PSTAT 135 Final Project

Aapthi Nagesh, Alex Roginski, Yuchen Wu, Nicholas Axl Andrian

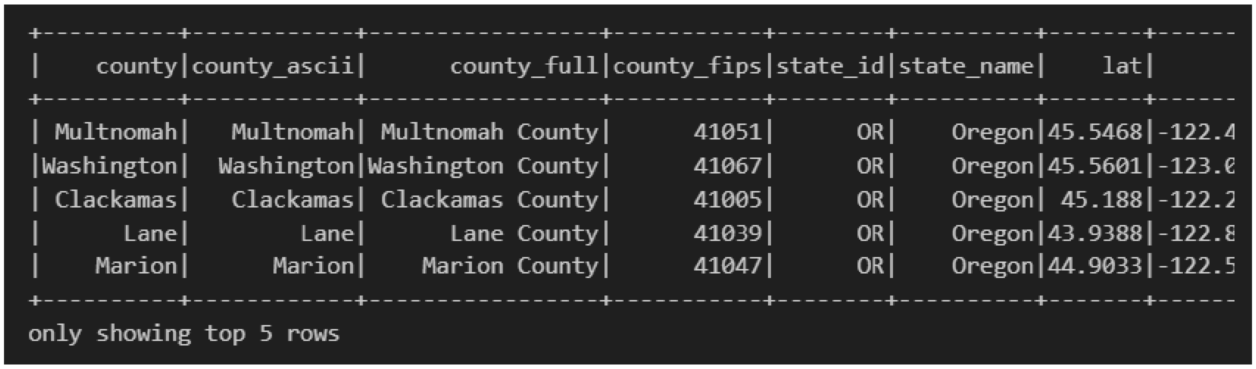
1. Introduction

The voting system in any country is crucial for understanding and predicting its political leaning. In the United States, there are several parties on the political spectrum, but the two major parties are Democrats and Republicans. Using voter files of the states, we are aiming to be able to use several demographic trends, such as age, income level, and more. In this project, we aim to explore these trends to see if they could have an impact on voter preferences and then use the results to build a model that can predict voters' parties based on these demographics. From this model, we can utilize feature importance to see which predictors are the most influential and which are the least important.

1.1 Initial Setup

We decided to focus our investigation on the state of Oregon. Our voter file has over 2 million rows of data and 726 columns of predictors. Many of these predictors are irrelevant to our analysis and there is a lot of missing data as well, so we will have to clean up our data set and narrow it down. Let's start by setting up all our data.

```Python
#importing libraries
!pip install -q pyspark
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum, upper
import matplotlib.pyplot as plt
from pyspark.sql.functions import when,lit
import pandas as pd
from pyspark.sql.functions import regexp_replace
import time
from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import StandardScaler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

#set up of Voter_Data (downloaded off a Google Cloud Cluster)
DATA_DIRECTORY = ""  ## YOUR DIRECTORY HERE

spark =
SparkSession.builder.master("local").appName("Voter_Data").getOrCreate()

df = (
spark.read.format("parquet")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("{}".format(DATA_DIRECTORY))
)

# Andy, Editted by Alex
# County ID is in a separate file, path is
"gs://winter-2024-voter-file/uscounties.csv". The column name for the county ID
in the csv is "county_fips".
# I included the part to load the csv into df2 variable, and we will need to
join it to df on county names, after resolving any missig data.

# Replace '' with the path to your CSV file
file_path = '/Users/alexroginski/Desktop/School/Y4Q2/PSTAT135/uscounties.csv'

# Read the CSV file into a DataFrame
raw_df2 = spark.read.csv(file_path, header=True, inferSchema=True)

# Filter rows where state_name is "Oregon"
df2 = raw_df2.filter(raw_df2["state_name"] == "Oregon")

# Show the resulting DataFrame
df2.show()
```

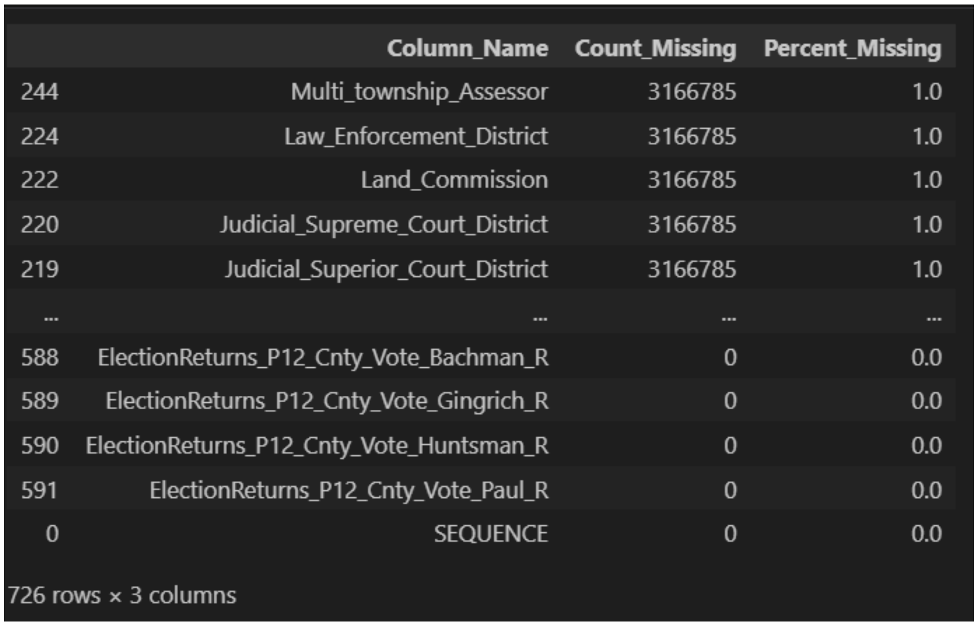


```
+----------+------------+----------------+-----------+--------+----------+-------+------
|    county|county_ascii|     county_full|county_fips|state_id|state_name|    lat|
+----------+------------+----------------+-----------+--------+----------+-------+------
| Multnomah|   Multnomah| Multnomah County|      41051|      OR|    Oregon|45.5468|-122.4
|Washington|  Washington|Washington County|      41067|      OR|    Oregon|45.5601|-123.0
| Clackamas|   Clackamas| Clackamas County|      41005|      OR|    Oregon| 45.188|-122.2
|      Lane|        Lane|     Lane County|      41039|      OR|    Oregon|43.9388|-122.8
|    Marion|      Marion|   Marion County|      41047|      OR|    Oregon|44.9033|-122.5
+----------+------------+----------------+-----------+--------+----------+-------+------
only showing top 5 rows
```

1.2 Missing Data Analysis

Now that we have our voter data downloaded into a data frame, we can look at and fix our missing data issue.

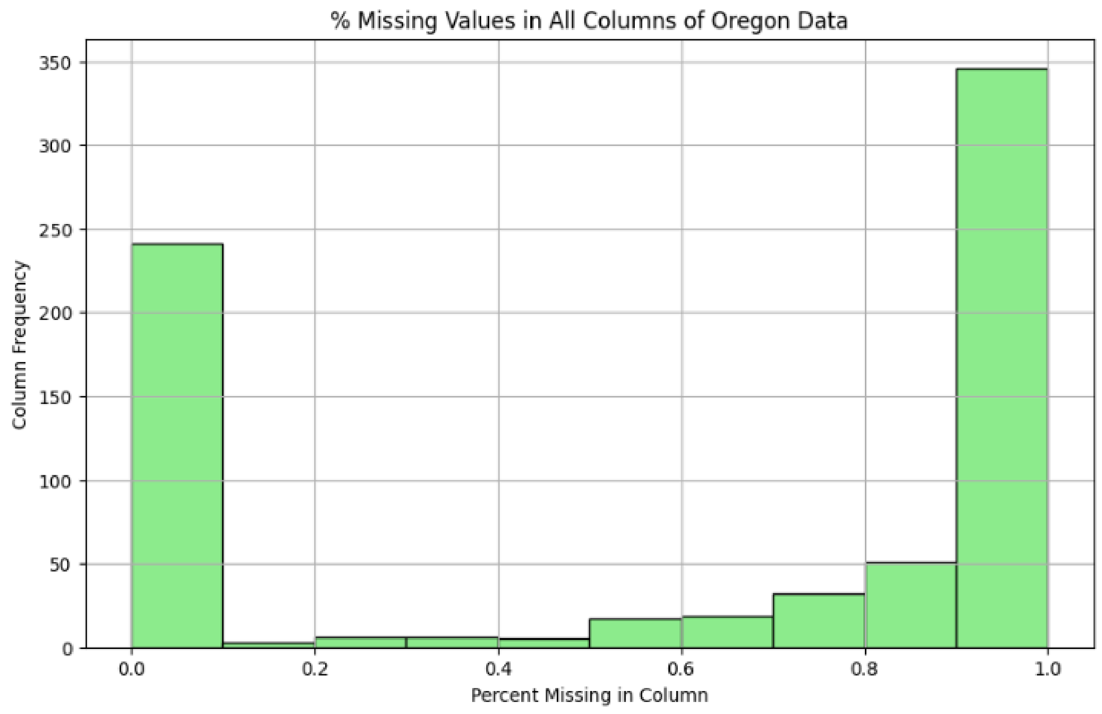
```
Python
# By Axl
# Get missing data for each column
missing_values = df.select([sum(col(column).isNull().cast("int")).alias(column)
for column in df.columns])

# The following took a few minutes to load
# Turn the data into a pandas dataframe to make it readable
missing_values_df = missing_values.toPandas()

# Count total number of rows in dataset
totalNumberOfRows = df.count()

# tranpose it to make the data go down rows instead of across columns
transposed_df = missing_values_df.T.reset_index()
transposed_df.columns = ['Column_Name', 'Count_Missing']  # Rename columns
transposed_df = transposed_df.sort_values(by='Count_Missing',ascending=False)

# sort values by number missing
transposed_df['Percent_Missing'] =
transposed_df['Count_Missing']/totalNumberOfRows # Get percentage missing

transposed_df
```



	Column_Name	Count_Missing	Percent_Missing
244	Multi_township_Assessor	3166785	1.0
224	Law_Enforcement_District	3166785	1.0
222	Land_Commission	3166785	1.0
220	Judicial_Supreme_Court_District	3166785	1.0
219	Judicial_Superior_Court_District	3166785	1.0
...	...	...	...
588	ElectionReturns_P12_Cnty_Vote_Bachman_R	0	0.0
589	ElectionReturns_P12_Cnty_Vote_Gingrich_R	0	0.0
590	ElectionReturns_P12_Cnty_Vote_Huntsman_R	0	0.0
591	ElectionReturns_P12_Cnty_Vote_Paul_R	0	0.0
0	SEQUENCE	0	0.0

726 rows × 3 columns

Python
```python
# By Axl
# Plotting our missing data
# Assuming sorted_df_descending is your DataFrame with sorted missing counts
plt.figure(figsize=(10, 6))
plt.hist(transposed_df['Percent_Missing'], bins=10, color='lightgreen',
edgecolor='black')
plt.xlabel('Percent Missing in Column')
plt.ylabel('Column Frequency')
plt.title('% Missing Values in All Columns of Oregon Data')
plt.grid(True)
plt.show()
```



To visualize how much missing data is in our dataset, we can look at a histogram. Here, each bar represents the number of columns that correspond to the percentage of missing data. Out of 726 columns, 246 of them have 0-10% missing data, so we will drop the columns that have higher amounts of data missing.

1.3 Column selection

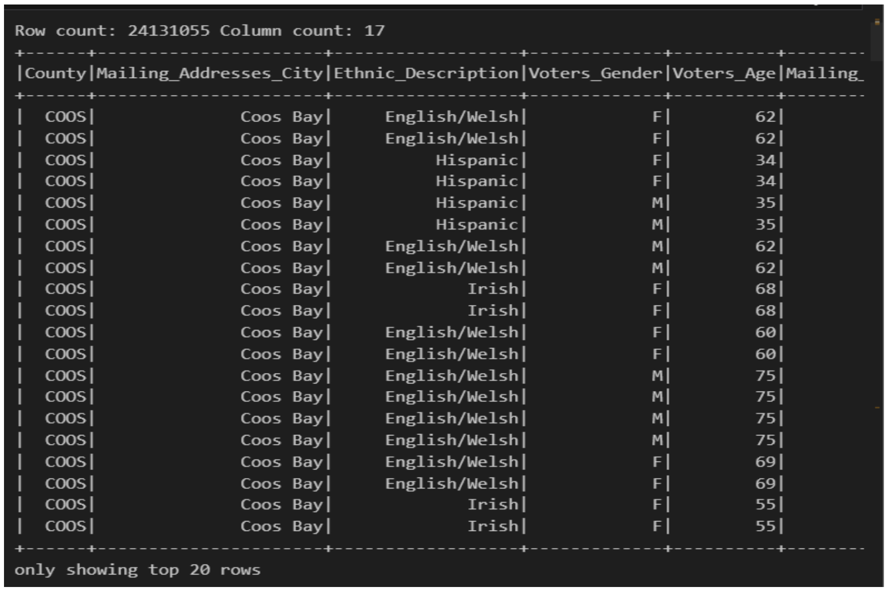
We decided to keep 15 predictor variables that were the most relevant to our project. Our response variable here is ‘Parties_Description’, the party each subject voted for.

```python
# By Alex

selectedColumns = ['County', 'Ethnic_Description', 'Voters_Gender',
'Voters_Age', 'Mailing_Families_HHCount', 'Mailing_HHGender_Description',
'Residence_Families_HHCount', 'CommercialData_EstHomeValue',
'CommercialData_EstimatedHHIncomeAmount',
'CommercialData_EstimatedAreaMedianHHIncome',
'CommercialData_AreaMedianHousingValue',
'CommercialData_AreaMedianEducationYears', 'CommercialData_PropertyType',
'CommercialData_StateIncomeDecile','Parties_Description']

cleaned_df = df.select(selectedColumns)

data = cleaned_df.dropna()
```

1.4 Merging voter data with country code dataframe

Now we can join our two data frames, df_counties and data, to make the final dataframe that we can use throughout our analysis. We will join these on 'counties'.

```
Python
# Andy

# Normalize the "County" column in both DataFrames to uppercase
df1 = data.withColumn("County", upper(data["County"]))
df2 = df2.withColumn("county", upper(df2["county"]))

df_joined = df1.join(df2, df1["County"] == df2["county"])

# Selecting the desired columns from the joined DataFrame
df_merged = df_joined.select(df1["*"], df2["county_fips"])

print("Row count:", df_merged.count(), "Column count:", len(df_merged.columns))
df_merged.show()
```



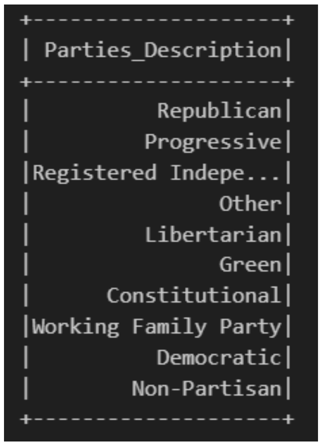
```
Row count: 24131055 Column count: 17
+------+---------------------+-----------------+-------------+----------+--------
|County|Mailing_Addresses_City|Ethnic_Description|Voters_Gender|Voters_Age|Mailing_
+------+---------------------+-----------------+-------------+----------+--------
|  COOS|             Coos Bay|    English/Welsh|            F|        62|
|  COOS|             Coos Bay|    English/Welsh|            F|        62|
|  COOS|             Coos Bay|         Hispanic|            F|        34|
|  COOS|             Coos Bay|         Hispanic|            F|        34|
|  COOS|             Coos Bay|         Hispanic|            M|        35|
|  COOS|             Coos Bay|         Hispanic|            M|        35|
|  COOS|             Coos Bay|    English/Welsh|            M|        62|
|  COOS|             Coos Bay|    English/Welsh|            M|        62|
|  COOS|             Coos Bay|            Irish|            F|        68|
|  COOS|             Coos Bay|            Irish|            F|        68|
|  COOS|             Coos Bay|    English/Welsh|            F|        60|
|  COOS|             Coos Bay|    English/Welsh|            F|        60|
|  COOS|             Coos Bay|    English/Welsh|            M|        75|
|  COOS|             Coos Bay|    English/Welsh|            M|        75|
|  COOS|             Coos Bay|    English/Welsh|            M|        75|
|  COOS|             Coos Bay|    English/Welsh|            M|        75|
|  COOS|             Coos Bay|    English/Welsh|            F|        69|
|  COOS|             Coos Bay|    English/Welsh|            F|        69|
|  COOS|             Coos Bay|            Irish|            F|        55|
|  COOS|             Coos Bay|            Irish|            F|        55|
+------+---------------------+-----------------+-------------+----------+--------
only showing top 20 rows
```

Here’s an example of how our merged dataframe looks at this point. It’s much cleaner and easier to work with during our investigation.

