PSTAT 135/235 Final Report

Siddharth Sabata\ Yang Hu\ Edwin Gao\ Arjun Sahasrabuddhe

Date: 03/20/2024

1.Introduction

Voter engagement is critical to the health of any democracy and a robust democracy relies on high levels of voter turnout. As the United States are having increasing economic disparities and diverse living conditions, understanding the factors that influence electoral participation is crucial. In this study we analyze the relevant data of Wyoming state towards finding answers to such questions. This research aims to dissect how household income levels and housing conditions—encompassing home ownership and housing costs—affect voter participation.

---

2. Dataset and Initial Data Cleaning

In [ ]:
```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Read Voter File Data") \
    .getOrCreate()

df = (
    spark.read
    .format("parquet")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("gs://winter-2024-voter-file/VM2Uniform/VM2Uniform--WY--2021-01-13")
)
```

Let's check the percentage of missing data in each column.

In [ ]:
```
from pyspark.sql.functions import col, count, when
import pandas as pd

total_rows = df.count()
missing_data_count = df.select([count(when(col(c).isNull(), c)).alias(c) for c
missing_data_percentage = missing_data_count.select([((col(c) / total_rows) * 

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

# display(missing_data_percentage.toPandas().T)
```

The table is hidden for visual purposes, but there are many columns with fully missing data. There are also plenty of columns with partially missing data. We can remove all these columns as we don't have enough data to study its impact on voter participation, and we still have enough columns leftover to further analyze.

Approach

1. Drop all features missing over 15% data
2. Drop unnecessary features
3. Drop unnecessary rows
4. Make sure all features are the correct type
5. Impute missing data with a random forest model

In [ ]:

```
"""
Step 1: Drop all features with more than 15% missing data
"""

from pyspark.sql.functions import col, count, lit

# Calculate the total number of rows in the DataFrame
total_rows = df.count()

# Collect the percentages into a Python dictionary for easier processing
missing_percentage_dict = missing_data_percentage.collect()[0].asDict()

# Identify columns with more than 15% missing values
columns_to_drop = [c for c, p in missing_percentage_dict.items() if p > 15]

# Drop the identified columns from the DataFrame
df_cleaned = df.drop(*columns_to_drop)

# View transposed head of the dataframe using Pandas
# display(df_cleaned.limit(5).toPandas().T)
```

Now we can see we dropped 502 features which had over 15% missing data.

Let's now drop the following additional features as these factors are not going to influence voter participation, for instance a voter's first name:

Additional features to drop
`Voters_StateVoterID`
`Voters_FirstName`
`Voters_MiddleName`
`Voters_LastName`
`Residence_Addresses_AddressLine`
`Residence_Addresses_Zip`
`Residence_Addresses_ZipPlus4`

Additional features to drop
`Residence_Addresses_HouseNumber`
`Residence_Addresses_StreetName`
`Residence_Addresses_Designator`
`Residence_Addresses_CassErrStatCode`
`Voters_SequenceZigZag`
`Voters_SequenceOddEven`
`Residence_Addresses_CensusTract`
`Residence_Addresses_CensusBlockGroup`
`Residence_Addresses_CensusBlock`
`Residence_Addresses_Latitude`
`Residence_Addresses_Longitude`
`Residence_Addresses_LatLongAccuracy`
`Residence_Addresses_Density`
`Mailing_Addresses_AddressLine`
`Mailing_Addresses_City`
`Mailing_Addresses_State:`
`Mailing_Addresses_Zip`
`Mailing_Addresses_ZipPlus4`
`Mailing_Addresses_StreetName`
`Mailing_Addresses_CassErrStatCode`
`Mailing_Families_FamilyID`
`Mailing_Families_HHCount`
`Mailing_HHGender_Description`
`Mailing_HHParties_Description`
`Voters_CalculatedRegDate`
`Voters_OfficialRegDate`
`AddressDistricts_Change_Changed_CD`
`AddressDistricts_Change_Changed_SD`
`2001_State_Senate_District`
`2001_State_House_District`
`Voters_FIPS`
`AddressDistricts_Change_Changed_County`
`County_Commissioner_District`

Additional features to drop
`Designated_Market_Area_DMA`
`Unified_School_District`
`CommercialData_ISPSA`
`CommercialData_MosaicZ4Global`
`CommercialData_StateIncomeDecile`
`ElectionReturns_G08_Cnty_Margin_McCain_R`
`ElectionReturns_G08_Cnty_Percent_McCain_R`
`ElectionReturns_G08_Cnty_Vote_McCain_R`
`ElectionReturns_G08_Cnty_Margin_Obama_D`
`ElectionReturns_G08_Cnty_Percent_Obama_D`
`ElectionReturns_G08_Cnty_Vote_Obama_D`
`ElectionReturns_G12_Cnty_Margin_Obama_D`
`ElectionReturns_G12_Cnty_Percent_Obama_D`
`ElectionReturns_G12_Cnty_Vote_Obama_D`
`ElectionReturns_G12_Cnty_Margin_Romney_R`
`ElectionReturns_G12_Cnty_Percent_Romney_R`
`ElectionReturns_G12_Cnty_Vote_Romney_R`
`ElectionReturns_G16_Cnty_Margin_Clinton_D`
`ElectionReturns_G16_Cnty_Percent_Clinton_D`
`ElectionReturns_G16_Cnty_Vote_Clinton_D`
`ElectionReturns_G16_Cnty_Margin_Trump_R`
`ElectionReturns_G16_Cnty_Percent_Trump_R`
`ElectionReturns_G16_Cnty_Vote_Trump_R`
`ElectionReturns_P08_Cnty_Pct_Clinton_D`
`ElectionReturns_P08_Cnty_Pct_Obama_D`
`ElectionReturns_P08_Cnty_Vote_Biden_D`
`ElectionReturns_P08_Cnty_Vote_Clinton_D`
`ElectionReturns_P08_Cnty_Vote_Dodd_D`
`ElectionReturns_P08_Cnty_Vote_Edwards_D`
`ElectionReturns_P08_Cnty_Vote_Gravel_D`
`ElectionReturns_P08_Cnty_Vote_Kucinich_D`
`ElectionReturns_P08_Cnty_Vote_Obama_D`
`ElectionReturns_P08_Cnty_Vote_Richardson_D`

Additional features to drop
`ElectionReturns_P12_Cnty_Pct_Gingrich_R`
`ElectionReturns_P12_Cnty_Pct_Paul_R`
`ElectionReturns_P12_Cnty_Pct_Romney_R`
`ElectionReturns_P12_Cnty_Pct_Santorum_R`
`ElectionReturns_P12_Cnty_Vote_Bachman_R`
`ElectionReturns_P12_Cnty_Vote_Gingrich_R`
`ElectionReturns_P12_Cnty_Vote_Huntsman_R`
`ElectionReturns_P12_Cnty_Vote_Paul_R`
`ElectionReturns_P12_Cnty_Vote_Perry_R`
`ElectionReturns_P12_Cnty_Vote_Romney_R`
`ElectionReturns_P12_Cnty_Vote_Santorum_R`
`ElectionReturns_P16_Cnty_Pct_Bush_R`
`ElectionReturns_P16_Cnty_Pct_Carson_R`
`ElectionReturns_P16_Cnty_Pct_Christie_R`
`ElectionReturns_P16_Cnty_Pct_Cruz_R`
`ElectionReturns_P16_Cnty_Pct_Fiorina_R`
`ElectionReturns_P16_Cnty_Pct_Kasich_R`
`ElectionReturns_P16_Cnty_Pct_Rubio_R`
`ElectionReturns_P16_Cnty_Pct_Trump_R`
`ElectionReturns_P16_Cnty_Vote_Bush_R`
`ElectionReturns_P16_Cnty_Vote_Carson_R`
`ElectionReturns_P16_Cnty_Vote_Christie_R`
`ElectionReturns_P16_Cnty_Vote_Cruz_R`
`ElectionReturns_P16_Cnty_Vote_Fiorina_R`
`ElectionReturns_P16_Cnty_Vote_Kasich_R`
`ElectionReturns_P16_Cnty_Vote_Rubio_R`
`ElectionReturns_P16_Cnty_Vote_Trump_R`
`ElectionReturns_P16_Cnty_Pct_Clinton_D`
`ElectionReturns_P16_Cnty_Pct_Sanders_D`
`ElectionReturns_P16_Cnty_Vote_Clinton_D`
`ElectionReturns_P16_Cnty_Vote_Sanders_D`

