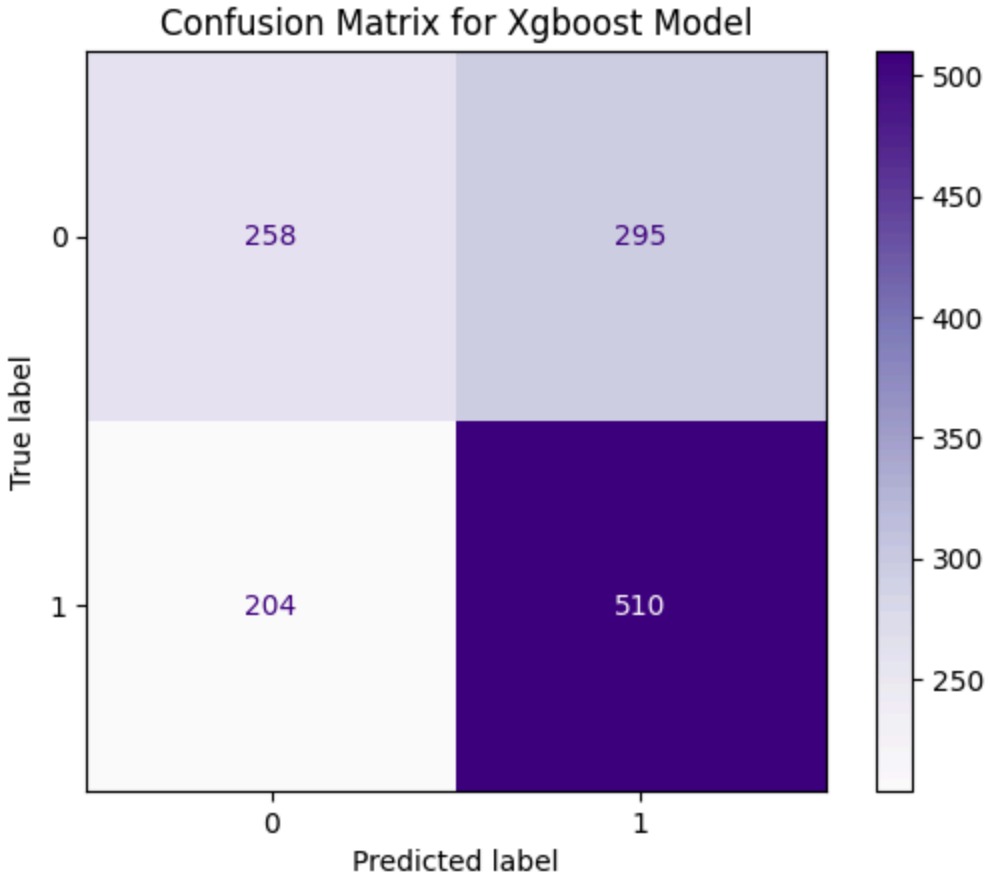
In [ ]:
```
cm = confusion_matrix(y_test, y_pred)

# Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Reds')
plt.title('Confusion Matrix for Decistion tree Model')
plt.show()
```