1.5 Binning party descriptions

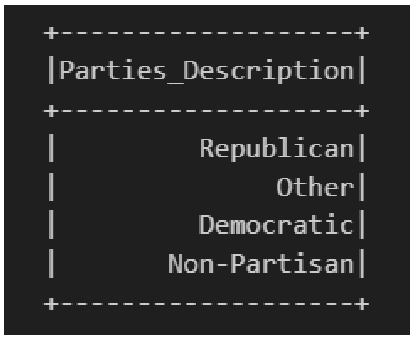
The last step to cleaning our data is to put our party descriptions into bins. There are 10 unique parties listed in the voter files, but only a very small percentage of voters voted for most of the parties, so we binned those parties into a category called ‘Other.’

```python
# Axl - Checking for unique values in the Parties_Description column
unique_parties = df_merged.select('Parties_Description').distinct()
unique_parties.show()
```



```
+--------------------+
| Parties_Description|
+--------------------+
|          Republican|
|         Progressive|
|Registered Indepe...|
|               Other|
|         Libertarian|
|               Green|
|       Constitutional|
|Working Family Party|
|          Democratic|
|         Non-Partisan|
+--------------------+
```

```python
# Andy's code to merge bin

# List of parties to keep
parties_to_keep = ['Republican', 'Democratic', 'Non-Partisan']

df_merged = df_merged.withColumn("Parties_Description",
when(col("Parties_Description").isin(parties_to_keep),
col("Parties_Description")).otherwise(lit("Other")))
```


```
+--------------------+
|Parties_Description|
+--------------------+
|          Republican|
|               Other|
|          Democratic|
|        Non-Partisan|
+--------------------+
```

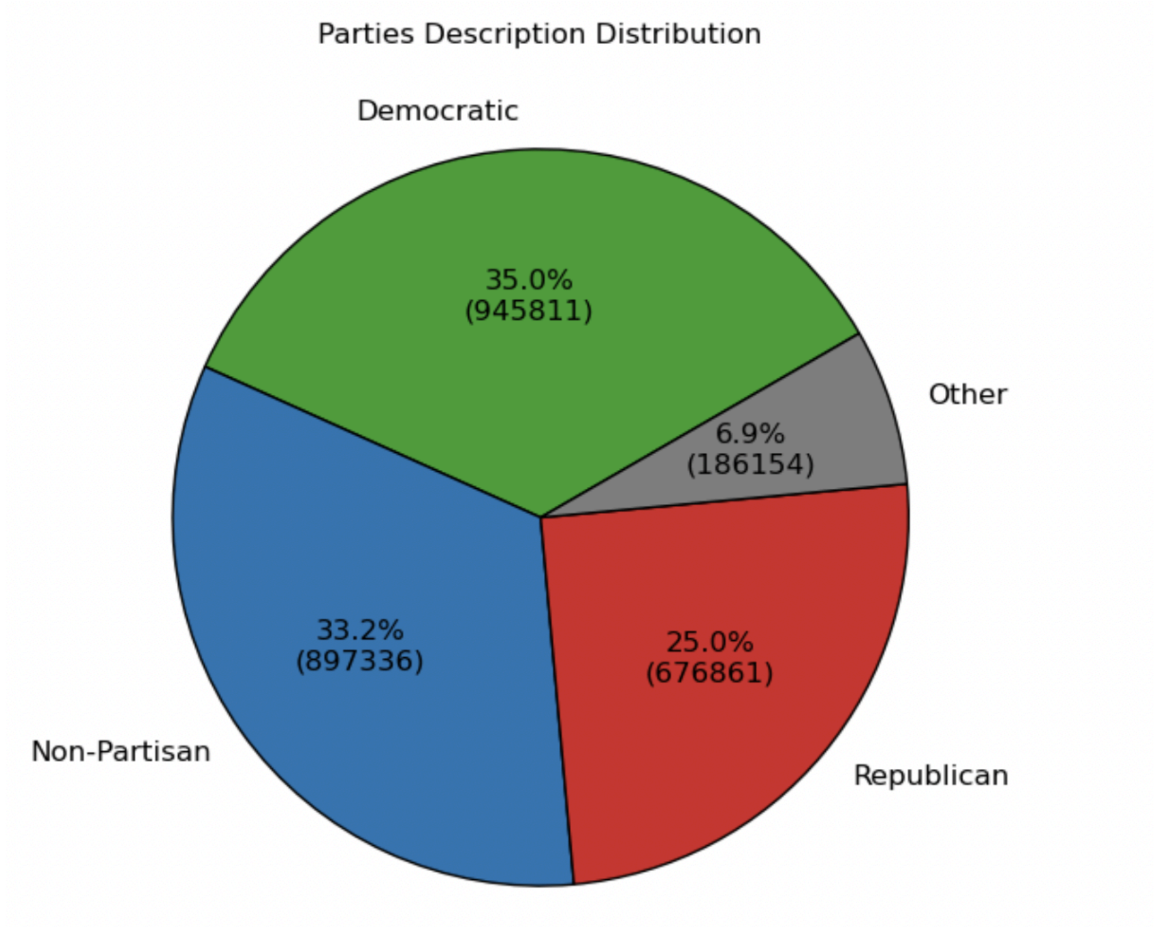
Now that our data is all set-up, we can start with our EDA.

2. Exploratory Data Analysis

2.1 Party Preference Distribution

The first step of our exploratory data analysis is to look at the distribution of our response variable Party Description. To do this, we can look at a pie chart.

Python
```python
#  By Aapthi, edited by Alex

import matplotlib.pyplot as plt

import numpy as np

parties_count_df = df_merged.groupBy("Parties_Description").count()

# Convert to Pandas DataFrame
parties_count_pdf = parties_count_df.toPandas()

# Sorting for consistency
parties_count_pdf.sort_values('count', ascending=False, inplace=True)

# Customization parameters
colors = ['tab:green', 'tab:blue', 'tab:red', 'tab:grey',]  # Custom colors
wp = {'linewidth': 1, 'edgecolor': "black"}  # Wedge properties

# Function to format the autopct
def func(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d})".format(pct, absolute)

# Plotting
fig, ax = plt.subplots(figsize=(10, 7))
wedges, texts, autotexts = ax.pie(parties_count_pdf['count'],
                                  autopct=lambda pct: func(pct,
parties_count_pdf['count']),

labels=parties_count_pdf['Parties_Description'],
                                  startangle=30,
                                  wedgeprops=wp,
                                  colors=colors,
                                  textprops=dict(color="black", size=12))

plt.title('Parties Description Distribution')
plt.show()
```



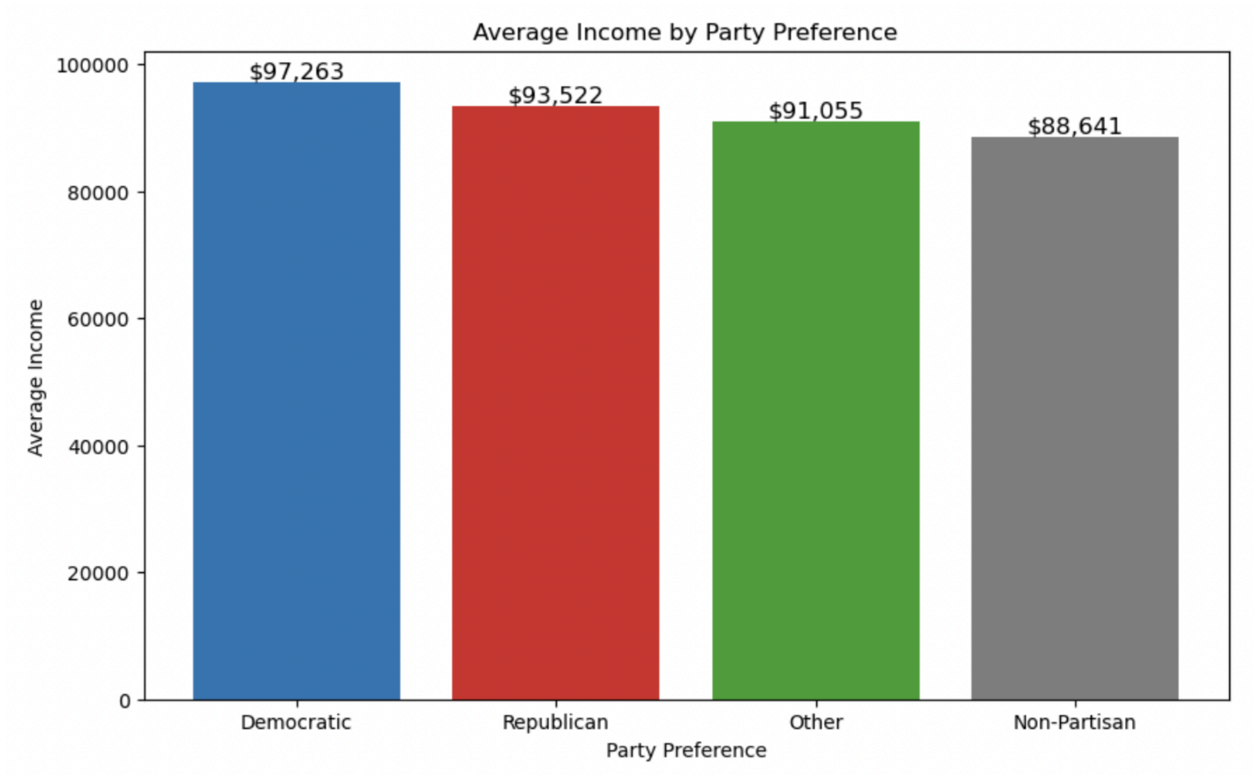
The Democratic party received the most votes, followed closely by the Nonpartisan and Republican parties. Since the rest of the parties made up less than 7% of the data, they've all been binned into the 'Other' category. It's important to note that this distribution is specific to the Oregon voter files, not the whole country.

2.2 Income vs. Party Preference

In this section we will examine the relationship between economic status and political party preferences in the state of Oregon. As we know, the state of Oregon leans Democratic in all of its elections, so it is a particularly interesting case study for examining and analyzing the dynamics between economic status and party preferences. Specifically, we will be examining whether household income levels influence people's party preferences in Oregon, and how party preferences vary across different income levels.

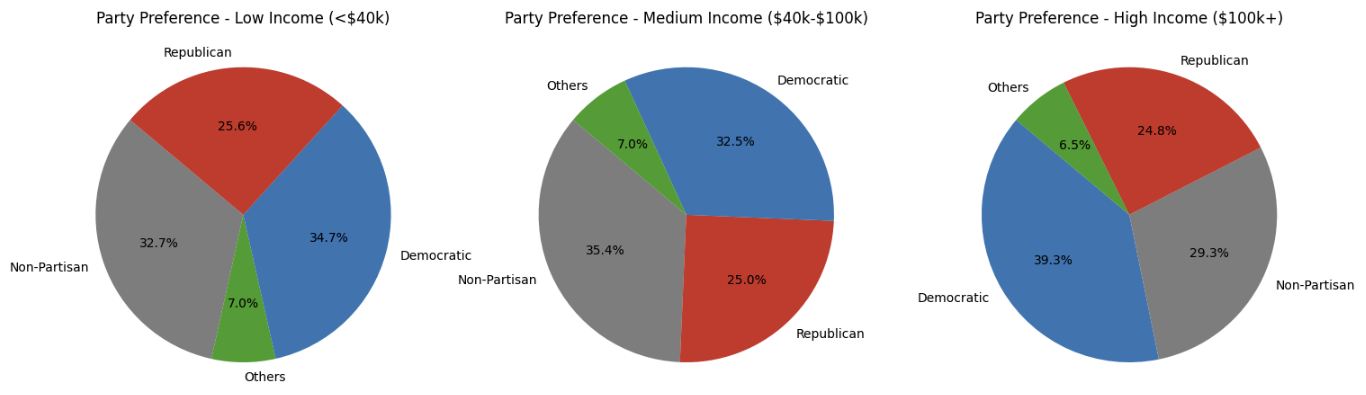
Python

```python
from pyspark.sql.functions import regexp_replace, col
from pyspark.sql import functions as F
import matplotlib.pyplot as plt

# Define colors for the bars
colors = ['tab:blue', 'tab:red', 'tab:green', 'tab:grey']

df_merged_temp = df_merged.withColumn('Income_Amount',

regexp_replace(col('CommercialData_EstimatedHHIncomeAmount'), '[$,]',
'').cast('float'))

party_income_df = df_merged_temp.groupBy('Parties_Description')\
    .agg(F.avg('Income_Amount').alias('Average_Income'))

party_income_pdf = party_income_df.toPandas()

# Sort the DataFrame by 'Average_Income' column in descending order
party_income_pdf_sorted = party_income_pdf.sort_values(by='Average_Income',
ascending=False)

plt.figure(figsize=(10, 6))
bars = plt.bar(party_income_pdf_sorted['Parties_Description'],
party_income_pdf_sorted['Average_Income'], color=colors)
plt.xlabel('Party Preference')
plt.ylabel('Average Income')
plt.title('Average Income by Party Preference')
plt.xticks(rotation=0)

# Iterate over the bars and use plt.text() to display the value on each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, '$' +
'{:,.0f}'.format(round(yval, 0)), fontsize=12, va='bottom', ha='center')

plt.show()
```