In [ ]:

```
"""
Step 2: Drop unnecessary features
```

```
"""
# List of columns to be dropped
columns_to_drop = [
    "Voters_StateVoterID", "Voters_FirstName", "Voters_MiddleName", "Voters_Las
    "Residence_Addresses_AddressLine", "Residence_Addresses_Zip", "Residence_Ad
    "Residence_Addresses_HouseNumber", "Residence_Addresses_StreetName", "Resid
    "Residence_Addresses_CassErrStatCode", "Voters_SequenceZigZag", "Voters_Seq
    "Residence_Addresses_CensusTract", "Residence_Addresses_CensusBlockGroup",
    "Residence_Addresses_Latitude", "Residence_Addresses_Longitude", "Residence
    "Residence_Addresses_Density", "Mailing_Addresses_AddressLine", "Mailing_Ad
    "Mailing_Addresses_State:", "Mailing_Addresses_Zip", "Mailing_Addresses_Zip
    "Mailing_Addresses_StreetName", "Mailing_Addresses_CassErrStatCode", "Maili
    "Mailing_Families_HHCount", "Mailing_HHGender_Description", "Mailing_HHPart
    "Voters_CalculatedRegDate", "Voters_OfficialRegDate", "AddressDistricts_Cha
    "AddressDistricts_Change_Changed_SD", "2001_State_Senate_District", "2001_S
    "Voters_FIPS", "AddressDistricts_Change_Changed_County", "County_Commission
    "Designated_Market_Area_DMA", "Unified_School_District", "CommercialData_IS
    "CommercialData_MosaicZ4Global", "CommercialData_StateIncomeDecile", "Elect
    "ElectionReturns_G08_Cnty_Percent_McCain_R", "ElectionReturns_G08_Cnty_Vote
    "ElectionReturns_G08_Cnty_Margin_Obama_D", "ElectionReturns_G08_Cnty_Percen
    "ElectionReturns_G08_Cnty_Vote_Obama_D", "ElectionReturns_G12_Cnty_Margin_O
    "ElectionReturns_G12_Cnty_Percent_Obama_D", "ElectionReturns_G12_Cnty_Vote_
    "ElectionReturns_G12_Cnty_Margin_Romney_R", "ElectionReturns_G12_Cnty_Perce
    "ElectionReturns_G12_Cnty_Vote_Romney_R", "ElectionReturns_G16_Cnty_Margin_
    "ElectionReturns_G16_Cnty_Percent_Clinton_D", "ElectionReturns_G16_Cnty_Vot
    "ElectionReturns_G16_Cnty_Margin_Trump_R", "ElectionReturns_G16_Cnty_Percen
    "ElectionReturns_G16_Cnty_Vote_Trump_R", "ElectionReturns_P08_Cnty_Pct_Clin
    "ElectionReturns_P08_Cnty_Pct_Obama_D", "ElectionReturns_P08_Cnty_Vote_Bide
    "ElectionReturns_P08_Cnty_Vote_Clinton_D", "ElectionReturns_P08_Cnty_Vote_D
    "ElectionReturns_P08_Cnty_Vote_Edwards_D", "ElectionReturns_P08_Cnty_Vote_G
    "ElectionReturns_P08_Cnty_Vote_Kucinich_D", "ElectionReturns_P08_Cnty_Vote_
    "ElectionReturns_P08_Cnty_Vote_Richardson_D", "ElectionReturns_P12_Cnty_Pct
    "ElectionReturns_P12_Cnty_Pct_Paul_R", "ElectionReturns_P12_Cnty_Pct_Romney
    "ElectionReturns_P12_Cnty_Pct_Santorum_R", "ElectionReturns_P12_Cnty_Vote_B
    "ElectionReturns_P12_Cnty_Vote_Gingrich_R", "ElectionReturns_P12_Cnty_Vote_
    "ElectionReturns_P12_Cnty_Vote_Paul_R", "ElectionReturns_P12_Cnty_Vote_Per
    "ElectionReturns_P12_Cnty_Vote_Romney_R", "ElectionReturns_P12_Cnty_Vote_Sa
    "ElectionReturns_P16_Cnty_Pct_Bush_R", "ElectionReturns_P16_Cnty_Pct_Carson
    "ElectionReturns_P16_Cnty_Pct_Christie_R", "ElectionReturns_P16_Cnty_Pct_C
    "ElectionReturns_P16_Cnty_Pct_Fiorina_R", "ElectionReturns_P16_Cnty_Pct_Kas
    "ElectionReturns_P16_Cnty_Pct_Rubio_R", "ElectionReturns_P16_Cnty_Pct_Trump
    "ElectionReturns_P16_Cnty_Vote_Bush_R", "ElectionReturns_P16_Cnty_Vote_Cars
    "ElectionReturns_P16_Cnty_Vote_Christie_R", "ElectionReturns_P16_Cnty_Vote_
    "ElectionReturns_P16_Cnty_Vote_Fiorina_R", "ElectionReturns_P16_Cnty_Vote_H
    "ElectionReturns_P16_Cnty_Vote_Rubio_R", "ElectionReturns_P16_Cnty_Vote_Tru
    "ElectionReturns_P16_Cnty_Pct_Clinton_D", "ElectionReturns_P16_Cnty_Pct_San
    "ElectionReturns_P16_Cnty_Vote_Clinton_D", "ElectionReturns_P16_Cnty_Vote_S
    "General_2020", "CommercialData_EstimatedHHIncome"
]

# Drop the columns from the DataFrame
df_drop = df_cleaned.drop(*columns_to_drop)

# Show the difference
print(str(len(df_drop.columns)) + ', ' + str(len(df_cleaned.columns)))
```

Here, we can see that 119 columns were dropped out of 224 columns leaving us with 105 columns, which is still a lot to work with.

With this tidied dataset, let's check for the count of missing values.

In [ ]:
```
from pyspark.sql import DataFrame
from pyspark.sql.functions import col, sum as sum_

def print_missing_value_counts(df: DataFrame):
    """
    Prints the count of missing values for each column in the given PySpark Da

    Parameters:
    - df: A PySpark DataFrame.
    """
    # Create an expression that sums up the null instances for each column
    missing_value_counts = [sum_(col(c).isNull().cast("int")).alias(c) for c in

    # Apply the aggregation and collect the results
    missing_counts_df = df.select(missing_value_counts).collect()[0]

    # Print the missing value count for each column
    for column in df.columns:
        print(f"Missing values in {column}: {missing_counts_df[column]}")

# print_missing_value_counts(df_drop)
```

Once again, the output is hidden for visual purposes. Here are the actions we're going to take based off of the missing value counts for each feature.

We are going to drop the 398 rows with missing precinct data since there are 290,408 rows total. Dropping ~400 rows is not going to make much of a difference to this dataset.

We're also going to drop the 29184 rows of missing ethnic data. This is fair amount of data that is going to be lost, but there is no other way to deal with it.

We're going to drop the 3011 missing gender values. It is impossible to impute it, and doesn't make much of a difference since there are 290,408 total rows. Same goes for `Judicial_District`, `State_House_District`, and `State_Senate_District`.

We will impute the missing data for the following features, as these will be important for our research question: `CommercialData_EstHomeValue`, `CommercialData_EstimatedHHIncomeAmount`, `CommercialData_EstimatedAreaMedianHHIncome`, `CommercialData_AreaMedianEducationYears`, `CommercialData_AreaMedianHousingValue`, `CommercialData_AreaPcntHHMarriedCoupleNoChild`, `CommercialData_AreaPcntHHMarriedCoupleWithChild`, `CommercialData_AreaPcntHHSpanishSpeaking`, and `CommercialData_AreaPcntHHWithChildren`.

In [ ]:
```
"""
Step 3: Drop unnecessary rows
"""
# Drop missing ethnic values
```

```
df_drop_1 = df_drop.filter(df_drop["Ethnic_Description"].isNotNull())

# Drop precinct data
df_drop_2 = df_drop_1.filter(df_drop_1["Precinct"].isNotNull())

# Drop gender data
df_drop_3 = df_drop_2.filter(df_drop_1["Voters_Gender"].isNotNull())

# Drop judicial discrict data
df_drop_4 = df_drop_3.filter(df_drop_1["Judicial_District"].isNotNull())

# Drop state house discrict data
df_drop_5 = df_drop_4.filter(df_drop_1["State_House_District"].isNotNull())

# Drop state senate district data
df_drop_final = df_drop_5.filter(df_drop_1["State_Senate_District"].isNotNull(

# View missing value counts again
# print_missing_value_counts(df_drop_final)
```

We now have a slight problem. All of the features are strings, so we need to go through and resassign the types accordingly.

In [ ]:

```
"""
Step 4: Make sure all features are the correct type
"""

from pyspark.sql import DataFrame
from pyspark.sql.functions import col, regexp_replace
from pyspark.sql.types import FloatType

def infer_and_convert_column_types(df: DataFrame):
    """
    Attempts to clean and convert column types in a PySpark DataFrame from str
    after removing specific characters ('$' and '%') that can hinder numeric co
    This function iterates through each column, removes '$' and '%' characters
    checks if the cleaned values are numeric, and attempts to convert them to 
    Columns that contain non-numeric values after cleaning are kept as strings

    Parameters:
    - df: A PySpark DataFrame with columns potentially containing numeric value

    Returns:
    - A PySpark DataFrame with columns converted to float type where applicable
    """
    # This pattern matches strings that are potentially numeric, ignoring '%' a
    numeric_pattern = "^-?\d*\.?\d+%?$"

    for column_name in df.columns:
        # Remove percentage signs and dollar signs and then attempt to convert
        cleaned_column = regexp_replace(regexp_replace(col(column_name), "%", '
        # Select a sample of the data to test conversion (to avoid scanning ve
        sample = df.select(cleaned_column).limit(1000).toPandas()

        # Check if all sampled values in the column match the numeric pattern
        if sample[column_name].str.match(numeric_pattern).all():
            # Attempt to convert the entire column to float, since the sample 
            df = df.withColumn(column_name, cleaned_column.cast(FloatType()))
            print(f"Column {column_name} converted to FloatType.")
```

```
        else:
            # If any value does not match the numeric pattern, keep as is (str
            print(f"Column {column_name} contains non-numeric values, kept as S

    return df

# Apply the function to your DataFrame
df_adj_regex = infer_and_convert_column_types(df_drop_final)

# Deal with `Voters_VotingPerformanceEvenYearPrimary` and `Voters_VotingPerfor
columns_to_convert = ["Voters_VotingPerformanceEvenYearPrimary", "Voters_Voting

# Convert "Not eligible" to null and remove any non-numeric characters (e.g.,
for column in columns_to_convert:
    df_adj_regex = df_adj_regex.withColumn(column, regexp_replace(col(column),
    df_adj_regex = df_adj_regex.withColumn(column, regexp_replace(col(column),
    df_adj_regex = df_adj_regex.withColumn(column, col(column).cast(FloatType(

# Drop NaN rows from the dataframe that came up from this process
df_adj_regex_1 = df_adj_regex.filter(df_adj_regex["Voters_VotingPerformanceEve
df_adj_regex_2 = df_adj_regex_1.filter(df_adj_regex_1["Voters_VotingPerformanc

# Display new dataframe
# display(df_adj_regex_2.limit(5).toPandas().T)
```