XGBoost

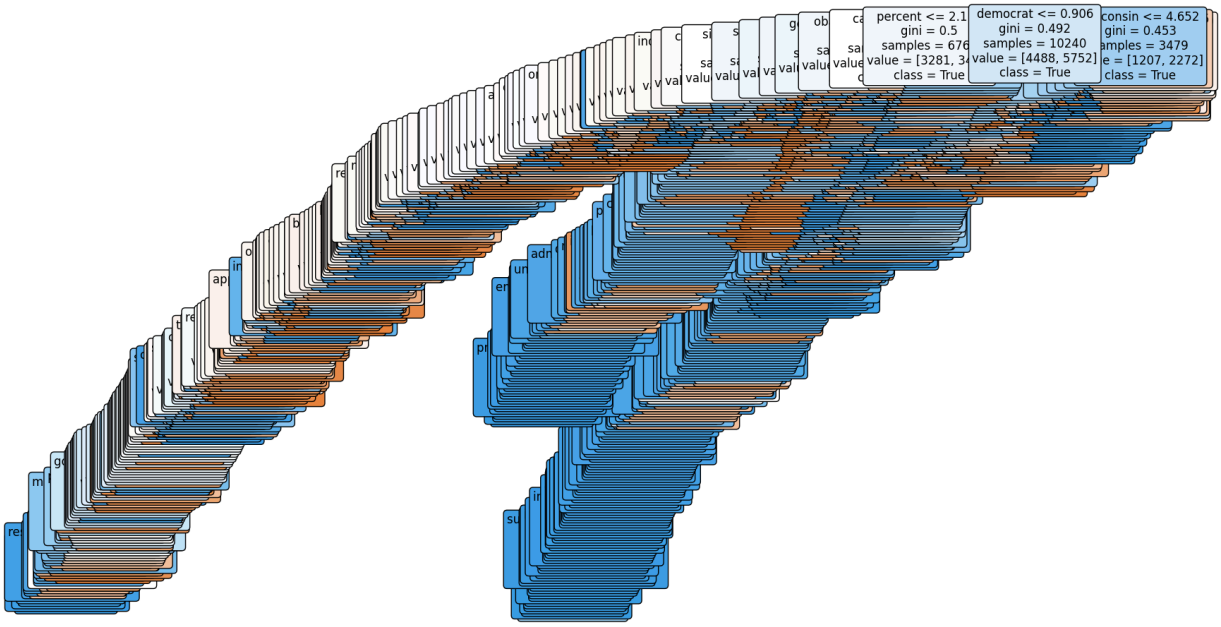
In [ ]:
```
import xgboost as xgb


xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(xv_train, y_train)

y_pred = xgb_model.predict(xv_test)

accuracy_4 = accuracy_score(y_test, y_pred)
print(accuracy_4)
print(classification_report(y_test, y_pred))


cm = confusion_matrix(y_test, y_pred)

# Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Purples')
plt.title('Confusion Matrix for Xgboost Model')
plt.show()
```

```
0.606156274664562
              precision    recall  f1-score   support

           0       0.56      0.47      0.51       553
           1       0.63      0.71      0.67       714

    accuracy                           0.61      1267
   macro avg       0.60      0.59      0.59      1267
weighted avg       0.60      0.61      0.60      1267
```



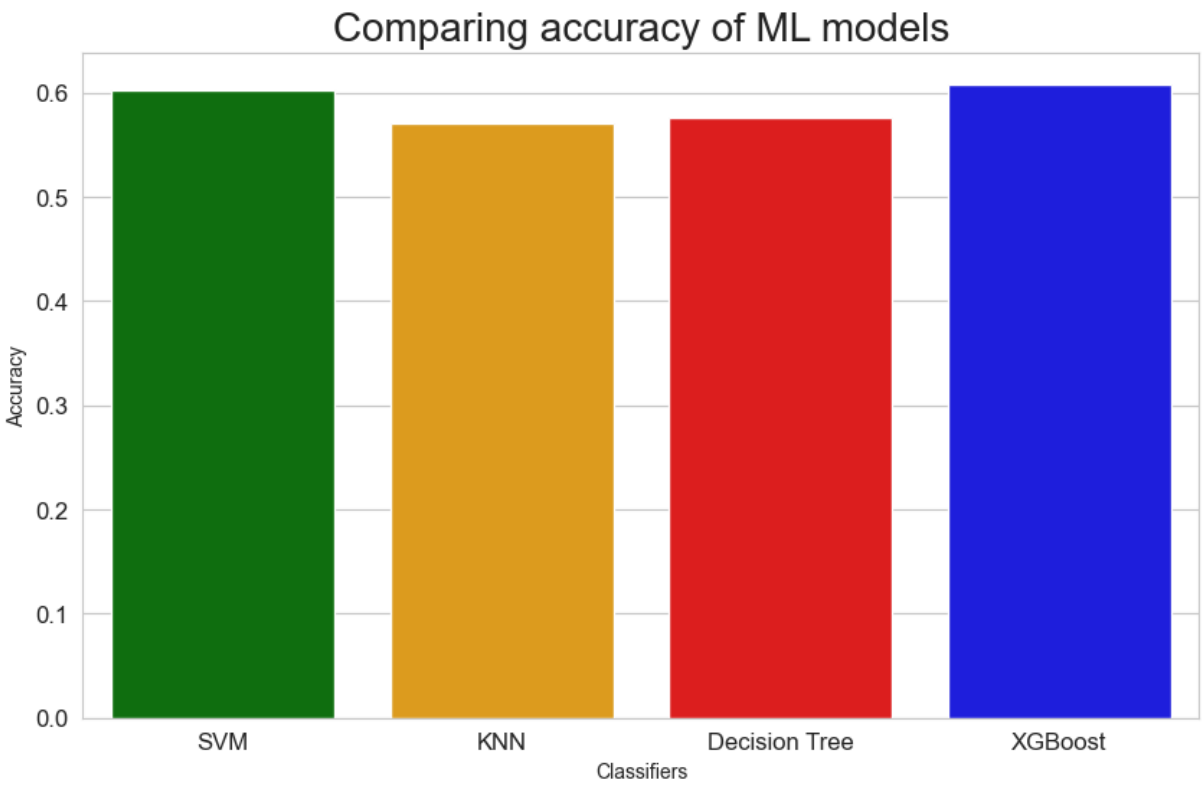
Plotting the decision tree where blue = true, and orange = false.