The bar plot above shows the average household income for each party preference. As seen, people who voted democrat have a slightly higher household income on average than the other parties. People who are non-partisan have the lowest household income on average. Overall, the differences between parties are minimal, we should take a further look by breaking income into distinct income levels, and see which party each level tends to favor. Here we used data provided by the state of Oregon, and classified income into levels accordingly; Less than 40k a year is Low, 40k - 100k is Medium, and over 100k is High.

```
Python
from pyspark.sql.functions import when, col

df_merged_temp = df_merged_temp.withColumn('Income_Level',
              when(col('Income_Amount') < 40000, 'Low Income')
              .when(col('Income_Amount') <= 100000, 'Medium Income')
              .otherwise('High Income'))

income_party_dist = df_merged_temp.groupBy('Income_Level',
'Parties_Description').count()

income_party_pdf = income_party_dist.toPandas()

# Plotting
income_levels = ['Low Income', 'Medium Income', 'High Income']
party_colors = {'Democratic': 'tab:blue', 'Republican': 'tab:red', 'Others':
'tab:green', 'Non-Partisan': 'tab:grey'}
income_levels_amount = {'Low Income': "(<$40k)", 'Medium Income':
"(\$40k-\$100k)", 'High Income': "($100k+)"}
fig, axs = plt.subplots(1, len(income_levels), figsize=(18, 6))  # 1 row,
len(income_levels) columns

for idx, level in enumerate(income_levels):
    subset = income_party_pdf[income_party_pdf['Income_Level'] == level]
    colors = [party_colors[party] for party in subset['Parties_Description']]
    axs[idx].pie(subset['count'], labels=subset['Parties_Description'],
autopct='%1.1f%%', startangle=140, colors=colors)
    axs[idx].set_title(f'Party Preference - {level}
{income_levels_amount[level]}')

plt.show()
```



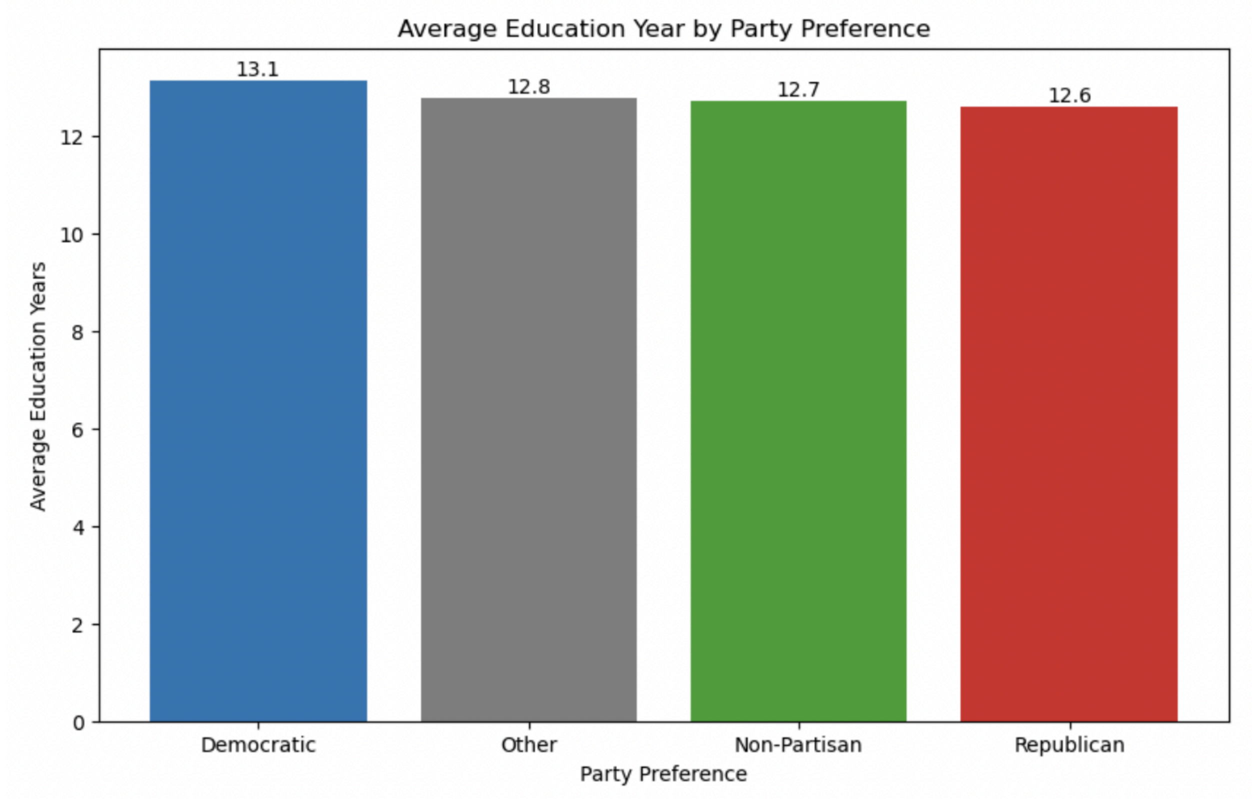
From the plots, we can see that the Democratic Party is the preferred party in both low-income and high-income households, with them leading the high-income households by a significant margin (~7%). Preferences for the Republican party are fairly stable across all income levels, potentially indicating a consistent base of support regardless of income. A significant portion of each income level also prefers the non-partisan party, with the highest in the medium-income house in which they lead the group by a small margin (~3%). These results suggest that income level does correlate with political party preference in the state of Oregon to an extent, however, the relationship seems very weak.

2.3 Education Level vs Party Preference

The next section focuses on the highest level of education of the voters in Oregon. We want to see whether the highest average education level varies among voters of different party preferences. This can help us determine if education levels are influential in predicting party affiliation amongst voters in Oregon.

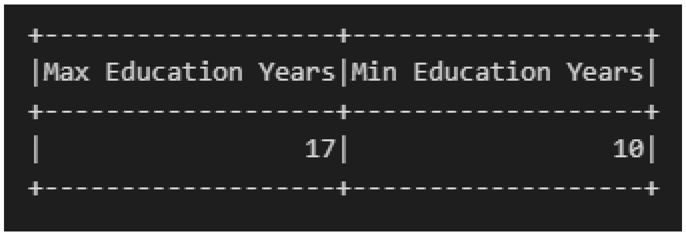
```python
# by Aapthi, edited by Alex
from pyspark.sql.functions import regexp_replace, col
from pyspark.sql import functions as F
import matplotlib.pyplot as plt

df_merged_temp = df_merged.withColumn('Education_Years',

regexp_replace(col('CommercialData_AreaMedianEducationYears'), '[$,]',
'').cast('int'))

education_df = df_merged_temp.groupBy('Parties_Description')\
    .agg(F.avg('Education_Years').alias('Education_Years'))

party_education_df = education_df.toPandas()

# Sort the DataFrame by 'Education_Years'
party_education_df_sorted =
party_education_df.sort_values(by='Education_Years',  ascending=False)

# Plotting the Average Education Years vs Parties
plt.figure(figsize=(10, 6))
bars = plt.bar(party_education_df_sorted['Parties_Description'],
party_education_df_sorted['Education_Years'],
        color = ['tab:blue', 'tab:Grey', 'tab:green', 'tab:red'])
```

```python
plt.xlabel('Party Preference')
plt.ylabel('Average Education Years')
plt.title('Average Education Year by Party Preference')
plt.xticks(rotation=0)

# Adding the value above the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 1),
ha='center', va='bottom')

plt.show()
```



Here, we can see that average years of schooling really doesn't have a large impact on voting preferences. The average education level of Oregon voters is between 12 and 13 years. Democratic voters have slightly more years of schooling than the other parties, but only by a very small margin. Republican voters have slightly less schooling than the other parties, but also by a very small margin. Similar to the previous section, we should take a closer look by classifying education years into three distinct groups: Low, Middle, and High. Since not much is known about education years, let's take a look at the max and min values for it to determine the range we should set each group to.

```python
# by Andy
from pyspark.sql.functions import max, min

df_merged_education_temp = df_merged.withColumn("Education_Years",
col("CommercialData_AreaMedianEducationYears").cast("int"))

max_min = df_merged_education_temp.agg(
  max("Education_Years").alias("Max Education Years"),
  min("Education_Years").alias("Min Education Years")
)

max_min.show()
```

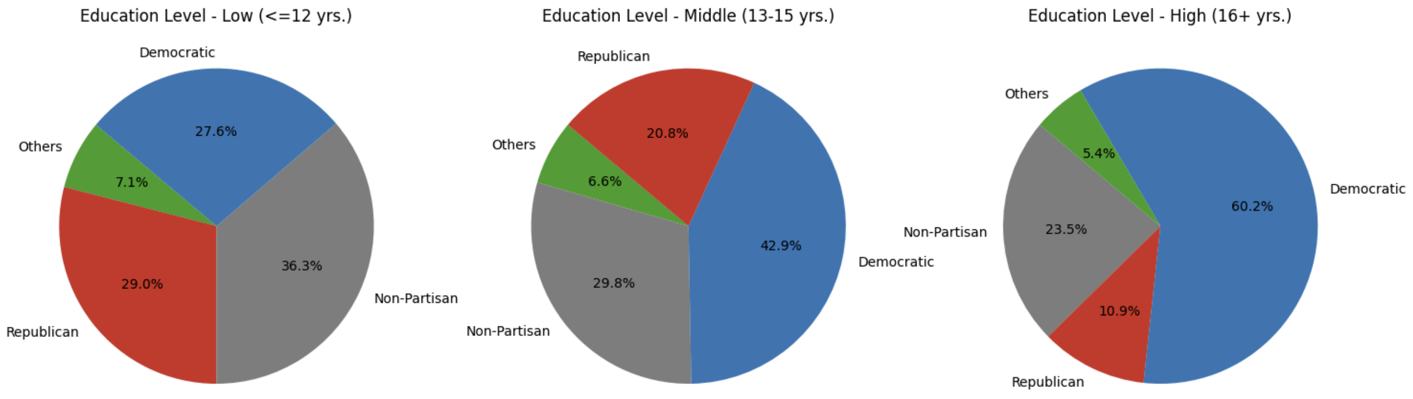


```
+-------------------+-------------------+
|Max Education Years|Min Education Years|
+-------------------+-------------------+
|                 17|                 10|
+-------------------+-------------------+
```

Max and Min education years in Oregon data are 17 and 10 years respectively. With this knowledge, we will assign anyone who has below 12 years of education to the Low group, 12-15 years to the Middle group, and 15+ years to the High group. Below is a visualization of the groups and their party preferences.

```python
df_binned = df_merged_education_temp.withColumn("Education_Years_Bin",
  when(col("Education_Years") < 12, "Low")
  .when(col("Education_Years") <= 15, "Middle")
  .otherwise("High")
)

df_grouped = df_binned.groupBy("Education_Years_Bin",
"Parties_Description").count()
df_grouped_pdf = df_grouped.toPandas()
```

```
ed_levels = ['Low', 'Middle', 'High']
party_colors = {'Democratic': 'tab:blue', 'Republican': 'tab:red', 'Others':
'tab:green', 'Non-Partisan': 'tab:grey'}
ed_levels_amount = {'Low': "(<12 yrs.)", 'Middle': "(12-15 yrs.)", 'High':
"(15+ yrs.)"}

fig, axs = plt.subplots(1, len(ed_levels), figsize=(18, 6))

for idx, level in enumerate(ed_levels):
    subset = df_grouped_pdf[df_grouped_pdf['Education_Years_Bin'] == level]
    colors = [party_colors[party] for party in subset['Parties_Description']]
    axs[idx].pie(subset['count'], labels=subset['Parties_Description'],
autopct='%1.1f%%', startangle=140, colors=colors)
    axs[idx].set_title(f'Education Level - {level} {ed_levels_amount[level]}')

plt.show()
```



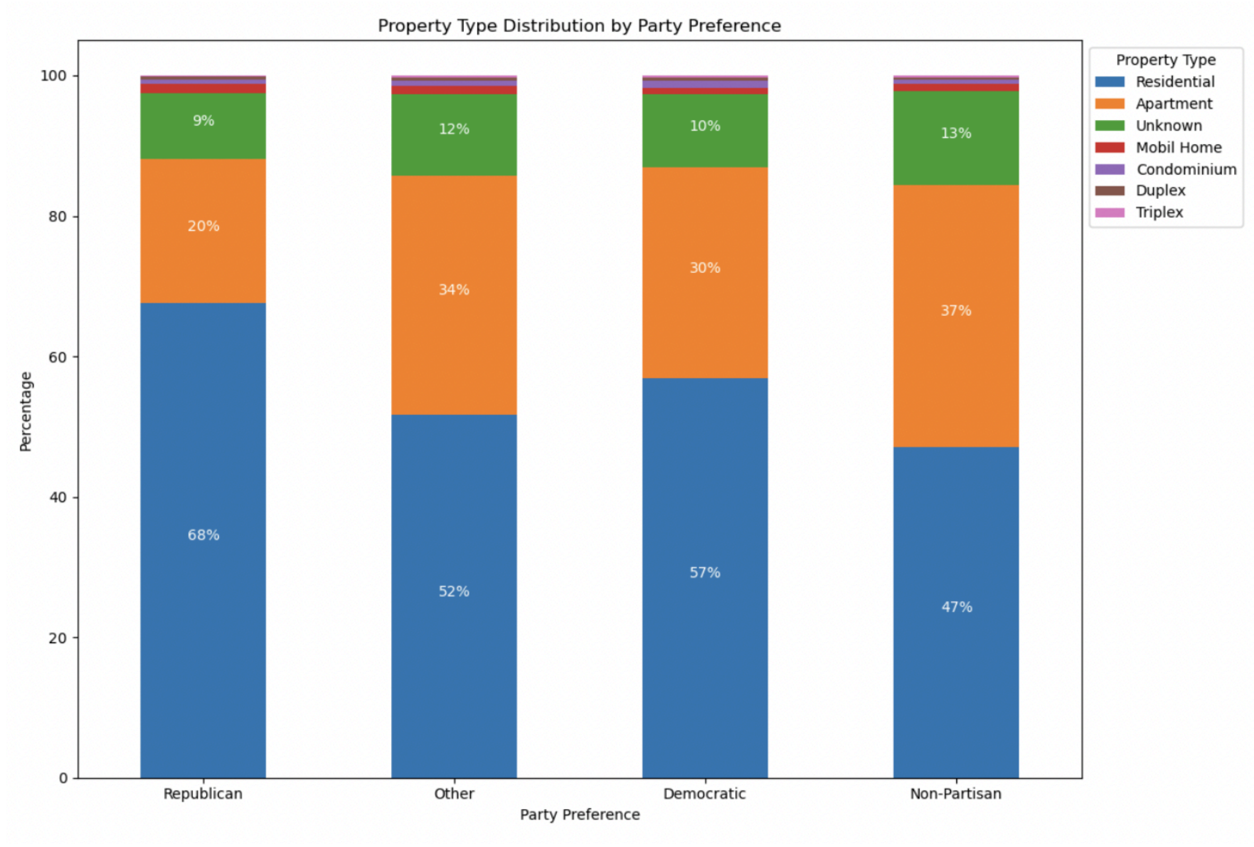
From the plots, we can see that Democratic Party is the clear favorite amongst voters who have a Middle or High education level, and they lead the High Education level by an enormous margin over the other parties. Though, they have relatively low support from voters who have a Low education level. Non-partisan on the other hand, have the most support from voters with Low education levels compared to the other parties, and the support decreases relatively as we move up the educational ladder. Support for the Republican party is also mainly concentrated in the Low education level, and their support is very weak amongst the Highly educated. These pie charts do suggest that Education level plays a significant role in people's party preferences, as seen by the large differences.