Now that we have corrected the types of all of our features, we can begin to impute our values

In [ ]:
```
# """
# Step 5: Impute missing data with a random forest model
# """
# from pyspark.sql import DataFrame
# from pyspark.ml import Pipeline
# from pyspark.ml.feature import VectorAssembler
# from pyspark.ml.regression import RandomForestRegressor
# from pyspark.sql.functions import col, lit
# from pyspark.sql.types import IntegerType, FloatType, DoubleType
# from pyspark.sql.functions import monotonically_increasing_id

# def impute_with_random_forest(df: DataFrame, feature: str, input_features: li
#     """
#     Imputes missing values for a given feature in a DataFrame using a Random

#     Assumes there's a unique 'SEQUENCE' column for each row used for joining

#     Parameters:
#     - df: A PySpark DataFrame.
#     - feature: The name of the feature (column) to impute missing values for
#     - input_features: A list of column names to use as input features for the

#     Returns:
#     - A DataFrame with missing values in the specified feature imputed.
#     """
#     # Split the data into records with known and unknown feature values
#     known_df = df.filter(col(feature).isNotNull())
#     unknown_df = df.filter(col(feature).isNull())

#     # Define the assembler and model
#     assembler = VectorAssembler(inputCols=input_features, outputCol="feature
```

```
#     rf = RandomForestRegressor(featuresCol="features", labelCol=feature)

#     # Pipeline: VectorAssembler -> RandomForest
#     pipeline = Pipeline(stages=[assembler, rf])

#     # Train the model on data where the feature is known
#     model = pipeline.fit(known_df)

#     # Predict the missing values
#     predictions = model.transform(unknown_df).select("SEQUENCE", "prediction"

#     # Join predictions back with the original dataset
#     # Join on 'SEQUENCE', and replace the original column values with the pre
#     df_imputed = known_df.unionByName(
#         unknown_df.join(predictions, "SEQUENCE")
#                   .drop(feature)
#                   .withColumnRenamed("prediction", feature),
#         allowMissingColumns=True
#     )

#     return df_imputed

# # List of features we want to exclude
# features_to_exclude = [
#     "CommercialData_EstHomeValue",
#     "CommercialData_EstimatedHHIncomeAmount",
#     "CommercialData_EstimatedAreaMedianHHIncome",
#     "CommercialData_AreaMedianEducationYears",
#     "CommercialData_AreaMedianHousingValue",
#     "CommercialData_AreaPcntHHMarriedCoupleNoChild",
#     "CommercialData_AreaPcntHHMarriedCoupleWithChild",
#     "CommercialData_AreaPcntHHSpanishSpeaking",
#     "CommercialData_AreaPcntHHWithChildren",
#     "US_Congressional_District",
#     "State_Senate_District",
#     "State_House_District"
# ]

# # Inspect column data types and exclude non-numeric columns along with the sp
# input_features = [column.name for column in df_adj_regex_2.schema.fields if
#                   isinstance(column.dataType, (IntegerType, FloatType, Double
#                   column.name not in features_to_exclude]

# # List of features we want to impute
# features_to_impute = [
#     "CommercialData_EstHomeValue",
#     "CommercialData_EstimatedHHIncomeAmount",
#     "CommercialData_EstimatedAreaMedianHHIncome",
#     "CommercialData_AreaMedianEducationYears",
#     "CommercialData_AreaMedianHousingValue",
#     "CommercialData_AreaPcntHHMarriedCoupleNoChild",
#     "CommercialData_AreaPcntHHMarriedCoupleWithChild",
#     "CommercialData_AreaPcntHHSpanishSpeaking",
#     "CommercialData_AreaPcntHHWithChildren"
# ]

# df_imputed = df_adj_regex_2

# # For loop imputing
# for feature in features_to_impute:
```

```
#     # Update df_imputed by imputing one feature at a time
#     df_imputed = impute_with_random_forest(df_imputed, feature, input_feature

# # df_imputed now contains the DataFrame with imputed values for the specifie

# print("done")
```

We've packed the initial data cleaning steps in to a funciton for reusablility

In [ ]:
```
from pyspark.sql import DataFrame
from pyspark.sql.functions import col, regexp_replace, when
from pyspark.sql.types import FloatType, StringType

columns_to_drop = [
    "Voters_StateVoterID", "Voters_FirstName", "Voters_MiddleName", "Voters_La
    "Residence_Addresses_AddressLine", "Residence_Addresses_Zip", "Residence_A
    "Residence_Addresses_HouseNumber", "Residence_Addresses_StreetName", "Resi
    "Residence_Addresses_CassErrStatCode", "Voters_SequenceZigZag", "Voters_Se
    "Residence_Addresses_CensusTract", "Residence_Addresses_CensusBlockGroup",
    "Residence_Addresses_Latitude", "Residence_Addresses_Longitude", "Residence
    "Residence_Addresses_Density", "Mailing_Addresses_AddressLine", "Mailing_A
    "Mailing_Addresses_State:", "Mailing_Addresses_Zip", "Mailing_Addresses_Zi
    "Mailing_Addresses_StreetName", "Mailing_Addresses_CassErrStatCode", "Mail
    "Mailing_Families_HHCount", "Mailing_HHGender_Description", "Mailing_HHPar
    "Voters_CalculatedRegDate", "Voters_OfficialRegDate", "AddressDistricts_Ch
    "AddressDistricts_Change_Changed_SD", "2001_State_Senate_District", "2001_
    "Voters_FIPS", "AddressDistricts_Change_Changed_County", "County_Commissio
    "Designated_Market_Area_DMA", "Unified_School_District", "CommercialData_I
    "CommercialData_MosaicZ4Global", "CommercialData_StateIncomeDecile", "Elec
    "ElectionReturns_G08_Cnty_Percent_McCain_R", "ElectionReturns_G08_Cnty_Vote
    "ElectionReturns_G08_Cnty_Margin_Obama_D", "ElectionReturns_G08_Cnty_Perce
    "ElectionReturns_G08_Cnty_Vote_Obama_D", "ElectionReturns_G12_Cnty_Margin_
    "ElectionReturns_G12_Cnty_Percent_Obama_D", "ElectionReturns_G12_Cnty_Vote_
    "ElectionReturns_G12_Cnty_Margin_Romney_R", "ElectionReturns_G12_Cnty_Perc
    "ElectionReturns_G12_Cnty_Vote_Romney_R", "ElectionReturns_G16_Cnty_Margin_
    "ElectionReturns_G16_Cnty_Percent_Clinton_D", "ElectionReturns_G16_Cnty_Vo
    "ElectionReturns_G16_Cnty_Margin_Trump_R", "ElectionReturns_G16_Cnty_Perce
    "ElectionReturns_G16_Cnty_Vote_Trump_R", "ElectionReturns_P08_Cnty_Pct_Cli
    "ElectionReturns_P08_Cnty_Pct_Obama_D", "ElectionReturns_P08_Cnty_Vote_Bid
    "ElectionReturns_P08_Cnty_Vote_Clinton_D", "ElectionReturns_P08_Cnty_Vote_
    "ElectionReturns_P08_Cnty_Vote_Edwards_D", "ElectionReturns_P08_Cnty_Vote_
    "ElectionReturns_P08_Cnty_Vote_Kucinich_D", "ElectionReturns_P08_Cnty_Vote_
    "ElectionReturns_P08_Cnty_Vote_Richardson_D", "ElectionReturns_P12_Cnty_Pc
    "ElectionReturns_P12_Cnty_Pct_Paul_R", "ElectionReturns_P12_Cnty_Pct_Romne
    "ElectionReturns_P12_Cnty_Pct_Santorum_R", "ElectionReturns_P12_Cnty_Vote_
    "ElectionReturns_P12_Cnty_Vote_Gingrich_R", "ElectionReturns_P12_Cnty_Vote_
    "ElectionReturns_P12_Cnty_Vote_Paul_R", "ElectionReturns_P12_Cnty_Vote_Per
    "ElectionReturns_P12_Cnty_Vote_Romney_R", "ElectionReturns_P12_Cnty_Vote_S
    "ElectionReturns_P16_Cnty_Pct_Bush_R", "ElectionReturns_P16_Cnty_Pct_Carso
    "ElectionReturns_P16_Cnty_Pct_Christie_R", "ElectionReturns_P16_Cnty_Pct_C
    "ElectionReturns_P16_Cnty_Pct_Fiorina_R", "ElectionReturns_P16_Cnty_Pct_Ka
    "ElectionReturns_P16_Cnty_Pct_Rubio_R", "ElectionReturns_P16_Cnty_Pct_Trum
    "ElectionReturns_P16_Cnty_Vote_Bush_R", "ElectionReturns_P16_Cnty_Vote_Car
    "ElectionReturns_P16_Cnty_Vote_Christie_R", "ElectionReturns_P16_Cnty_Vote_
    "ElectionReturns_P16_Cnty_Vote_Fiorina_R", "ElectionReturns_P16_Cnty_Vote_
    "ElectionReturns_P16_Cnty_Vote_Rubio_R", "ElectionReturns_P16_Cnty_Vote_Tr
    "ElectionReturns_P16_Cnty_Pct_Clinton_D", "ElectionReturns_P16_Cnty_Pct_Sa
    "ElectionReturns_P16_Cnty_Vote_Clinton_D", "ElectionReturns_P16_Cnty_Vote_
```

```
    "General_2020", "CommercialData_EstimatedHHIncome"
]

rows_to_drop = ['Precinct', 'Ethnic_Description', 'Voters_Gender', 'Judicial_Di

def clean_df(df: DataFrame,
                 missing_percentage_threshold: float = 15.0,
                 unnecessary_columns: list = columns_to_drop,
                 unnecessary_rows: list = rows_to_drop,
                 columns_with_special_handling: list = ["Voters_VotingPerforman
    """
    Cleans a DataFrame for imputation

    Parameters:
    - df: The input DataFrame.
    - missing_percentage_threshold: The percentage threshold of missing data al
    - unnecessary_columns: A list of column names that should be dropped from t
    - unnecessary_rows: A list of column names for which any row with missing d
    - columns_with_special_handling: Columns that require specific handling due

    Returns:
    - A cleaned and type-converted DataFrame.
    """
    print(f"Initial state -> Rows: {df.count()}, Columns: {len(df.columns)}")

    # Identify and drop columns based on missing data percentage and unnecessar
    print("Identifying columns with excessive missing data...")
    columns_with_missing_data = [column_name for column_name in df.columns if
    print("Dropping unnecessary columns...")
    df = df.drop(*columns_with_missing_data, *unnecessary_columns)

    # Dropping unnecessary rows with missing data
    print("Dropping unnecessary rows...")
    for column_name in unnecessary_rows:
        df = df.filter(col(column_name).isNotNull())

    # Convert types where applicable
    print("Converting data types...")
    for column_name in df.columns:
        if column_name not in columns_with_special_handling:
            df = df.withColumn(column_name, regexp_replace(col(column_name), "
        else:
            # Handle columns with special requirements
            df = df.withColumn(column_name, when(col(column_name) == "Not elig
                                   .otherwise(regexp_replace(col(column_name), "[^(
                                   .cast(FloatType()))

    print("Done!")

    print(f"Final state -> Rows: {df.count()}, Columns: {len(df.columns)}")
    return df
```