In [ ]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(DT, filled=True, feature_names = feature_names, class_names=['False', 'Tr
plt.show()
```



Plotted below is the accuracy rating for each of the models.

In [ ]:
```
# Models accuracy scores
accuracies = {
    "SVM": accuracy_1,
    "KNN": accuracy_2,
    "Decision Tree": accuracy_3,
    "XGBoost": accuracy_4
}

colors = ["green", "orange", "red", "blue"]

# Plotting
plt.figure(figsize=(10, 6))
sns.set_style("whitegrid")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors,
plt.title('Comparing accuracy of ML models', fontsize=20)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.ylabel("Accuracy")
plt.xlabel("Classifiers")

plt.show()
```

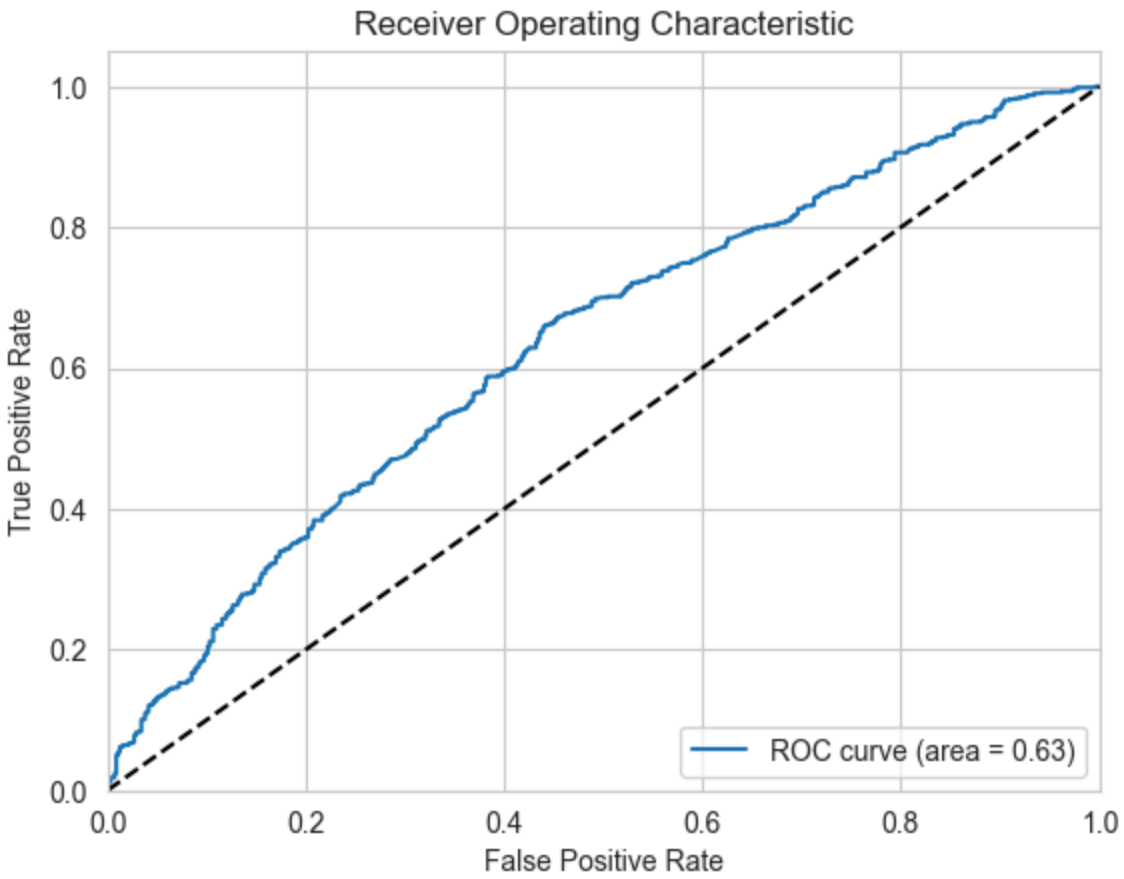


Plotted below is the ROC curve for the XGBoost model.

In [ ]:
```
from sklearn.metrics import  roc_auc_score, roc_curve, auc
y_proba = xgb_model.predict_proba(xv_test)[:, 1]  # Probability estimates for the p
```