2.4 Property type vs Party Preference

Previous section has suggested there may be a relationship between household income and their party preferences, so in this section we continue to investigate if different property types owned – residential, apartment, condominium, etc. – relate to political party preferences. The variable 'CommercialData_PropertyType' in our dataset categorizes the type of the property owned by each individual voter.

```
Python
# Originally by Andy, editted by Alex

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Create a pivot table with 'Parties_Description' as rows and
'CommercialData_PropertyType' as columns, with counts
party_property_df = df_merged.groupBy('Parties_Description')\
  .pivot('CommercialData_PropertyType')\
  .count()

# Convert to Pandas DataFrame
party_property_pdf = party_property_df.toPandas()

# Set the index to 'Parties_Description' for plotting
party_property_pdf.set_index('Parties_Description', inplace=True)

# Convert counts to percentages
party_property_pdf = party_property_pdf.div(party_property_pdf.sum(axis=1),
axis=0) * 100

# Sort columns by their total percentage across all parties
party_property_pdf =
party_property_pdf.reindex(party_property_pdf.sum().sort_values(ascending=False
).index, axis=1)

# Plotting
fig, ax = plt.subplots(figsize=(12, 8))
party_property_pdf.plot(kind='bar', stacked=True, ax=ax)

plt.xlabel('Party Preference')
plt.ylabel('Percentage')
plt.title('Property Type Distribution by Party Preference')
plt.xticks(rotation=0)
```

```
# Place the legend outside the plot area
plt.legend(title='Property Type', loc='upper left', bbox_to_anchor=(1, 1))

# Custom function to add labels
def add_labels(ax):
  for c in ax.containers:
    labels = [f'{v.get_height():.0f}%' if v.get_height() > 5 else '' for v in c]
    ax.bar_label(c, labels=labels, label_type='center', color='white', rotation=0,
padding=3)

add_labels(ax)

plt.tight_layout()
```



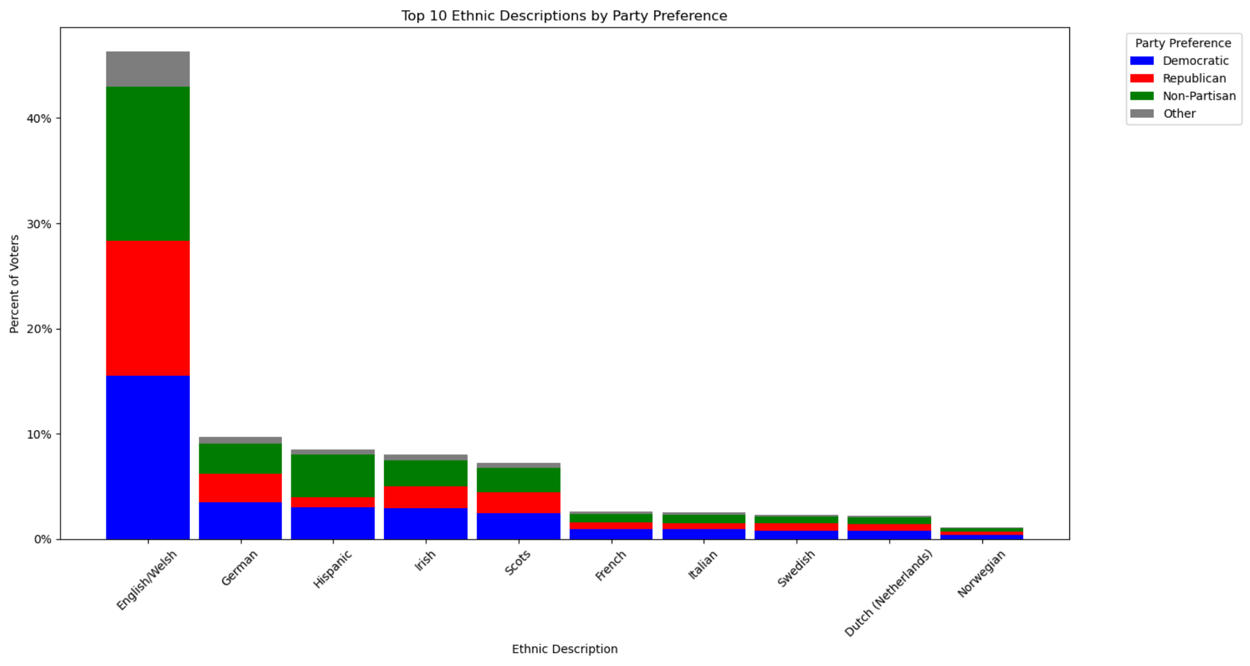
From this stacked bar plot, we can see that for all Oregon voters, regardless of party preference, the residential property type is the highest reported. Republicans have the most support from voters who live in Residential property, while voters who live in an Apartment tend to favor Non-Partisan and Other parties more. However, the differences are not very large, so we cannot conclude property type is an influential indicator of party preference.

2.5 Top 10 Ethnic Descriptions vs Party Preference

When exploring the ethnic descriptions predictor, we decided to narrow our EDA to the top 10 ethnicities. This decision was made when we saw that the large majority of our voters fall into these 10 ethnicities, while the rest of them had very few ethnicities. By narrowing down the ethnicities, we can still get a thorough understanding of the distribution and its interaction with party affiliation.

```python
# Axl - Plotting histogram for party description per the top 20 ethnic
descriptions
# had to limit it to 10 due to the large number which made visualization
# difficult

from pyspark.sql.functions import col, count
import matplotlib.pyplot as plt


# df for plotting
result_df =
df_merged.groupBy('Ethnic_Description').pivot('Parties_Description').count().na
.fill(0)
result_df = result_df.withColumn('Total', __builtins__.sum(result_df[col] for
col in result_df.columns[1:]))
result_df = result_df.orderBy(col('Total').desc()).limit(10)
result_df = result_df.drop('Total')

# Define the desired order
desired_order = ["Democratic", "Republican", "Non-Partisan", "Registered
Independent", "Other"]
parties_ordered = [party for party in desired_order if party in
result_df.columns]
parties_ordered += [party for party in result_df.columns if party not in
desired_order and party != 'Ethnic_Description']

result_data = result_df.select(['Ethnic_Description'] +
parties_ordered).collect()
categories = [row['Ethnic_Description'] for row in result_data]
counts = [list(row[1:]) for row in result_data]  # Exclude the first column
which is 'Ethnic_Description'
```

```
total_per_group = [__builtins__.sum(row) for row in counts]
grand_total = df_merged.count()

fig, ax = plt.subplots(figsize=(15, 8))
bar_width = 0.9
bottoms = [0] * len(categories)


party_colors = {
    "Democratic": "blue",
    "Republican": "red",
    "Non-Partisan": "green",
    "Other": "grey",
}


for i, party in enumerate(parties_ordered):
    party_counts = [row[i + 1] for row in result_data]  # Offset due to
'Ethnic_Description'
    party_counts = [count / grand_total * 100 for count in party_counts]
    ax.bar(categories, party_counts, width=bar_width, label=party,
bottom=bottoms, color=party_colors.get(party, "black"))
    bottoms = [bottom + party_counts[j] for j, bottom in enumerate(bottoms)]



ax.set_xlabel('Ethnic Description')
ax.set_ylabel('Percent of Voters')
ax.set_title('Top 10 Ethnic Descriptions by Party Preference')
ax.set_xticks(range(len(categories)))
ax.set_xticklabels(categories, rotation=45)


from matplotlib.ticker import PercentFormatter
plt.gca().yaxis.set_major_formatter(PercentFormatter(decimals=0))

ax.legend(title='Party Preference', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



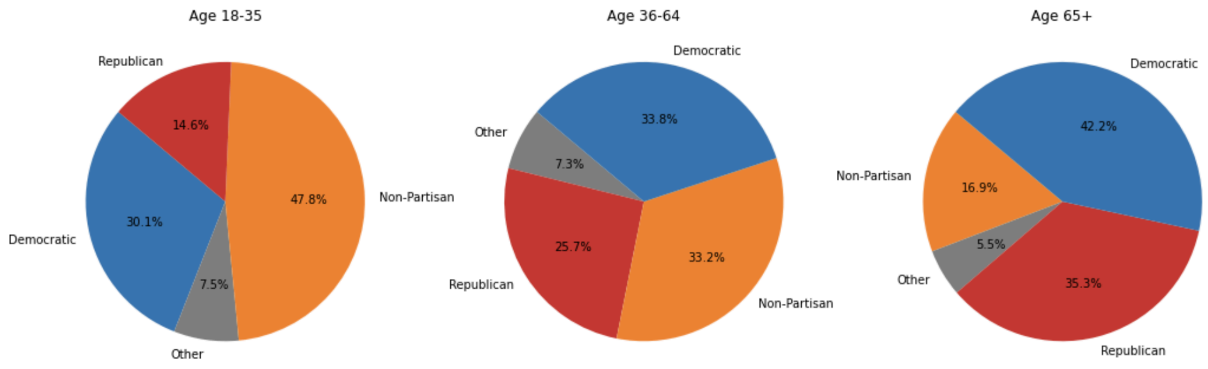
While looking at the distribution of ethnicity in our data, we can see from the bar plot that by far, the largest amount of voters are English/Welsh. This is worthy of note since Oregon is known to be less diverse in its population than other, more populated states. The parties are split relatively evenly among the top ten ethnic descriptions, with the exception of the 'other' bin, which has the least votes in all ethnic groups. Also, we can see that the Hispanic ethnic population of Oregon has fewer Republican voters and more Non-partisan voters.

2.5 Age vs Party Preference

Next, we can look at the age distribution of the voters in our data and the interaction between it and party affiliation. In order to understand the distribution of ages better, we binned our data into 3 levels and used this to separately look at different age groups and their preferences.

```
Python
# By Aapthi, adapted from Andy's code
df_merged_temp = df_merged_temp.withColumn('Age_Level',
                                 when(col('Voters_Age') < 36, '18-35',)
                                 .when(col('Voters_Age') < 65, '36-64')
                                 .otherwise('65+'))
```

```
age_party_dist = df_merged_temp.groupBy('Age_Level',
'Parties_Description').count()

age_party_pdf = age_party_dist.toPandas()


# Plotting

age_levels = ['18-35', '36-64', '65+']
party_colors = {'Democratic': 'tab:blue', 'Republican': 'tab:red', 'Other':
'tab:grey', 'Non-Partisan': 'tab:orange'}
age_levels_titles = {'Low Income': "(<$40k)", 'Medium Income':
"(\$40k-\$100k)", 'High Income': "($100k+)"}
fig, axs = plt.subplots(1, len(age_levels), figsize=(18, 6))  # 1 row,
len(income_levels) columns


for idx, level in enumerate(age_levels):
    subset = age_party_pdf[age_party_pdf['Age_Level'] == level]
    colors = [party_colors[party] for party in subset['Parties_Description']]
    axs[idx].pie(subset['count'], labels=subset['Parties_Description'],
                 autopct='%1.1f%%', startangle=140, colors=colors)
    axs[idx].set_title(f'Age {level}')

plt.show()
```



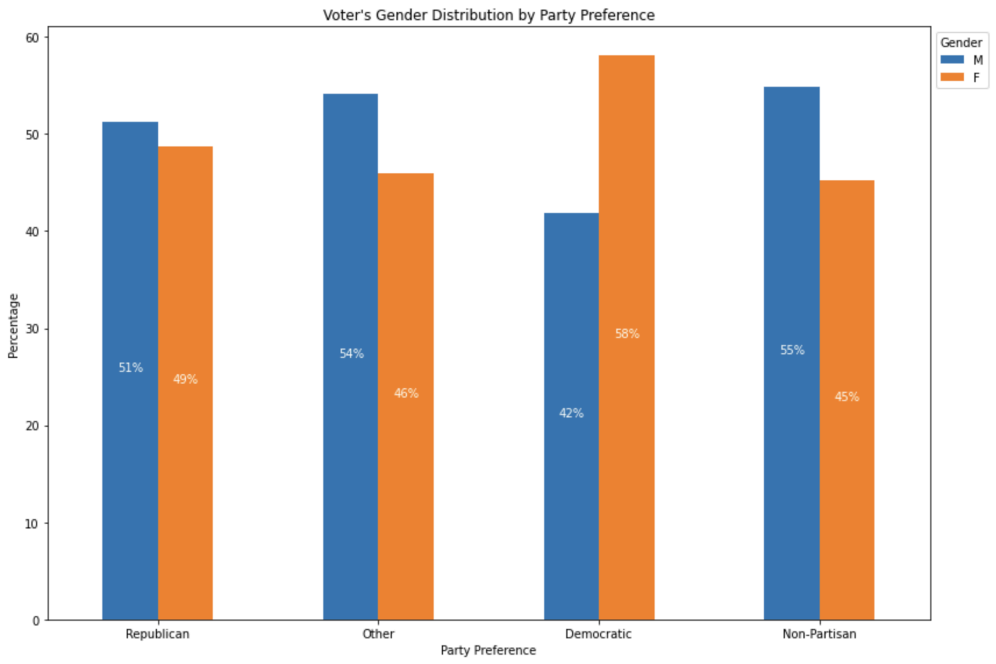
For the distribution of party preferences in regards to different age groups, we can use pie charts. Here, we binned the ages into 3 groups: 18-35 years, 36-64 years, and 65+ years old. In the first and youngest age group, Non-Partisan votes make up nearly half of the distribution, followed by Democratic votes, which makes up about a third. In the second age group, Democratic, Non-Partisan, and Republican votes are more evenly split, with Democratic votes taking the majority this time. Finally, for the third age group we can see that the Non-Partisan votes are fewer in percentage and there are both more Democratic and Republican votes in the distribution. This indicates that younger voters in Oregon tend to vote more in the middle, but they also tend to choose a side ideologically as they get older, and the vote splits more between the Democratic and Republican parties.

2.6 Gender Description vs Party Preference

The final section of our EDA is focused on the gender description distribution of Oregon voters and its interaction with party preferences.