In [ ]:
```
# wyoming_cleaned = clean_df(wyoming_raw)
# Save the dataset to a parquet file for reproduction
# wyoming_cleaned.write.parquet("./wyoming_cleaned.parquet")
```

3. Household income levels

The first question we are interested in is whether there is a disparity in voter turnout among households based on income levels. In addressing this question, we selected specific features from the full dataset to eliminate irrelevant information and ensure a focused investigation. 'CommercialData_EstimatedHHIncomeAmount' provides an estimate of household income, which is a central variable in our analysis. To assess voting participation, we consider four key indicators of voter engagement: 'Voters_VotingPerformanceEvenYearGeneral', 'Voters_VotingPerformanceEvenYearPrimary', and 'Voters_VotingPerformanceMinorElection'. These variables offer a comprehensive view of voting behaviors in general elections, primary elections,in even years, as well as minor elections.

Dataset Loading

We load the cleaned_dataset from the parquet file.

In [ ]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Load Parquet File") \
    .getOrCreate()

# Load the cleaned dataset
wyoming_cleaned = spark.read.parquet('./wyoming_cleaned.parquet')
```

In [ ]:
```
income = wyoming_cleaned.select(
                                "CommercialData_EstimatedHHIncomeAmount",
                                "Voters_VotingPerformanceEvenYearGeneral",
                                "Voters_VotingPerformanceEvenYearPrimary",
                                "Voters_VotingPerformanceMinorElection"
)
```

3.1 Exploratory Data Analysis

3.1.1 Data cleaning

In [ ]:
```
from pyspark.sql.functions import col, count, when

print(f"Total rows: {income.count()}")

# Count the number of missing values in each row
income_missing_per_row = income.select([count(when(col(c).isNull(), c)).alias(

# To count the number of rows that have at least one missing value
income_rows_with_missing = income.filter(sum([col(c).isNull().cast("int") for 

print(f"Total rows with missing values: {income_rows_with_missing}")
```

```
Total rows: 257847
[Stage 5:=============================>                                (1 + 1) / 2]
Total rows with missing values: 53988
```

We drop the rows with missing values since we will still have 80% of the data for analysis.

In [ ]:
```python
income = income.dropna()
print(f"Total rows: {income.count()}")

# Count the number of missing values in each row
income_missing_per_row = income.select([count(when(col(c).isNull(), c)).alias(

# To count the number of rows that have at least one missing value
income_rows_with_missing = income.filter(sum([col(c).isNull().cast("int") for 

print(f"Total rows with missing values: {income_rows_with_missing}")
```

```
Total rows: 203859
Total rows with missing values: 0
```

3.1.2 Descriptive statistics

In [ ]:
```python
income.describe().show()
```