```
# Calculate AUC
roc_auc = roc_auc_score(y_test, y_proba)
print(f'AUC-ROC: {roc_auc}')

# To plot the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

```
AUC-ROC: 0.6337902249507398
```



Interpretation of results

In terms of accuracy, as observed, the XGBoost model outperformed the other three models, achieving an accuracy of 60.9%. Despite efforts to improve the model's accuracy through hyperparameter tuning, the accuracy remained within a similar range of the baseline. For XGBoost, the performance metrics for the positive class were generally okay, with a precision of 63%, recall of 72%, and a balance between precision and recall resulting in an F1-score of

0.68. Conversely, Class 0 (false) displayed a lower F1-score of 0.51. Another performance metric calculated was the AUC-ROC score, which yielded a score of 63%. This indicates that the XGBoost model has potential but still requires improvement.

The SVM model came in second with an accuracy of 60.3%. It arguably was the best at discerining true positives, having the highest F1-score in the positive class with 0.69. However, it's biggest shortcoming came with correctly identifying false statements. The KNN model essentially followed the same pattern as the SVM model in detecting the true class and struggling with the false class. Lastly, the Decision Tree model was the most balanced of all the ones tested. It correctly identified false statements better than every other model. However, it was also the worst at identifying true statements, and ended up with one of the lower scores.

The biggest takeaway from these scores is that it's clear that generally, the trained models are much worse at identifying false statements than true statements, which isn't much better. With a maximum accuracy of around 60%, and comparing that to our baseline, it could be better. To add, some of the lowest accuracy ratings witnessed throughout testing are only slightly the 56% baseline, which is very alarming.

One last piece of insight, I curious as to what the runtime duration of each of the machine learning models is. Through multiple trials of each model, we found that XGBoost, Decision Trees and KNN classification run in a reasonable amount of time at 1.37s, 1.80s and 0.85s on average respectively. However, the SVM model takes a considerably longer amount of time to complete. On average, the model takes 51.2s to complete fitting, training, and evaluation, as the others did too. Although the SVM model performed better compared to some of the other models tested, the difference is accuracy isn't massive, unlike the difference in runtime.

Reflection

Returning to the initial objectives

- **Are certain individuals or entities more inclined to propagate misinformation?**

  Going back to the analysis of the data--at least when concerning political entities--there is a clear trend of the democratic party being the most trustworthy when it comes to giving statements that are at least somewhat true. For every other political entity, they tend to have just as many truths as lies.

- **Is misinformation more rampant in discussions about certain subjects, and does political affiliation influence the spread of false information??**

  At least when looking at the total counts of true/false statements for each subject matter, there isn't a significant difference in true or false counts across different

subjects. The story does change when taking political affiliation into account. However, those findings aren't much different than those mentioned above.

- **Is it possible to reliably detect whether a politically charged statement is true or false using models trained on similar statements?**

  Unfortunately, from the results of testing four different models, I cannot say that it's possible. That is, when trained on the LIAR dataset. As mentioned in the interpretation of testing results, all of the models struggled to identify false statements, and instead were much better at identifying true ones. This goes back to the fact that the dataset contains more true statements than false ones. I believe the models would have benefitted greatly from a more balanced distribution of true/false statements.

Conclusion

The ability to identify misinformation in the media and on the internet is becoming an increasingly crucial skill to have in this day and age. It's also rare to see people genuinely uphold the age old "Don't believe everything you see on the internet". Which is why the necessity for a tool such as this is so important. Although I couldn't create a reliable model for detecting misinformation, I absolutely believe that it can be accomplished. However, I don't feel that the LIAR dataset is good enough to produce that model. Some things can't reasonably be controlled, such as the possible biases in the dataset, since it would be virtually impossible to have no bias in such a dataset. Seeing past that, there are a multitude of problems with the dataset. Many entries have missing columns, so much so that some features are not usable in the training process. The six true/false classifiers are unnecessary--though I understand the benefit of more scrutiny in individual statements. However, the issue of having more truthful statements than false statements is likely the biggest shortcoming. All these factors lead me to believe that the LIAR dataset wouldn't produce a reliable model regardless of how many optimizations and hyperparameters are used.