```python
party_gender_df = df_merged.groupBy('Parties_Description')
    .pivot('Voters_Gender')
    .count()

# convert to pd dataframe

party_gender_pdf = party_gender_df.toPandas()
party_gender_pdf.set_index('Parties_Description', inplace=True)

party_gender_pdf = party_gender_pdf.div(party_gender_pdf.sum(axis=1), axis=0) *
100

# sort columns by their total percentage

party_gender_pdf =
party_gender_pdf.reindex(party_gender_pdf.sum().sort_values(ascending=False).in
dex, axis=1)

# plotting

fig, ax = plt.subplots(figsize=(12, 8))
party_gender_pdf.plot(kind='bar', stacked=False, ax=ax)
plt.xlabel('Party Preference')
plt.ylabel('Percentage')
plt.title('Voter\'s Gender Distribution by Party Preference')
plt.xticks(rotation=0)
plt.legend(title='Gender', loc='upper left', bbox_to_anchor=(1, 1))
```

```
# Custom function to add percent labels

def add_labels(ax):
    for c in ax.containers:
        labels = [f'{v.get_height():.0f}%' if v.get_height() > 5 else '' for v
in c]
        ax.bar_label(c, labels=labels, label_type='center', color='white',
rotation=0, padding=3)

add_labels(ax)

plt.tight_layout()
```



Here we can see that for all parties, males in Oregon have a higher percentage of the votes, with the exception of the democratic party where females make up 58% of the votes while males only make up 42%. This shows that gender does seem to matter when it comes to party preference in Oregon, so we can study this further in our modeling.

3. Modeling

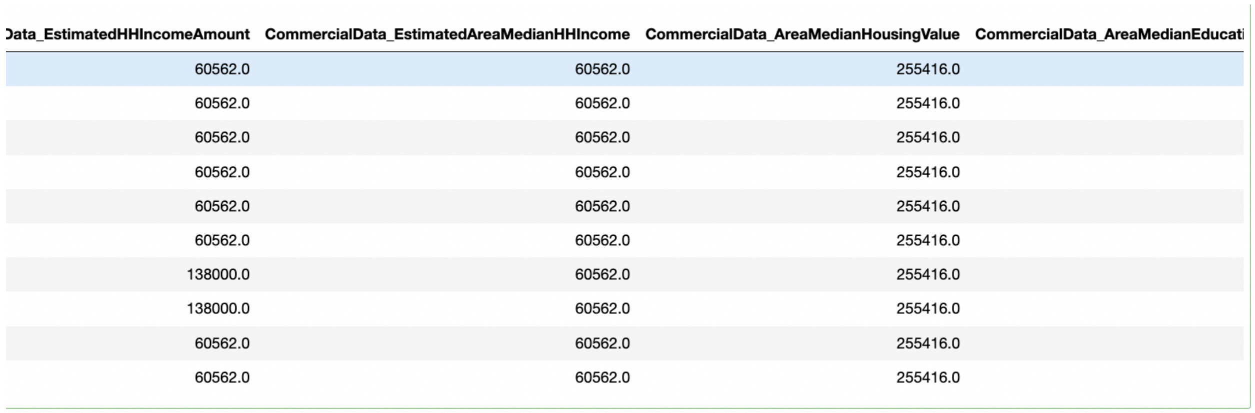
Now, we will create a random forest and multinomial logistic regression to predict voter party preference.

3.1 Preparations for modeling

To prepare the data for the models, we must change the data formats of some of the columns in the voter data. Some of the numerical data, such as home value, is stored in the data with dollar signs, e.g. $150,000, so we have to convert it to numeric. Also, all of the numeric columns, such as voter age, are also stored as categorical columns, so we have to convert them to numeric.

```python
# By Axl

# Converting $ values to numeric AND Convert numeric columns stored as string
to numeric

# Convert string columns that contain "$" into numeric

df_merged = df_merged.withColumn('CommercialData_EstHomeValue',

regexp_replace(col('CommercialData_EstHomeValue'), '[$,]', '').cast('float'))

df_merged = df_merged.withColumn('CommercialData_EstimatedHHIncomeAmount',

regexp_replace(col('CommercialData_EstimatedHHIncomeAmount'), '[$,]',
'').cast('float'))

df_merged = df_merged.withColumn('CommercialData_EstimatedAreaMedianHHIncome',

regexp_replace(col('CommercialData_EstimatedAreaMedianHHIncome'), '[$,]',
'').cast('float'))

df_merged = df_merged.withColumn('CommercialData_AreaMedianHousingValue',

regexp_replace(col('CommercialData_AreaMedianHousingValue'), '[$,]',
'').cast('float'))


# Convert numeric columns stored as numeric into string

# Columns to convert to numeric
```

```
columns_to_convert = ["Voters_Age", "Mailing_Families_HHCount",
"Residence_Families_HHCount", "CommercialData_AreaMedianEducationYears"]

# Convert columns to numeric type
for column in columns_to_convert:
    df_merged = df_merged.withColumn(column, col(column).cast('float'))
# PRINT RESULT

test_rows = df_merged.head(10)

column_headers= df_merged.columns

new_df = pd.DataFrame(test_rows, columns=column_headers)

new_df
```

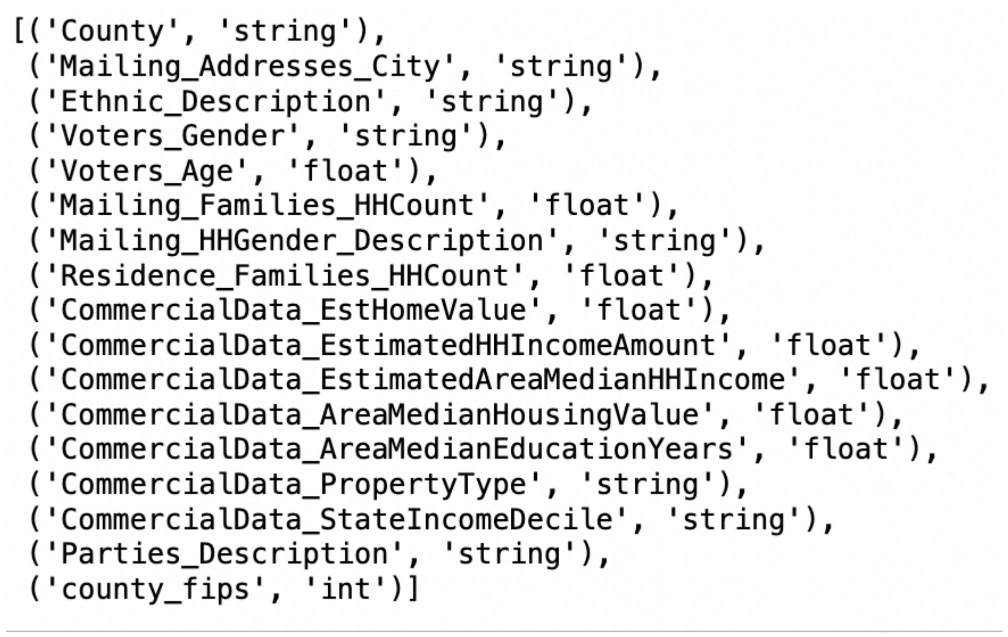


Data_EstimatedHHIncomeAmount	CommercialData_EstimatedAreaMedianHHIncome	CommercialData_AreaMedianHousingValue	CommercialData_AreaMedianEducati
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
138000.0	60562.0	255416.0	
138000.0	60562.0	255416.0	
60562.0	60562.0	255416.0	
60562.0	60562.0	255416.0	

Check column types (after converting x$ values to x as numeric)

After the data preparation, this is how our columns look:

```
Python
df_merged.dtypes
```



```
[('County', 'string'),
 ('Mailing_Addresses_City', 'string'),
 ('Ethnic_Description', 'string'),
 ('Voters_Gender', 'string'),
 ('Voters_Age', 'float'),
 ('Mailing_Families_HHCount', 'float'),
 ('Mailing_HHGender_Description', 'string'),
 ('Residence_Families_HHCount', 'float'),
 ('CommercialData_EstHomeValue', 'float'),
 ('CommercialData_EstimatedHHIncomeAmount', 'float'),
 ('CommercialData_EstimatedAreaMedianHHIncome', 'float'),
 ('CommercialData_AreaMedianHousingValue', 'float'),
 ('CommercialData_AreaMedianEducationYears', 'float'),
 ('CommercialData_PropertyType', 'string'),
 ('CommercialData_StateIncomeDecile', 'string'),
 ('Parties_Description', 'string'),
 ('county_fips', 'int')]
```

We see that the columns have been coerced into formats necessary for modeling.

3.2 Random Forest Model

After fitting the model on the training data and evaluating the model on the testing data, we got an accuracy score of 0.459. Generally, above 70% is considered a valuable and realistic model. Our low accuracy score suggests that our model did not effectively capture the factors that predict a voter's party preference.

```python
Python
# By Axl
selectedColumns = [
'Ethnic_Description', 'Voters_Gender', 'Voters_Age' ,
'CommercialData_EstHomeValue','CommercialData_EstimatedHHIncomeAmount','Commerc
ialData_EstimatedAreaMedianHHIncome','CommercialData_AreaMedianHousingValue',
'Parties_Description',"Mailing_Families_HHCount","Residence_Families_HHCount",
'CommercialData_AreaMedianEducationYears', 'CommercialData_PropertyType',
'CommercialData_StateIncomeDecile', 'County','Mailing_HHGender_Description']

categoricalColumns =
['County','Mailing_HHGender_Description',"Ethnic_Description",
"Voters_Gender",
'CommercialData_PropertyType','CommercialData_StateIncomeDecile']
```

```
columns_with_dollar_signs =
['CommercialData_EstHomeValue','CommercialData_EstimatedHHIncomeAmount','Commer
cialData_EstimatedAreaMedianHHIncome','CommercialData_AreaMedianHousingValue']

numeric_columns_in_data_not_dollar_sign =
["Voters_Age","Mailing_Families_HHCount","Residence_Families_HHCount",'Commerci
alData_AreaMedianEducationYears']

for column in columns_with_dollar_signs:
    df_merged = df_merged.withColumn(column,
                                     regexp_replace(col(column), '[$,]',
'').cast('float'))

# Convert columns to numeric type
for column in numeric_columns_in_data_not_dollar_sign:
    df_merged = df_merged.withColumn(column, col(column).cast('float'))


start_time = time.time()

indexers = [StringIndexer(inputCol=column, outputCol=column+"_index",
handleInvalid='keep') for column in categoricalColumns]

assembler = VectorAssembler(
    inputCols=[indexer.getOutputCol() for indexer in indexers] +
numeric_columns_in_data_not_dollar_sign + columns_with_dollar_signs,
    outputCol="features"
)

(trainData, testData) = df_merged.randomSplit([0.7, 0.3], seed=123)

labelIndexer = StringIndexer(inputCol="Parties_Description", outputCol="label")

randomForest = RandomForestClassifier(labelCol="label", featuresCol="features",
numTrees=10,maxBins=100)

pipeline = Pipeline(stages=indexers + [labelIndexer, assembler, randomForest])

# FOR THE FOLLOWING CODE, YOU MUST RUN IT IN ON A CLUSTER ON GOOGLE COULD.
# YOU WILL RUN OUT OF MEMORY RUNNING LOCALLY (java.lang.OutOfMemoryError: Java
heap space)

model = pipeline.fit(trainData)
```

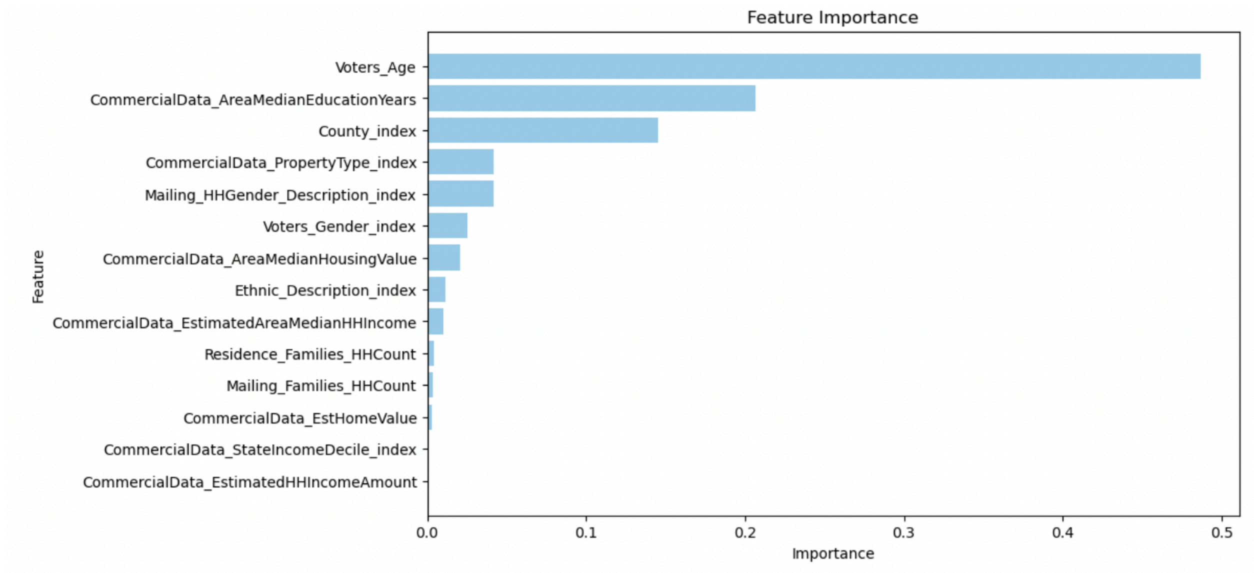
```
# End time
end_time = time.time()

elapsed_time_seconds = end_time - start_time

elapsed_time_minutes = elapsed_time_seconds / 60

print("Time taken:", elapsed_time_minutes, "minutes")
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

start_time = time.time()

predictions = model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:", accuracy)

end_time = time.time()

elapsed_time_seconds = end_time - start_time

elapsed_time_minutes = elapsed_time_seconds / 60

print("Time taken:", elapsed_time_minutes, "minutes")
```

3.3 Feature Importance (Random Forest)

Using our Random Forest model, we next looked at what features were most important to the model.

```python
# By Axl
rf_model = model.stages[-1]

feature_importance = rf_model.featureImportances.toArray()

feature_cols = model.stages[-2].getInputCols()

feature_df = pd.DataFrame({"Feature": feature_cols, "Importance":
feature_importance})
```

```
feature_df = feature_df.sort_values(by="Importance", ascending=False)

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.barh(feature_df["Feature"], feature_df["Importance"], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest importance at the
top
plt.show()
```



The Random Forest Feature Importance analysis shows that voter age, We see that a voter's age, the median number of years of education in a voter’s area, and their county are the most important variables among those we studied in predicting a person’s voting preference. Less important is voter the type of property a voter lives in, voter gender, and their ethnic description. Least important is estimated area median income, total number of voters residing at the given residence address, estimated home value, state income decile, and estimated income.

3.4 Multinomial Logistic Regression

Next, we’re going to fit a multinomial logistic regression to see how it compares with a Random Forest.