```
+-------+---------------------------------------+-------------------------------------------------+----------------------------------------+----------------------------------------+
|summary|CommercialData_EstimatedHHIncomeAmount|Voters_VotingPerformanceEvenYearGeneral|Voters_VotingPerformanceEvenYearPrimary|Voters_VotingPerformanceMinorElection|
+-------+---------------------------------------+-------------------------------------------------+----------------------------------------+----------------------------------------+
|  count|                                 203859|                                           203859|                                 203859|                                  203859|
|   mean|                      93569.84751225112|                              84.17885401184152|                       47.09607620953698|                                     0.0|
| stddev|                      52903.67266866898|                              21.695042764952422|                      36.91926810066215|                                     0.0|
|    min|                                 6000.0|                                              0.0|                                    0.0|                                     0.0|
|    max|                               250000.0|                                            100.0|                                  100.0|                                     0.0|
+-------+---------------------------------------+-------------------------------------------------+----------------------------------------+----------------------------------------+
```

The descriptive analysis of the dataset reveals some insights. With a wide range of household incomes from $\$6,000$ to $\$250,000$ and a mean of approximately $\$93,570$, the dataset encompasses a broad spectrum of economic statuses. The high average voter turnout in even year general elections (84.18%) underscores a strong engagement. The variation in turnout rates, as indicated by the standard deviation, suggests that the disparities in participation may correlate with income levels. The participation in minor elections are all 0.0% across the board, making the feature unnecessary for our analysis. These preliminary findings set the stage for a more detailed exploration.

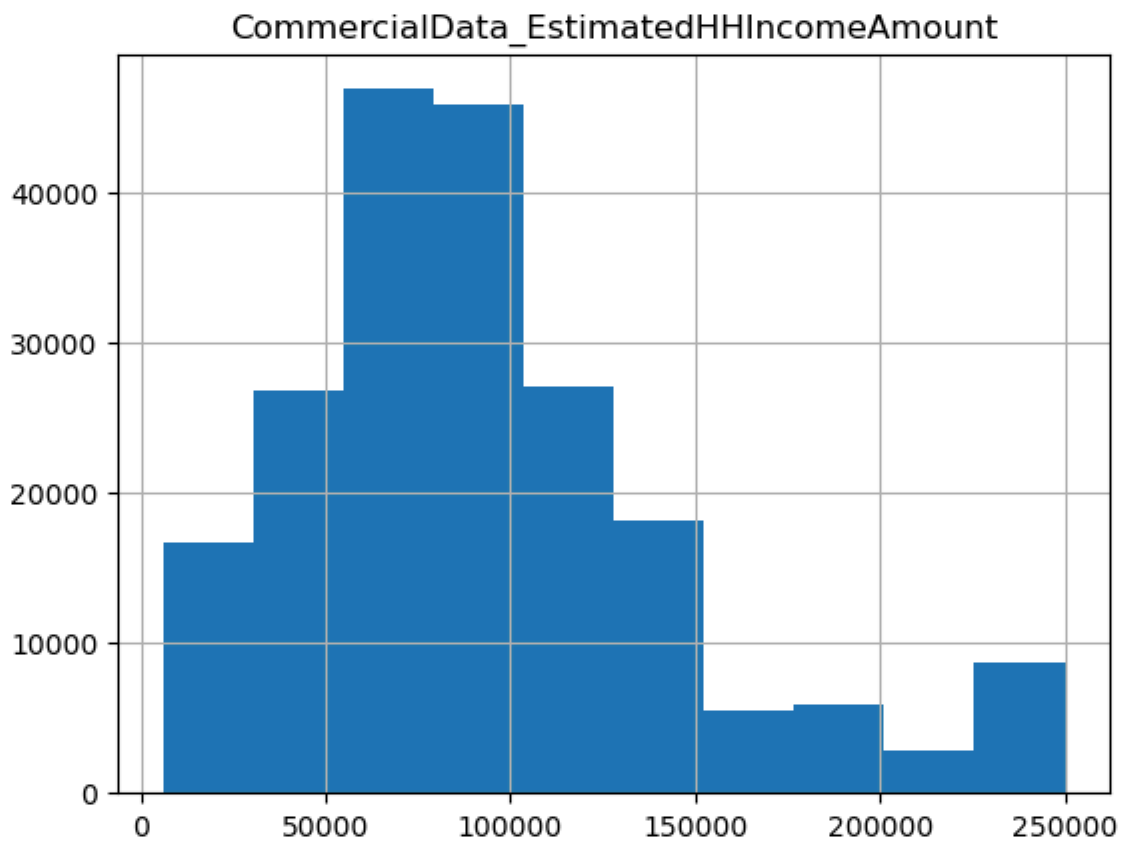
In [ ]:
```
# dump the minor election feature
income = income.drop('Voters_VotingPerformanceMinorElection')
```

3.1.3 Visualizations

Let's first see the distribution of CommercialData_EstimatedHHIncomeAmount to understand income distribution among the households.

In [ ]:
```
income.select("CommercialData_EstimatedHHIncomeAmount").toPandas().hist()
```

Out[ ]:
```
array([[<AxesSubplot:title={'center':'CommercialData_EstimatedHHIncomeAmount'}
>]],
      dtype=object)
```



The distribution of estimated household income appears to be roughly bell-shaped with a slight right skew, suggesting a normal distribution with most households earning between $\$50,000$ and $\$150,000$ annually with a presence of wealthier households,.

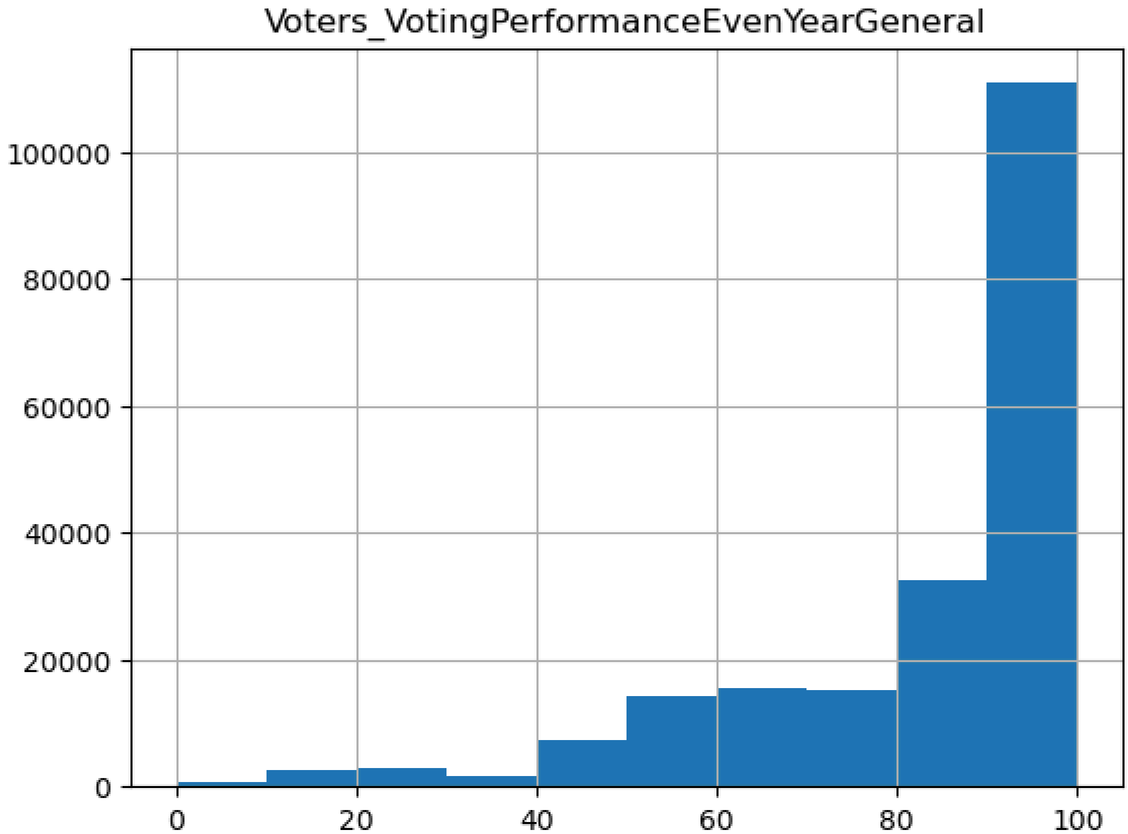
Similarly, we explore the distribution of voter turnout rates.

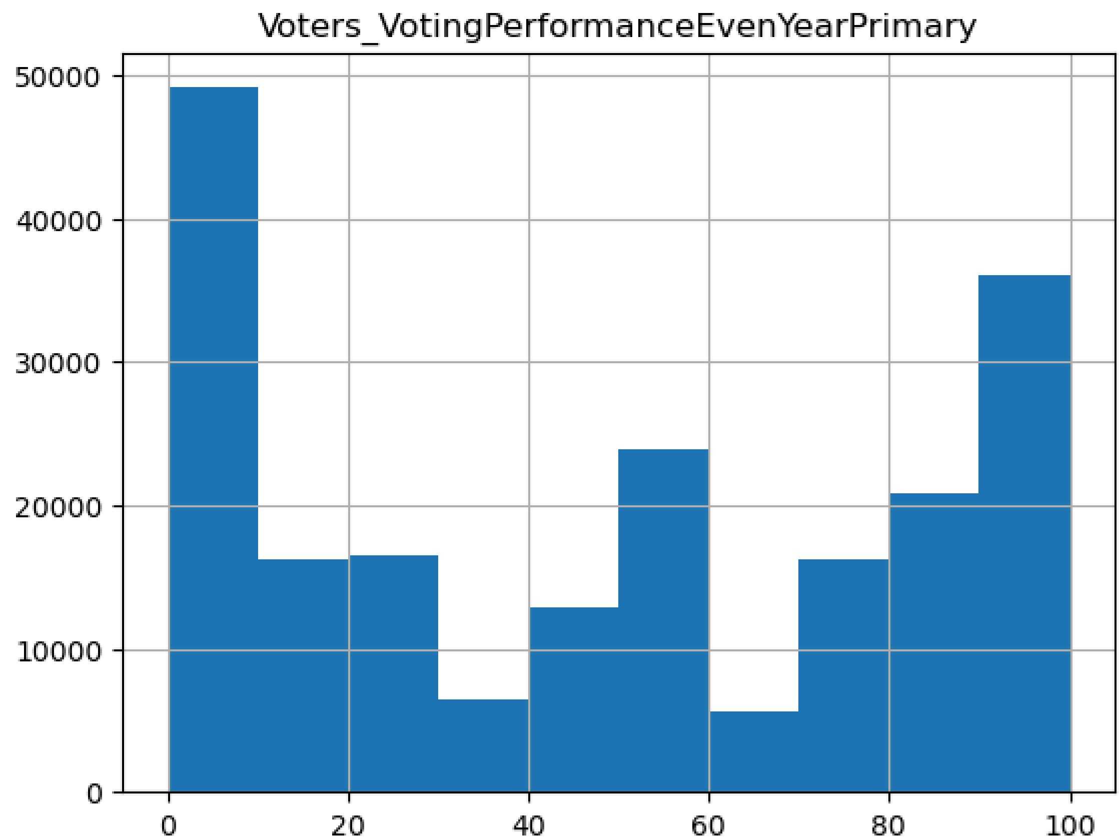
In [ ]:

```
income.select("Voters_VotingPerformanceEvenYearGeneral").toPandas().hist()
income.select("Voters_VotingPerformanceEvenYearPrimary").toPandas().hist()
```

Out[ ]:

```
array([[<AxesSubplot:title={'center':'Voters_VotingPerformanceEvenYearPrimar
y'}>]],
      dtype=object)
```





A very high number of households show close to 100% voting performance in even-year general elections, which might indicate a strong inclination to participate in these elections. However, the distribution for even-year primary elections shows a more varied pattern with several peaks, reflecting fluctuating levels of voter engagement.

3.1.4 Correlation Analysis

We now conduct the correlation analysis.

In [ ]:
```
for column in ["Voters_VotingPerformanceEvenYearGeneral", "Voters_VotingPerforn
    print(f"Correlation with {column}: ", income.stat.corr("CommercialData_Est:
```

```
Correlation with Voters_VotingPerformanceEvenYearGeneral:  0.02043330600753072
4
Correlation with Voters_VotingPerformanceEvenYearPrimary:  -0.0245627988989624
85
```

The correlation results indicate that there is a weak positive relationship between household income and voter turnout in even-year general elections, with a correlation coefficient of approximately 0.0204. This suggests that as household income increases, there is a slightly higher likelihood of participation in these elections.

Conversely, the weak negative correlation of approximately -0.0246 between household income and voter turnout in even-year primary elections suggests that there is a slightly lower likelihood of participation in these elections as income rises.

However, both relationships are weak and may suggest that household income is not a strong predictor of voter turnout in primary elections.

3.1.5 Modeling

Next, we aim to explore the relationships through statistical modleing. We employ Linear Regression model to quantifies the strength of the association and predict voter turnout based on income levels.

In [ ]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Assembler for input feature
income_assembler = VectorAssembler(
    inputCols=["CommercialData_EstimatedHHIncomeAmount"],
    outputCol="income_features"
)

# Model 1: Predicting Voters_VotingPerformanceEvenYearGeneral
income_lr_general = LinearRegression(featuresCol="income_features", labelCol="V
income_pipeline_general = Pipeline(stages=[income_assembler, income_lr_general]

# Split the dataset into training and testing sets
income_data_split = income.randomSplit([0.8, 0.2], seed=42)
income_training_data = income_data_split[0]
income_testing_data = income_data_split[1]

# Train Model 1 on even year general election data
income_model_general = income_pipeline_general.fit(income_training_data)

# Predict and evaluate Model 1
income_predictions_general = income_model_general.transform(income_testing_data
income_evaluator_general = RegressionEvaluator(labelCol="Voters_VotingPerforman
income_rmse_general = income_evaluator_general.evaluate(income_predictions_gene
print("RMSE (General Elections):", income_rmse_general)

# Model 2: Predicting Voters_VotingPerformanceEvenYearPrimary
income_lr_primary = LinearRegression(featuresCol="income_features", labelCol="V
income_pipeline_primary = Pipeline(stages=[income_assembler, income_lr_primary]

# Train Model 2 on even year primary election data
income_model_primary = income_pipeline_primary.fit(income_training_data)

# Predict and evaluate Model 2
income_predictions_primary = income_model_primary.transform(income_testing_data
income_evaluator_primary = RegressionEvaluator(labelCol="Voters_VotingPerforman
income_rmse_primary = income_evaluator_primary.evaluate(income_predictions_prim
print("RMSE (Primary Elections):", income_rmse_primary)
```

```
24/03/20 17:03:37 WARN Instrumentation: [65e7425e] regParam is zero, which mig
ht cause numerical instability and overfitting.
24/03/20 17:03:41 WARN Instrumentation: [7cec0566] regParam is zero, which mig
ht cause numerical instability and overfitting.
RMSE (General Elections): 21.776959108558728
```

```
RMSE (Primary Elections): 36.9983864857926
```