```
Python
# Axl and Alex
# categoricalColumns = ["County", "Mailing_Addresses_City",
"Ethnic_Description", "Voters_Gender", "Mailing_HHGender_Description",
"CommercialData_PropertyType", "CommercialData_StateIncomeDecile"]
indexers = [StringIndexer(inputCol=column, outputCol=column+"_index",
handleInvalid='keep') for column in categoricalColumns if column !=
'Parties_Description']


# numeric_columns_in_data_not_dollar_sign = ["Voters_Age",
"Mailing_Families_HHCount", "Residence_Families_HHCount",
"CommercialData_AreaMedianEducationYears"]

one_hot_encoder_list = []
for indexer in indexers:
    ohe_col = "{0}_encoded".format(indexer.getOutputCol())

    one_hot_encoder = OneHotEncoder(inputCol=indexer.getOutputCol(),
outputCol=ohe_col)
    one_hot_encoder_list.append(one_hot_encoder)


assembler = VectorAssembler(
    inputCols=[indexer.getOutputCol() for indexer in indexers] +
numeric_columns_in_data_not_dollar_sign + columns_with_dollar_signs,
    outputCol="features"
)


featureArr = [index.getOutputCol() for index in indexers] + [('scaled_' + f)
for f in numeric_columns_in_data_not_dollar_sign + columns_with_dollar_signs]

va1 = [VectorAssembler(inputCols = [f], outputCol=('vec_' + f)) for f in
numeric_columns_in_data_not_dollar_sign + columns_with_dollar_signs]
ss = [StandardScaler(inputCol="vec_" + f, outputCol="scaled_" + f,
withMean=True, withStd=True) for f in numeric_columns_in_data_not_dollar_sign +
columns_with_dollar_signs]

va2 = VectorAssembler(inputCols=featureArr, outputCol="features")
lr = LogisticRegression()



labelIndexer = StringIndexer(inputCol="Parties_Description", outputCol="label")
```

```
stages = va1 + ss + indexers + [va2, labelIndexer, lr]

log_pipeline = Pipeline(stages=stages)

(trainData, testData) = df_merged.randomSplit([0.7, 0.3], seed=123)

log_model = log_pipeline.fit(trainData)

# log_reg = LogisticRegression(labelCol="label", featuresCol="scaled_features")

# log_pipeline = Pipeline(stages=indexers + [scaler, labelIndexer, assembler,
log_reg])

# log_model = log_pipeline.fit(trainData)


# End time
end_time = time.time()

# Calculate elapsed time in seconds
elapsed_time_seconds = end_time - start_time

# Convert elapsed time to minutes
elapsed_time_minutes = elapsed_time_seconds / 60

print("Time taken:", elapsed_time_minutes, "minutes")


# Start time
start_time = time.time()

predictions = log_model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:", accuracy)



# End time
end_time = time.time()
```

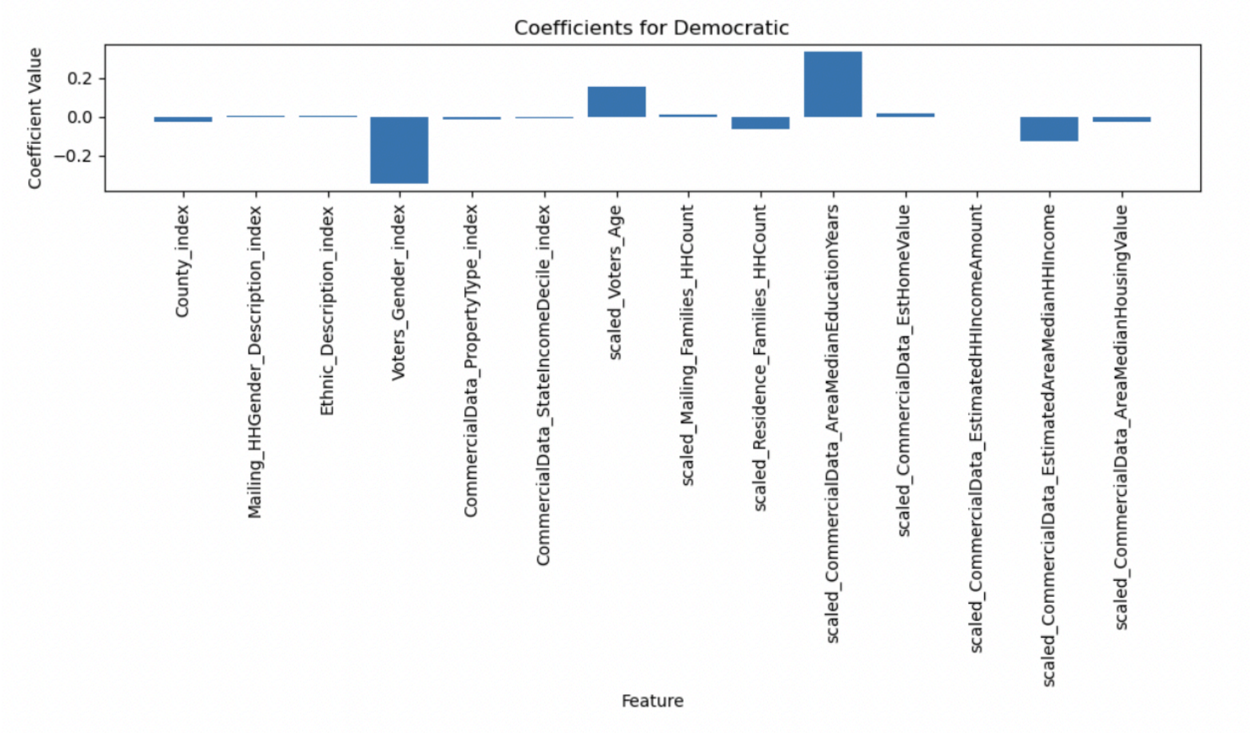
```
# Calculate elapsed time in seconds
elapsed_time_seconds = end_time - start_time

# Convert elapsed time to minutes
elapsed_time_minutes = elapsed_time_seconds / 60

print("Time taken:", elapsed_time_minutes, "minutes")
```

After fitting the model on the training data and evaluating the model on the testing data, we got an accuracy score of 0.461. It performs about the same as the random forest model with a difference in accuracy of 0.002 between the models. This suggests that both perform poorly. The low accuracy score indicates that neither model could effectively use the input variables to predict voter party preference.

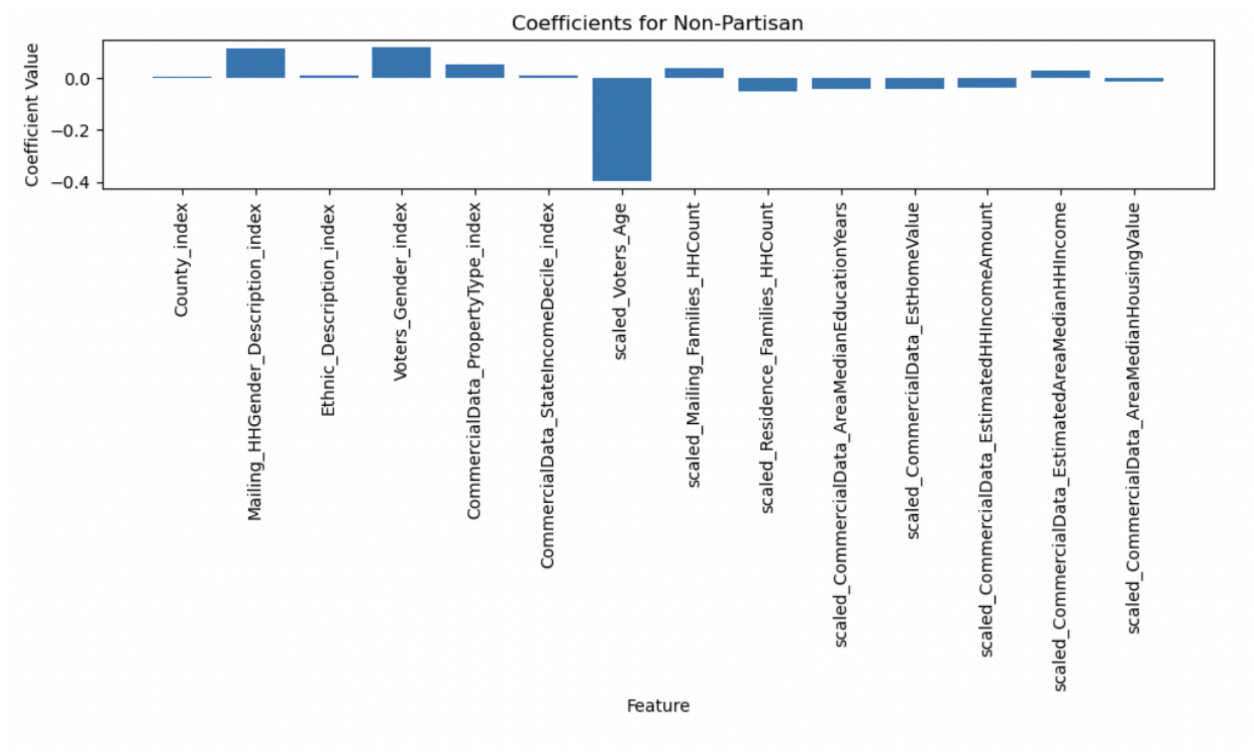
Now, we will look at the feature importance for the multinomial logistic regression to see which variables are most important for which parties.

3.5 Feature Importance (Multinomial LR)

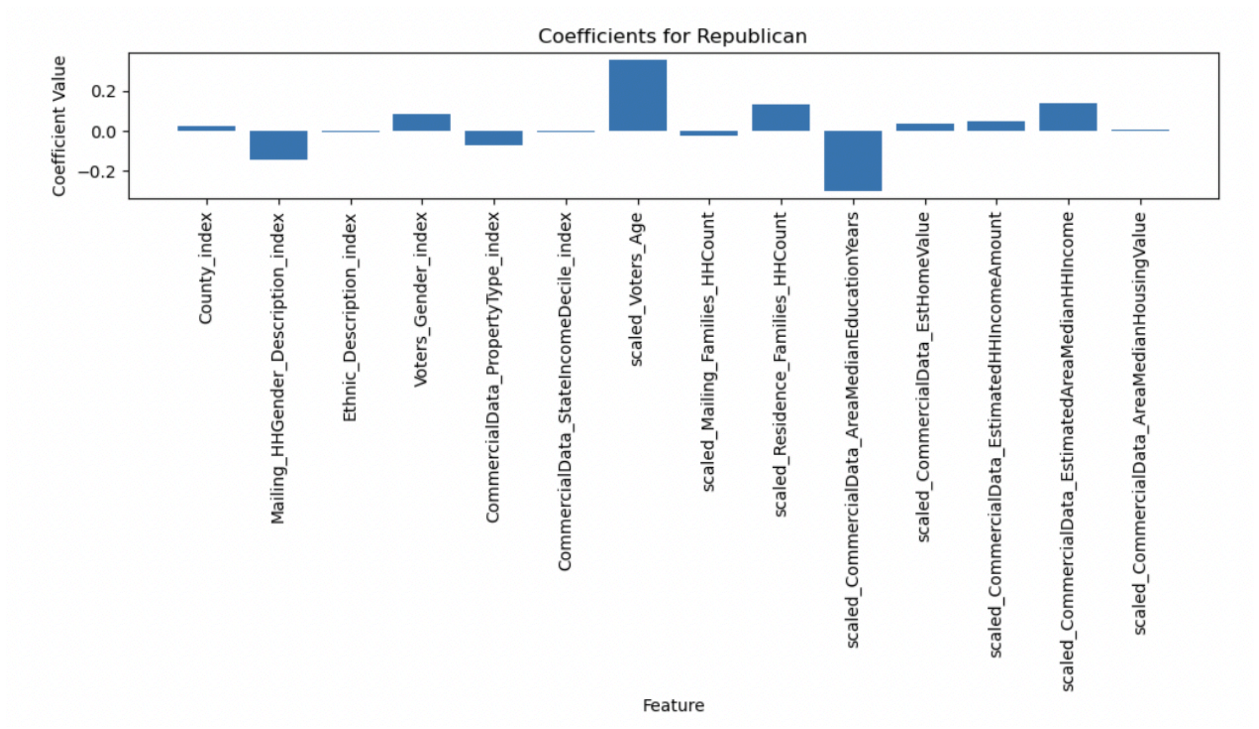
```python
# By Axl
orderedPartyList = ['Democratic','Non-Partisan','Republican','Other']  # This order comes from TA Cyrus. It is in this order because apparently it goes descending by count for each party in df_merged.  -Cyrus


coefficients = log_model.stages[-1].coefficientMatrix.toArray()

feature_names = log_model.stages[-3].getInputCols()


# Create 4 bar plots
for i in range(4):
    plt.figure(figsize=(10, 6))
    plt.bar(feature_names, coefficients[i])
    plt.xlabel('Feature')
    plt.ylabel('Coefficient Value')
    plt.title('Coefficients for {}'.format(orderedPartyList[i]))
    plt.xticks(rotation=90)
    plt.tight_layout()
plt.show()
```



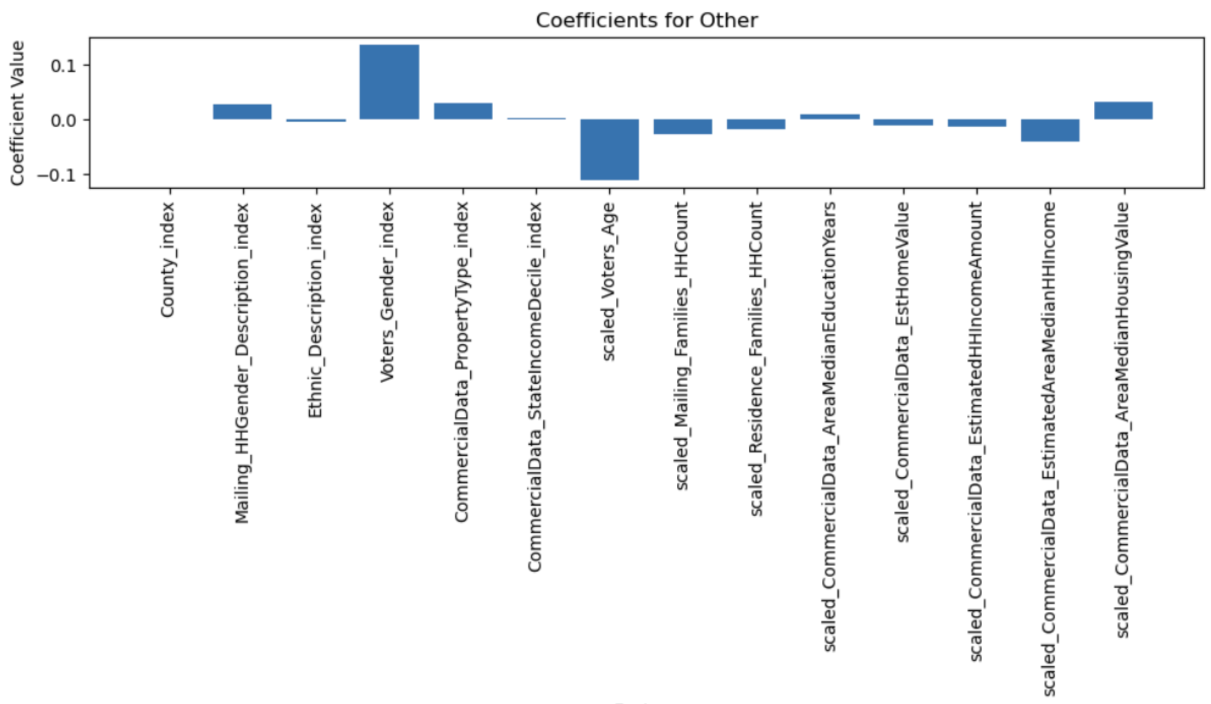
Looking at the plot, we see that being female is associated with being a member of the Democratic party. We also see that being older tends towards being a member of the Democratic party. Also, being more educated is associated with being part of the Democratic party. The other variables do not seem to have a strong correlation with being part of the Democratic party.



The plot suggests that being younger is the strongest indicator of being non-partisan. It also skews someone towards being male. The other variables are less important.



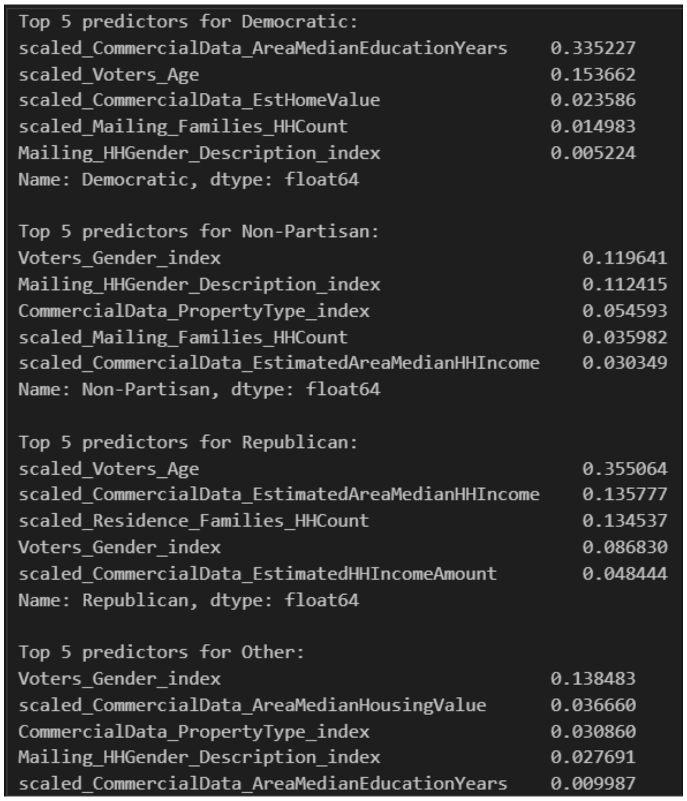
The plot suggests that being older and having less education is associated with being part of the Republican party. Also, having a higher area median income is associated with Republicans. The other variables are less important. Gender isn’t as important for predicting Republicans party membership as for Democrat.



Being male is associated with being part of Other parties besides Democrat and Republican. Also, being younger is associated with being part of "Other" parties as well. Other variables are not as important.

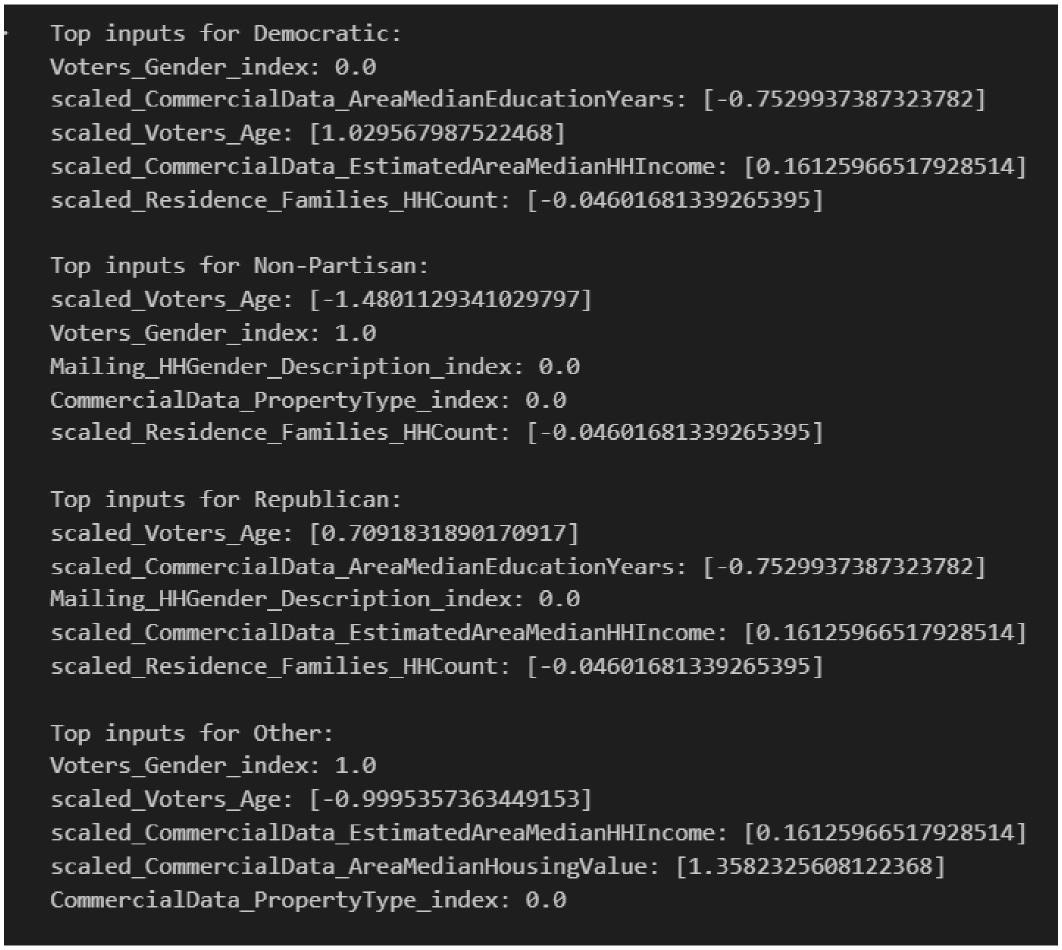
Getting the top input for each of the top 5 predictors given a party preference

```Python
coefficients_df = pd.DataFrame(coefficients, columns=feature_names,
index=orderedPartyList)

top_predictors = {}
for party in orderedPartyList:
  top_predictors[party] = coefficients_df.loc[party].nlargest(5)

for party, predictors in top_predictors.items():
  print(f"Top 5 predictors for {party}:")
  print(predictors)
  print()
```


```
Top 5 predictors for Democratic:
scaled_CommercialData_AreaMedianEducationYears    0.335227
scaled_Voters_Age                                 0.153662
scaled_CommercialData_EstHomeValue                0.023586
scaled_Mailing_Families_HHCount                   0.014983
Mailing_HHGender_Description_index                0.005224
Name: Democratic, dtype: float64

Top 5 predictors for Non-Partisan:
Voters_Gender_index                                  0.119641
Mailing_HHGender_Description_index                   0.112415
CommercialData_PropertyType_index                    0.054593
scaled_Mailing_Families_HHCount                      0.035982
scaled_CommercialData_EstimatedAreaMedianHHIncome    0.030349
Name: Non-Partisan, dtype: float64

Top 5 predictors for Republican:
scaled_Voters_Age                                    0.355064
scaled_CommercialData_EstimatedAreaMedianHHIncome    0.135777
scaled_Residence_Families_HHCount                    0.134537
Voters_Gender_index                                  0.086830
scaled_CommercialData_EstimatedHHIncomeAmount        0.048444
Name: Republican, dtype: float64

Top 5 predictors for Other:
Voters_Gender_index                               0.138483
scaled_CommercialData_AreaMedianHousingValue      0.036660
CommercialData_PropertyType_index                 0.030860
Mailing_HHGender_Description_index                0.027691
scaled_CommercialData_AreaMedianEducationYears    0.009987
```

Finding the top input for each predictor within each party group (standardized)

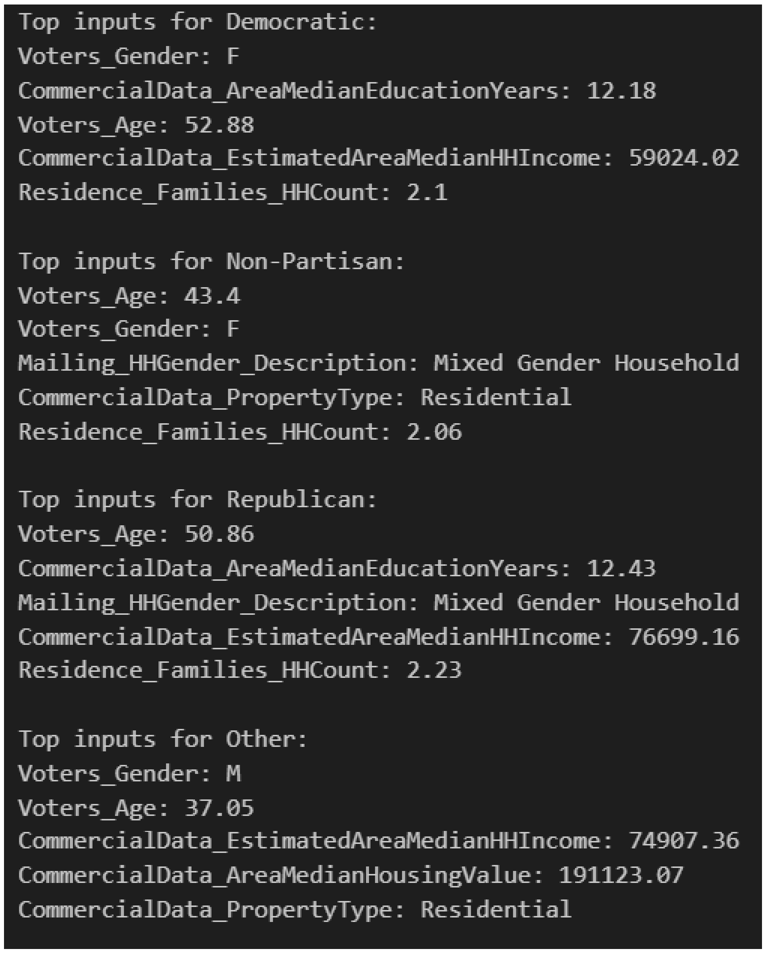
```python
# By Axl
# Find the top input for each predictor within each party group (standardized)
def find_top_inputs(predictions, top_predictors):
    top_inputs = {}

    for party, predictors in top_predictors.items():
        top_inputs[party] = {}

        party_df = predictions.filter(predictions["Parties_Description"] ==
party)

        for predictor in predictors.index:
            # Group by the predictor and count occurrences of each input value
            grouped_df = party_df.groupBy(predictor).count()

            # Order by count in descending order
            ordered_df = grouped_df.orderBy("count", ascending=False)
            # Get the top input for the predictor
            top_input = ordered_df.first()[predictor]

            # Store the top input for the predictor
            top_inputs[party][predictor] = top_input

    return top_inputs

top_inputs = find_top_inputs(predictions, top_predictors)

for party, predictors in top_inputs.items():
    print(f"Top inputs for {party}:")
    for predictor, top_input in predictors.items():
        print(f"{predictor}: {top_input}")
    print()
```



```
Top inputs for Democratic:
Voters_Gender_index: 0.0
scaled_CommercialData_AreaMedianEducationYears: [-0.7529937387323782]
scaled_Voters_Age: [1.029567987522468]
scaled_CommercialData_EstimatedAreaMedianHHIncome: [0.16125966517928514]
scaled_Residence_Families_HHCount: [-0.04601681339265395]

Top inputs for Non-Partisan:
scaled_Voters_Age: [-1.4801129341029797]
Voters_Gender_index: 1.0
Mailing_HHGender_Description_index: 0.0
CommercialData_PropertyType_index: 0.0
scaled_Residence_Families_HHCount: [-0.04601681339265395]

Top inputs for Republican:
scaled_Voters_Age: [0.7091831890170917]
scaled_CommercialData_AreaMedianEducationYears: [-0.7529937387323782]
Mailing_HHGender_Description_index: 0.0
scaled_CommercialData_EstimatedAreaMedianHHIncome: [0.16125966517928514]
scaled_Residence_Families_HHCount: [-0.04601681339265395]

Top inputs for Other:
Voters_Gender_index: 1.0
scaled_Voters_Age: [-0.9995357363449153]
scaled_CommercialData_EstimatedAreaMedianHHIncome: [0.16125966517928514]
scaled_CommercialData_AreaMedianHousingValue: [1.3582325608122368]
CommercialData_PropertyType_index: 0.0
```

Finding the top input for each predictor within each party group (True values)
Do note, the true value conversions were hard coded for this run

```
Python
# By Axl
from pyspark.sql.types import IntegerType, DoubleType, FloatType
top_5_unscaled_predictors_Democratic = ['Voters_Gender',
'CommercialData_AreaMedianEducationYears', 'Voters_Age',
'CommercialData_EstimatedAreaMedianHHIncome', 'Residence_Families_HHCount']
top_5_unscaled_predictors_Non_Partisan = ['Voters_Age', 'Voters_Gender',
'Mailing_HHGender_Description', 'CommercialData_PropertyType',
'Residence_Families_HHCount']
top_5_unscaled_predictors_Republican = ['Voters_Age',
'CommercialData_AreaMedianEducationYears', 'Mailing_HHGender_Description',
'CommercialData_EstimatedAreaMedianHHIncome', 'Residence_Families_HHCount']
top_5_unscaled_predictors_Other = ['Voters_Gender', 'Voters_Age',
'CommercialData_EstimatedAreaMedianHHIncome',
'CommercialData_AreaMedianHousingValue', 'CommercialData_PropertyType']

# Axl - find the top input for each predictor within each party group
(standardized)
def find_top_inputs(predictions, top_predictors):
    top_inputs = {}

    # Iterate over each party preference
    for party, predictors in top_predictors.items():
        top_inputs[party] = {}

        # Filter predictions DataFrame for the current party
        party_df = predictions.filter(predictions["Parties_Description"] ==
party)

        # Iterate over the top predictors for the current party
        for predictor in predictors:
            # Check if the predictor is numerical
            if predictor in party_df.columns and
isinstance(party_df.schema[predictor].dataType, (IntegerType, DoubleType,
FloatType)):
                # Calculate the mean for the predictor
                mean_value = party_df.agg({predictor: "avg"}).collect()[0][0]
                # Round off the mean value to 2 decimal places
                mean_value = round(mean_value, 2)
                # Store the mean value for the predictor
                top_inputs[party][predictor] = mean_value
            else:
                # Group by the predictor and count occurrences of each input
value
                grouped_df = party_df.groupBy(predictor).count()