We use RMSE (Root Mean Squared Error) as the metric to evaluate the predictive performance of our linear regression models. RMSE measures the average magnitude of the errors, or the differences between the predicted voter turnout and the actual voter turnout percentages. In our modeling, the RMSE values obtained from the models predicting even year general elections and primary elections are 21.78 and 37.00, respectively.

For the even-year general election model, an RMSE of 21.777 suggests that, on average, the model's predictions deviate from the actual voter turnout by approximately 21.8 percentage points. Considering the high average voter turnout in general elections (around 84%), this RMSE value can be considered relatively high, indicating that the model's predictions are not highly accurate in predicting general election turnout based solely on household income.

In the case of the even-year primary election model, an RMSE of 36.998 represents a significant deviation from the actual turnout, suggesting that the model's predictions for primary election turnout have a lower level of accuracy compared to the general election model.

The high RMSE values for both models indicate that household income alone is not a strong predictor of voter turnout, and there are likely other factors that play a more significant role in determining voter participation rates.

4. Household Count

The next question we are interested in is whether there is a disparity in voter turnout based off of household count. To answer this question we will be looking at the variables "Residence_Families_HHCount", "Voters_VotingPerformanceEvenYearGeneral", and "Voters_VotingPerformanceEvenYearPrimary". From the previous analysis, we have shown that the variable "Voters_VotingPerformanceMinorElection" provides no additional information. "Residence_Families_HHCount" gives us the number of people in a household and the other two variables display voting data in general and primary elections for even years.

4.1 Exploratory Data Analysis

4.1.1 Data cleaning

In [ ]:
```
HHcount = wyoming_cleaned.select(
                                    "Residence_Families_HHCount",
                                    "Voters_VotingPerformanceEvenYearGeneral",
```

```
                                  "Voters_VotingPerformanceEvenYearPrimary"
)
```

In [ ]:
```
from pyspark.sql.functions import col, count, when

print(f"Total rows: {HHcount.count()}")

# Count the number of missing values in each row
count_missing_per_row = HHcount.select([count(when(col(c).isNull(), c)).alias(

# To count the number of rows that have at least one missing value
count_rows_with_missing = HHcount.filter(sum([col(c).isNull().cast("int") for

print(f"Total rows with missing values: {count_rows_with_missing}")
```

```
Total rows: 257847
Total rows with missing values: 31552
```

After checking for missing values we drop those rows and retain approximately 88% of the data

In [ ]:
```
HHcount = HHcount.dropna()
```

In [ ]:
```
print(f"Total rows: {HHcount.count()}")

# Count the number of missing values in each row
count_missing_per_row = HHcount.select([count(when(col(c).isNull(), c)).alias(

# To count the number of rows that have at least one missing value
count_rows_with_missing = HHcount.filter(sum([col(c).isNull().cast("int") for

print(f"Total rows with missing values: {count_rows_with_missing}")
```

```
Total rows: 226295
Total rows with missing values: 0
```

4.1.2 Descriptive statistics

In [ ]:
```
HHcount.describe().show()
```

```
+-------+-------------------------+--------------------------------------+--
------------------------------------+
|summary|Residence_Families_HHCount|Voters_VotingPerformanceEvenYearGeneral|Vo
ters_VotingPerformanceEvenYearPrimary|
+-------+-------------------------+--------------------------------------+--
------------------------------------+
|  count|                   226295|                                226295|
226295|
|   mean|        1.914293289732429|                     84.33562385381913|
47.87237455533706|
| stddev|       0.7530058207391012|                    22.172494643497746|
37.666276965153074|
|    min|                      1.0|                                   0.0|
0.0|
|    max|                      8.0|                                 100.0|
100.0|
+-------+-------------------------+--------------------------------------+--
------------------------------------+
```

Looking at the descriptive statistics, we see that the average household count ranges from 1 to 8 with the average being almost 2. Since the standard deviation is also less than one, we expect most values to be 1, 2, or 3. This makes sense since 8 person households are pretty rare. Moreover, as seen previously, the voter turnout for even year general is high while the turnout for even year primary is lower.

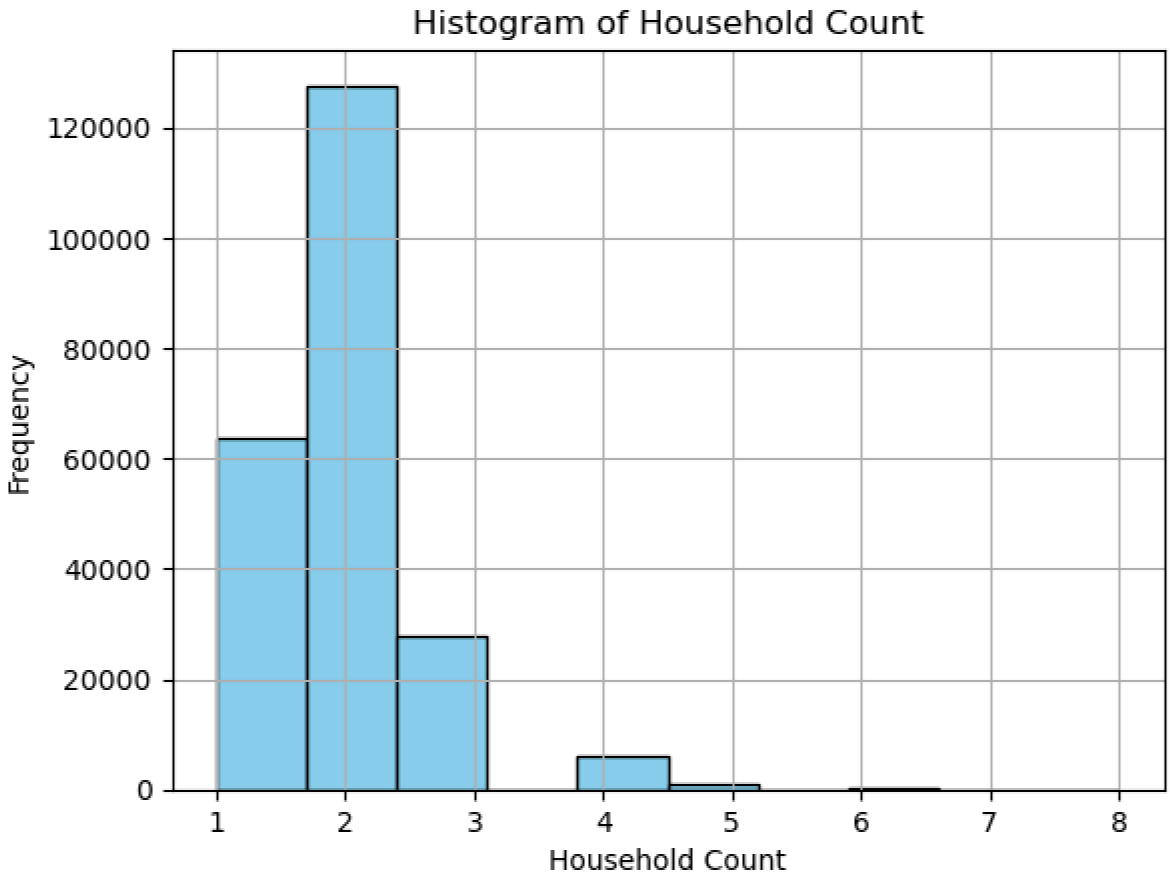
4.1.3 Visualizations

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
HHcount = HHcount.toPandas()
```

In [ ]:

```
HHcount['Residence_Families_HHCount'].plot(kind='hist', bins=10, color='skyblue
plt.title('Histogram of Household Count')
plt.xlabel('Household Count')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Looking at the histogram we see that the majority of values are indeed 1, 2, and 3 as predicted. This leads us to wonder what percentage of values are greater than 3. There seem to be some around 4 but past that the values are sparse.

In [ ]:
```
count_over_3 = (HHcount['Residence_Families_HHCount'] > 3).sum()
total_percent = round(count_over_3/226295, 3)
print("There are " + str(count_over_3) + " values over 3 which is " + str(tota
```

```
There are 7359 values over 3 which is 0.033 of the total.
```

From this we see that roughly only 3% of all the values are above 3. This leads us to believe that household count might not be such a good predictor of voter turnout since there isn't much distinction between the values for household count. There most likely isn't much of a difference from a one person household to a two person household.