                # Order by count in descending order
                ordered_df = grouped_df.orderBy("count", ascending=False)

                # Get the top input for the predictor
                top_input = ordered_df.first()[predictor]

                # Store the top input for the predictor
                top_inputs[party][predictor] = top_input

    return top_inputs

# Define top predictors for each party preference
top_predictors = {
    'Democratic': top_5_unscaled_predictors_Democratic,
    'Non-Partisan': top_5_unscaled_predictors_Non_Partisan,
    'Republican': top_5_unscaled_predictors_Republican,
    'Other': top_5_unscaled_predictors_Other
}

# Call the function to find the top inputs for each party preference
top_inputs = find_top_inputs(predictions, top_predictors)

# Print the top inputs for each party preference and predictor
for party, predictors in top_inputs.items():
    print(f"Top inputs for {party}:")
    for predictor, top_input in predictors.items():
        print(f"{predictor}: {top_input}")
    print()
```



```
Top inputs for Democratic:
Voters_Gender: F
CommercialData_AreaMedianEducationYears: 12.18
Voters_Age: 52.88
CommercialData_EstimatedAreaMedianHHIncome: 59024.02
Residence_Families_HHCount: 2.1

Top inputs for Non-Partisan:
Voters_Age: 43.4
Voters_Gender: F
Mailing_HHGender_Description: Mixed Gender Household
CommercialData_PropertyType: Residential
Residence_Families_HHCount: 2.06

Top inputs for Republican:
Voters_Age: 50.86
CommercialData_AreaMedianEducationYears: 12.43
Mailing_HHGender_Description: Mixed Gender Household
CommercialData_EstimatedAreaMedianHHIncome: 76699.16
Residence_Families_HHCount: 2.23

Top inputs for Other:
Voters_Gender: M
Voters_Age: 37.05
CommercialData_EstimatedAreaMedianHHIncome: 74907.36
CommercialData_AreaMedianHousingValue: 191123.07
CommercialData_PropertyType: Residential
```

Here we see the most frequent categorical inputs and the mean numerical inputs for the top 5 features with respect to each party preference. The data comes from predictions of our logistic model, suggesting that these characteristics are most common for members of each party preference. Political parties can use this information to advertise to potential voters to gain more votes. For example, if advertisers from the Democratic party are trying to get more Democratic votes, our model says they should advertise to women around age 53 that have a median education of 12. However, note that due to low model accuracy, the model's suggestions are prone to being mistaken.

We found that running the multinomial logistic regression multiple times yielded vastly different results for which attributes were associated with which parties. This might happen because our accuracy is low, so there is more variance in the model each time we run it.

4. Generalizability

Now that we've fit the random forest and multinomial logistic models, let's see how well they do on other datasets. We chose to test the two models we trained on Oregon data on three other states. We chose Washington since it is Oregon's neighbor, so we infer similar accuracy since voter behavior might be similar in neighboring states. Our second state was Maine because it's far away from Oregon and we were wondering if the model would work in different states. Our last selected state is Louisiana since it is far away from Oregon and also very culturally different since it's in the Deep South.

We will use each state's data as testing data for the model trained on Oregon and compare accuracies between states and between models.

```python
# By Axl and Alex
DATA_DIRECTORY = ' ' #input link here
# We used the following datasets to test for generalizability
# WA - gs://winter-2024-voter-file/VM2Uniform/VM2Uniform--WA--2020-12-09/
# ME - gs://asdfasdgasdgasdgasdgasd/VM2Uniform--ME--2021-05-28/
# LA - gs://winter-2024-voter-file/VM2Uniform/VM2Uniform--LA--2021-01-22/

df = (
spark.read.format("parquet")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("{}".format(DATA_DIRECTORY))
)

from pyspark.sql.functions import col
selectedColumns = ['County', 'Ethnic_Description', 'Voters_Gender',
'Voters_Age', 'Mailing_Families_HHCount', 'Mailing_HHGender_Description',
'Residence_Families_HHCount', 'CommercialData_EstHomeValue',
'CommercialData_EstimatedHHIncomeAmount',
'CommercialData_EstimatedAreaMedianHHIncome',
'CommercialData_AreaMedianHousingValue',
'CommercialData_AreaMedianEducationYears', 'CommercialData_PropertyType',
'CommercialData_StateIncomeDecile','Parties_Description']

cleaned_df = df.select(selectedColumns)

data = cleaned_df.dropna()

df_merged = data
from pyspark.sql.functions import when, col, lit

parties_to_keep = ['Republican', 'Democratic', 'Non-Partisan']  # List of
parties to keep
```

```
df_merged = df_merged.withColumn("Parties_Description",
when(col("Parties_Description").isin(parties_to_keep),
col("Parties_Description")).otherwise(lit("Other")))

import pandas as pd

from pyspark.sql.functions import regexp_replace, col
df_merged = df_merged.withColumn('CommercialData_EstHomeValue',

regexp_replace(col('CommercialData_EstHomeValue'), '[$,]', '').cast('float'))

df_merged = df_merged.withColumn('CommercialData_EstimatedHHIncomeAmount',

regexp_replace(col('CommercialData_EstimatedHHIncomeAmount'), '[$,]',
'').cast('float'))

df_merged = df_merged.withColumn('CommercialData_EstimatedAreaMedianHHIncome',

regexp_replace(col('CommercialData_EstimatedAreaMedianHHIncome'), '[$,]',
'').cast('float'))

df_merged = df_merged.withColumn('CommercialData_AreaMedianHousingValue',

regexp_replace(col('CommercialData_AreaMedianHousingValue'), '[$,]',
'').cast('float'))

columns_to_convert = ["Voters_Age", "Mailing_Families_HHCount",
"Residence_Families_HHCount", "CommercialData_AreaMedianEducationYears"]

for column in columns_to_convert:
    df_merged = df_merged.withColumn(column, col(column).cast('float'))
```

4.1 Accuracy test on the random forest

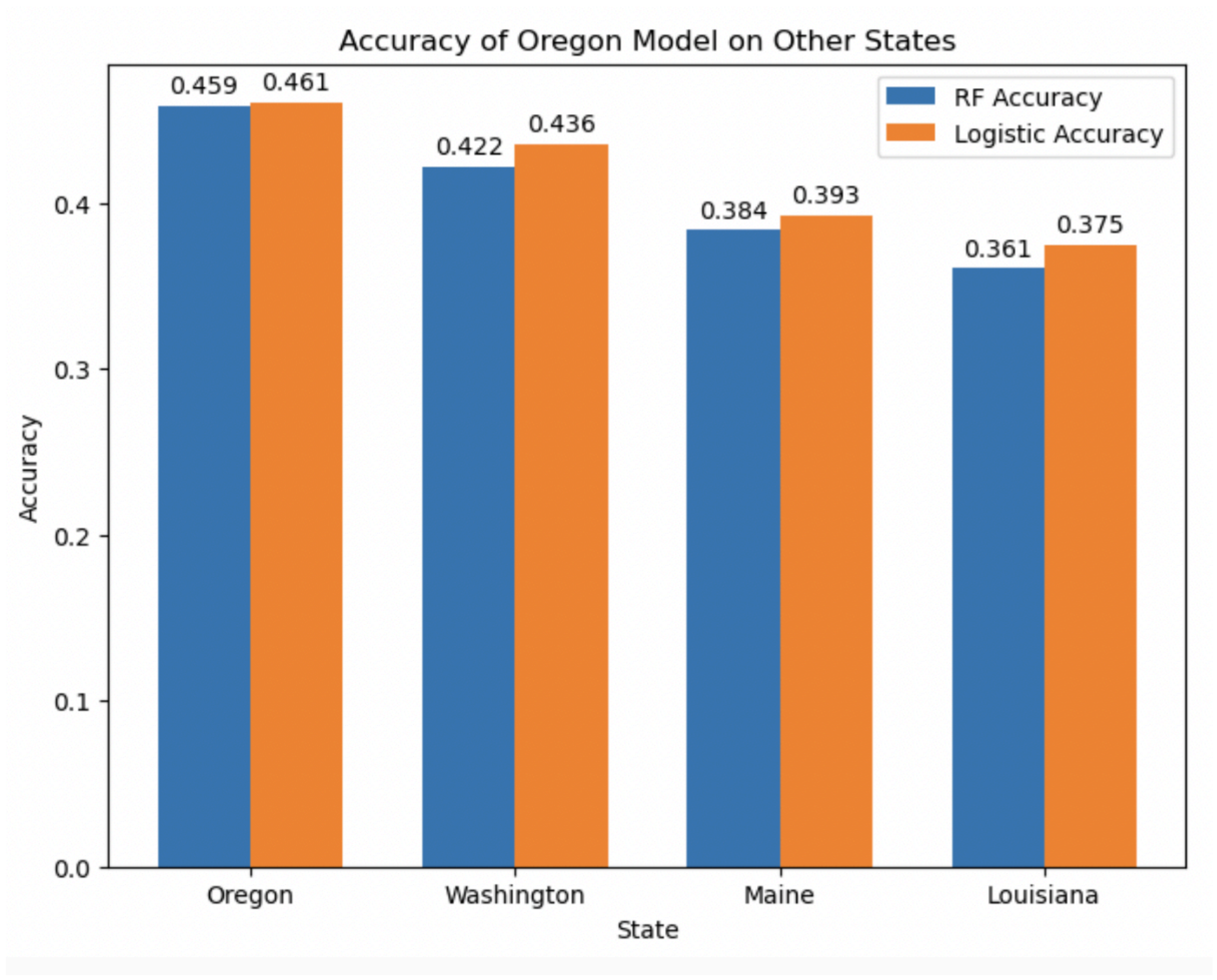
```
Python
testData = df_merged
predictions = model.transform(testData)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:", accuracy)
```

4.2 Accuracy test on the multinomial logistic regression

```python
predictions = log_model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:", accuracy)
```

```python
import matplotlib.pyplot as plt

# Data
states = ['Oregon', 'Washington', 'Maine', 'Louisiana']
rf_accuracies = [0.459, 0.422, 0.384, 0.361]
logistic_accuracies = [0.461, 0.436, 0.393, 0.375]

# Plotting
x = range(len(states))
width = 0.35

fig, ax = plt.subplots()
bars1 = ax.bar(x, rf_accuracies, width, label='RF Accuracy')
bars2 = ax.bar([i + width for i in x], logistic_accuracies, width, label='Logistic
Accuracy')

ax.set_xlabel('States')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of RF and Logistic Models by State')
ax.set_xticks([i + width/2 for i in x])
ax.set_xticklabels(states)

# Add data labels
def add_labels(bars):
  for bar in bars:
    height = bar.get_height()
    ax.annotate('{}'.format(height),
          xy=(bar.get_x() + bar.get_width() / 2, height),
          xytext=(0, 3),
          textcoords="offset points",
          ha='center', va='bottom')

add_labels(bars1)
add_labels(bars2)

# Move the legend outside
ax.legend(loc='upper left', bbox_to_anchor=(1, 1))

plt.show()
```



Looking at the plot, in terms of comparison between random forest and multinomial logistic regression, we see that they perform very similarly regardless of the state's data they're tested on. Also, we see that while the random forest and multinomial logistic regression perform about the same for the Oregon data, the logistic model outperforms the random forest by about 1% accuracy for the other states. This might suggest that the logistic model does better applied to new data (although a 1% accuracy difference might not be considered significant)

While comparing between states, we see that accuracy for both models is highest in Washington. This makes sense since Washington is physically closest to Oregon. Maine, which is across the country from Oregon, scores about 4% lower in accuracy. This might be because Maine is so far from Oregon. Louisiana has the lowest accuracy (6% lower than Washington), which makes sense since Louisiana is physically distant and the population attributes are quite different (income, party preference distribution, etc.)

In general, from these findings we can conclude that our model trained on the Oregon dataset, has an accuracy that decreases depending on its cultural and physical distance from Oregon. This makes sense as different voter attributes might be more important in other states, be it income, family size, etc. This can cause them to have more significant predictor coefficients that are not necessarily captured as well by the ones we have seen in Oregon's feature importance results.

5. Conclusion

In this project, we looked at what predicts voter party preference in the state of Oregon. For our Exploratory Data Analysis, we chose variables expected to impact voter party preference, and we explored and visualized the relationships. What we found from the graphs is that economic status, measured by household income levels and property type owned, generally displayed a weak relationship with party preferences, suggesting little to no impact. This was a little surprising as economics and taxes are usually a hot issue in political debates. On the contrary, educational level and age, each classified into their own distinct groups based on a criteria, displayed a clear difference in preferences, suggesting age and educational level plays a significant role in people's party preferences.

Next, we fitted models to the data. By fitting a random forest and multinomial logistic regression to 15 predictor variables, we attempted to evaluate what factors are most important in a person's party preference. Overall, both models did poorly with random forest and logistic accuracies of 0.459 and 0.461 respectively. This suggests that the models cannot accurately predict voter party preference in Oregon given the chosen variables.

Our random forest model found that a voter's age, their median number of years of education in their area, and their county are the most important variables among those we studied for predicting voter party affiliation. Less important variables included area median income, estimated home value, and state income decile.

Our multinomial logistic regression analyzed which factors are most important for each party. Some of its findings included that being female is associated with being part of the Democratic party, younger people tend to be part of "Other" parties, and that being older is associated with being Republican.

From our logistic regression predictions, we calculated the attributes of the average member of each party. This data can give campaigners for the Democratic party insight on which kinds of people to target for advertising. However, due to low model accuracy, the averages are often incorrect or misleading. We also found that while running our models, we would get vastly different results for which attributes were associated with which parties. This might occur due to low accuracy of the models.

We also found that our model is not very generalizable. After evaluating voting records from Washington, Maine, and Louisiana as testing data for our random forest and logistic regression trained on Oregon data, we found poor accuracy scores of around 0.42, 0.38, and 0.37, respectively for both models (with the logistic model having a 1% accuracy increase compared to the random forest). This suggests that the model does worse with increasing physical distance and difference in voter behavior and lifestyle. Also, note that even if the states we tested on remained very close in accuracy scores to the original Oregon models, the accuracy is still far too low to make meaningful conclusions.

In future research, we would like to explore how we can use the model data to convert potential voters into a party preference. Using the top input per most important feature column results, we calculated which factors are most important to each party, but low accuracy from the model gives us low confidence in these findings. Also, the models trained on Oregon data struggle to accurately predict voter preference in other states, as well.

One important question we'd love to answer in further research: how would we tune our parameters better to have a better fitting model? An approach that we did attempt was the hypertuning of the parameters which unfortunately resulted in an unreasonably long loading time before returning a timeout error. We could however re-evaluate the columns we chose to work with and the criterias we ended up choosing from. Another approach we could look at would be to instead survey a group of a certain party preference in real life to conduct an investigation on whether or not it fits the predictions we have gotten from our models. All these potential improvements allow our project to have a wide room for improvement.