4.1.4 Correlation Analysis

In [ ]:
```
correlation_general = HHcount["Residence_Families_HHCount"].corr(HHcount['Vote
print('The correlation coefficient between Residence_Families_HHCount and Vote
```

```
The correlation coefficient between Residence_Families_HHCount and Voters_Voti
ngPerformanceEvenYearGeneral is 0.0661870351181536
```

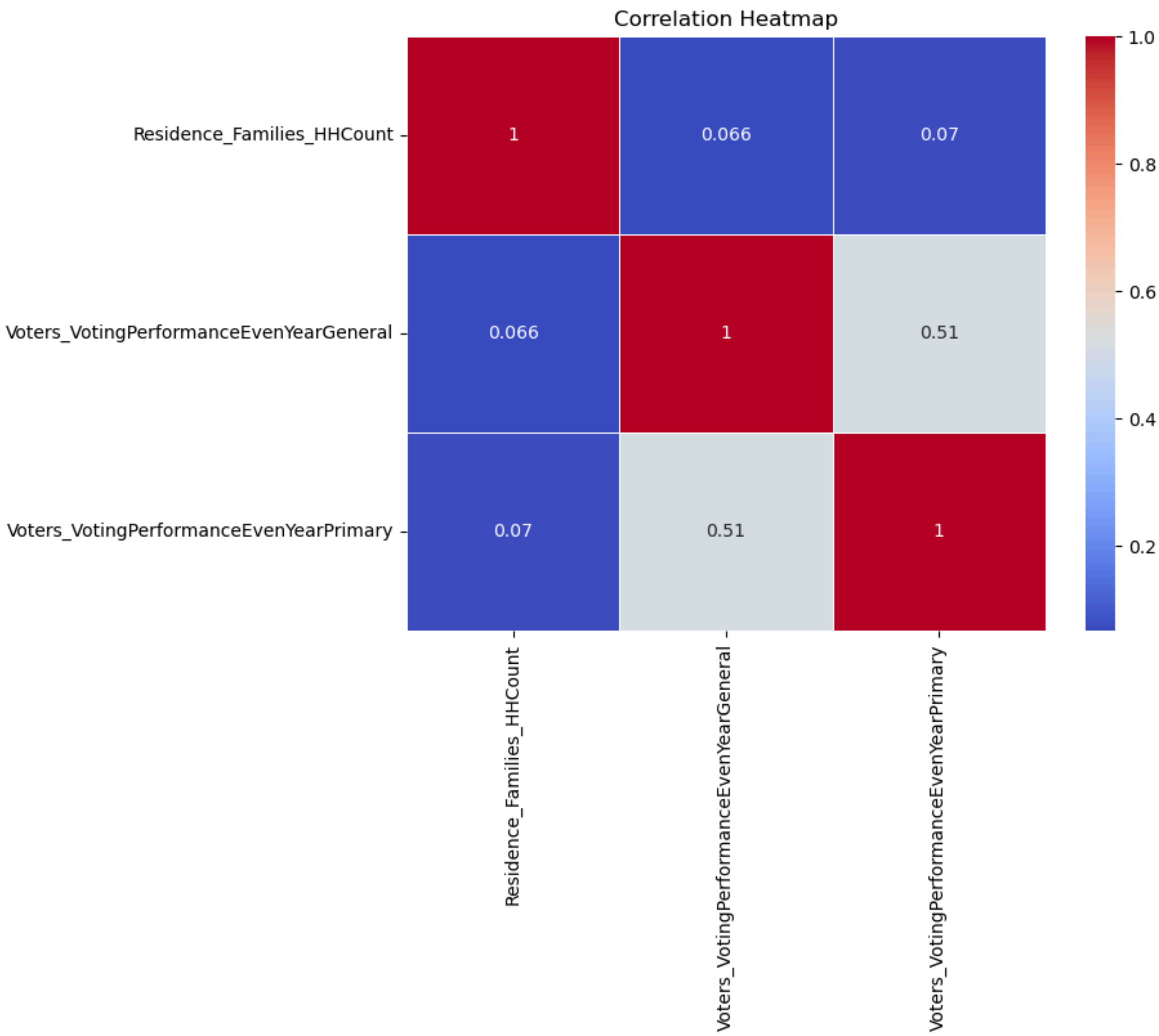
In [ ]:
```
correlation_primary = HHcount["Residence_Families_HHCount"].corr(HHcount['Vote
print('The correlation coefficient between Residence_Families_HHCount and Vote
```

```
The correlation coefficient between Residence_Families_HHCount and Voters_Voti
ngPerformanceEvenYearPrimary is 0.07027600654147977
```

In [ ]:

```python
correlation_matrix = HHcount.corr()

# Creating heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



From the heatmap and the correlation coefficients, we can see that the household count is not closely related with voter turnout since a score is 0.07 essentially means no relation. General and primary voting are more highly correlated with a score of .51. This reaffirms our prediction that household count isn't a good predictor of voter turnout.

5 Housing Cost

So far we have analyzed the impact of Household Income Levels and Household Size on our voter turnout. Our third question of interest to us is whether the value, or price of a house has any relationship with voting. We still plan to include the Voters_VotingPerformanceEvenYearGeneral and Voters_VotingPerformanceEvenYearPrimary as they are relevant variables to our question. We also plan to include the CommercialData_AreaMedianHousingValue variable which will help us determine if the price of a house is a good predictor of voter turnout.

```
In [ ]: housing_cost = wyoming_cleaned.select("CommercialData_AreaMedianHousingValue",
                                  "Voters_VotingPerformanceEvenYearGeneral",
                                  "Voters_VotingPerformanceEvenYearPrimary")
```

```
In [ ]: housing_cost
```

```
+-------------------------------------+---------------------------------------
+-------------------------------------+
|CommercialData_AreaMedianHousingValue|Voters_VotingPerformanceEvenYearGeneral
|Voters_VotingPerformanceEvenYearPrimary|
+-------------------------------------+---------------------------------------
+-------------------------------------+
|                             172844.0|                                   50.0
|                                  0.0|
|                             172844.0|                                  100.0
|                                100.0|
|                             189582.0|                                   71.0
|                                 16.0|
|                             253332.0|                                   66.0
|                                  0.0|
|                             253332.0|                                   60.0
|                                  0.0|
|                             253332.0|                                  100.0
|                                  0.0|
|                             253332.0|                                   80.0
|                                  0.0|
|                             253332.0|                                  100.0
|                                 25.0|
|                             253332.0|                                  100.0
|                                 50.0|
|                             253332.0|                                   60.0
|                                  0.0|
|                             253332.0|                                   40.0
|                                  0.0|
|                             253332.0|                                   66.0
|                                  0.0|
|                             253332.0|                                  100.0
|                                 50.0|
|                             253332.0|                                   80.0
|                                 25.0|
|                             253332.0|                                  100.0
|                                100.0|
|                             253332.0|                                   50.0
|                                 33.0|
|                             253332.0|                                   66.0
|                                 50.0|
|                             253332.0|                                   42.0
|                                 16.0|
|                             253332.0|                                   60.0
|                                 40.0|
|                             253332.0|                                   66.0
|                                  0.0|
+-------------------------------------+---------------------------------------
+-------------------------------------+
only showing top 20 rows
```

Let us confirm that the number of observations or rows we have in our dataset is 257847, but we still need to check for missing values so let's do this below.

In [ ]:
```
from pyspark.sql.functions import col, count, when

print(f"Total rows: {housing_cost.count()}")

# Count the number of missing values in each row
count_missing_per_row = housing_cost.select([count(when(col(c).isNull(), c)).a

# To count the number of rows that have at least one missing value
count_rows_with_missing = housing_cost.filter(sum([col(c).isNull().cast("int")

print(f"Total rows with missing values: {count_rows_with_missing}")
```

```
Total rows: 257847
Total rows with missing values: 65096
```

We have roughly 25% of missing data based on what we see here. A lot of data is being dealt with so removing rows with missing values is not going to prevent us from observing any relationships between these features. Let's do that below:

In [ ]:
```
housing_cost = housing_cost.dropna()
housing_cost.count()
```

Out[ ]:
```
192751
```

In [ ]:
```
housing_cost.describe().show()
```

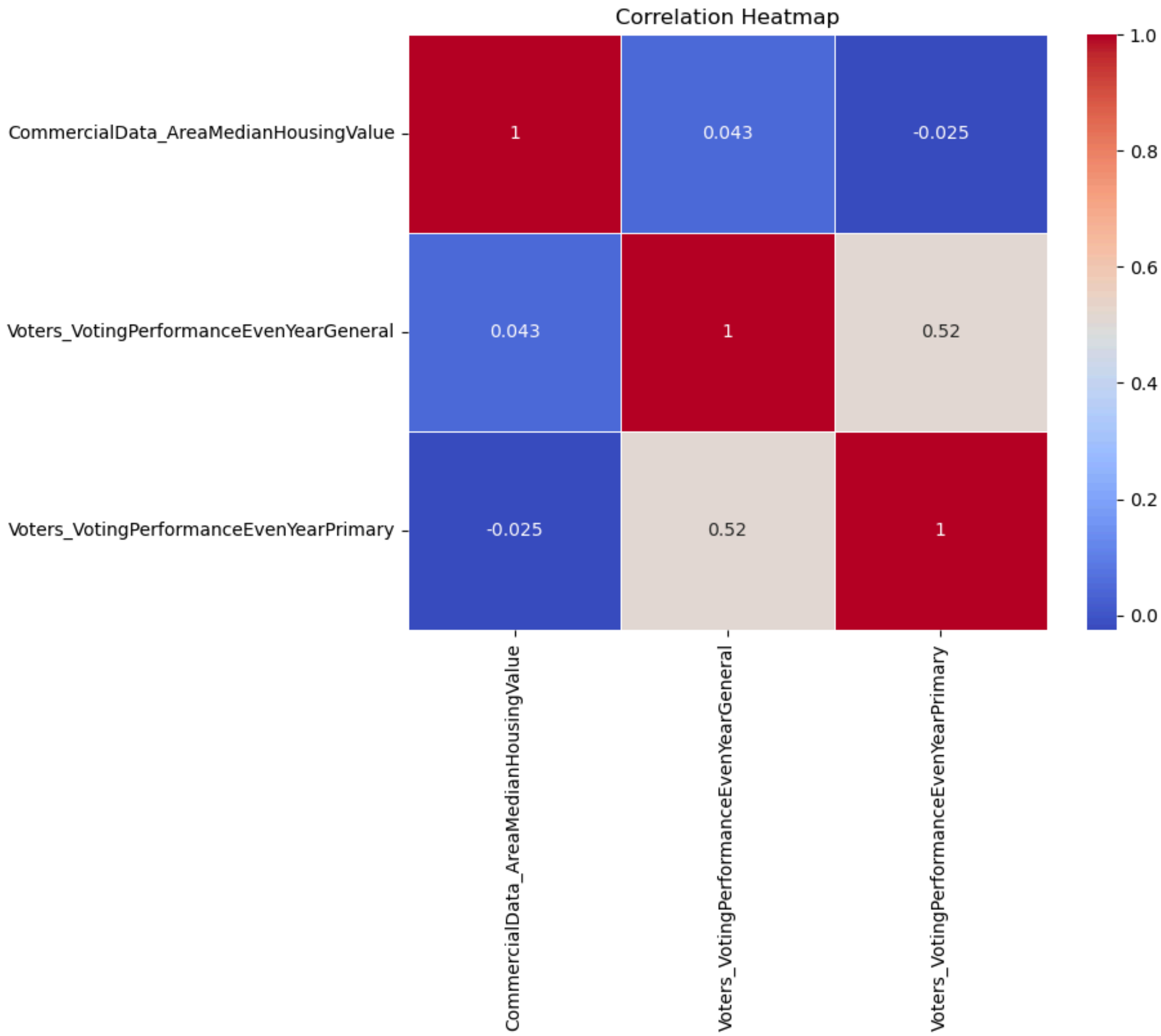
```
+-------+-------------------------------------+-------------------------------
-------+-------------------------------------+
|summary|CommercialData_AreaMedianHousingValue|Voters_VotingPerformanceEvenYea
rGeneral|Voters_VotingPerformanceEvenYearPrimary|
+-------+-------------------------------------+-------------------------------
-------+-------------------------------------+
|  count|                               192751|
192751|                               192751|
|   mean|                   241716.21803259128|                           84.448454
22332439|                    48.21414156087388|
| stddev|                   127146.45525727492|                            22.03946
53232576|                    37.69336444151071|
|    min|                                  0.0|
0.0|                                  0.0|
|    max|                            1202380.0|
100.0|                                100.0|
+-------+-------------------------------------+-------------------------------
-------+-------------------------------------+
```

After conducting summary statistics, we can see that there is a great deal of variation in median housing value. The average house is roughly $250K, with a deviation of about$130K. This however isn't too surprising as we are dealing with data all throughout Wyoming and families come from different socioeconomic backgrounds.Let's conduct a correlation plot of these three features to further explore relationships between the variables.

In [ ]:
```
hcp = housing_cost.toPandas()
correlation_mat = hcp.corr()

# Create the heatmap
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_mat, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



5.4 Data Visualization

It is evident from this plot that there is no real correlation between the median housing value and voter turnout, whether that be for primaries or the general election. There is a positive correlation between voting in the primaries and the general election, which we have observed above.Let us now confirm this belief through a scatterplot with Median Housing Value on the x-axis, and Average Voter Turnout on the y-axis colorcoded by primary and general elections.

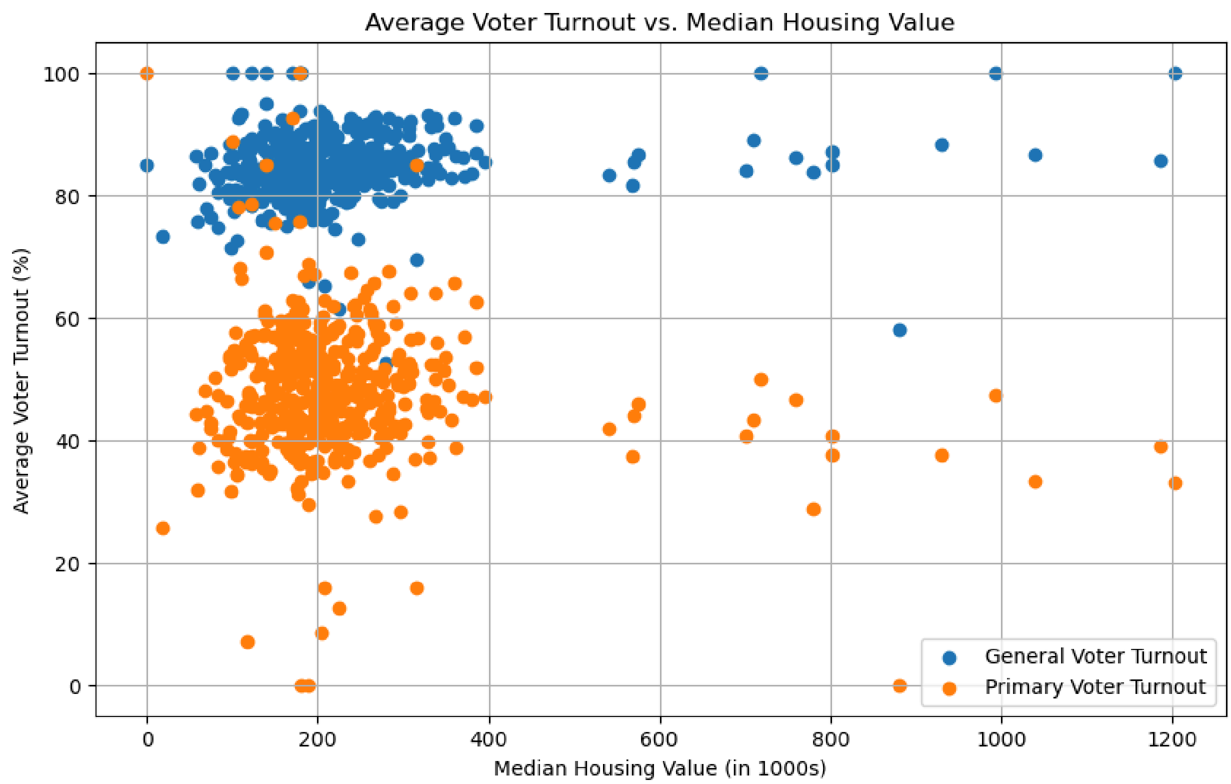
In [ ]:
```
# Grouping the data by median housing values and calculating the average voter
avg_turnout = housing_cost.groupby("CommercialData_AreaMedianHousingValue").mea

median_housing_value = avg_turnout.select(['CommercialData_AreaMedianHousingVal
median_housing_value_list = median_housing_value['CommercialData_AreaMedianHous
median_housing_value_list = [int(v)/1000 for v in median_housing_value_list]

avg_general_turnout = avg_turnout.select(['avg(Voters_VotingPerformanceEvenYea
avg_general_turnout_list = avg_general_turnout['avg(Voters_VotingPerformanceEve
```

```
avg_primary_turnout = avg_turnout.select(['avg(Voters_VotingPerformanceEvenYea
avg_primary_turnout_list = avg_primary_turnout['avg(Voters_VotingPerformanceEv
```

In [ ]:
```
import matplotlib.pyplot as plt
# Creating a line plot for both General and Primary Voter Turnout vs. Median Ho
plt.figure(figsize=(10, 6))
plt.scatter(median_housing_value_list, avg_general_turnout_list, label="Genera
plt.scatter(median_housing_value_list, avg_primary_turnout_list, label="Primary
plt.title("Average Voter Turnout vs. Median Housing Value")
plt.xlabel("Median Housing Value (in 1000s)")
plt.ylabel("Average Voter Turnout (%)")
plt.legend()
plt.grid(True)
plt.show()
```



From this scatterplot, we can see that there isn't any noticeable impact of mean housing value on the average voter turnout for general or primary. On a side note, if we eyeball the scatterplot for Median Housing Values below 400K for primary voter turnout, we roughly see that the average voter turnout is about 50% and when we look at housing values above 400K, we see that turnout is closer to around 40%. This could be because the families with a lower income background value the primary election more than the wealthier individuals. This is different for the general voter turnout where around 80% of the below 400K cohort voted, and around 90% of the above 400K cohort voted.

6 Conclusion

Based on our analysis, we couldn't make any conclusions for the impact of our three features: Income, Household size, and Median Housing Value on voting turnout. We chose these variables because we felt that these would have made a difference to the amount people voted but our data didn't support this. We conducted modeling for Income, but we were unable to make any conclusions even after doing so so we didn't conduct any modeling for the other two variables. It is entirely possible that there were features that would have been able to predict voter turnout and for that, we would have to spend more time understanding the variables. In addition, it could be that the state of Wyoming didn't have a correlation but another state would have had one. This isn't due to a shortage of data as we had an abundance of data to work with. But as we have learned, it can be very difficult at times to predict a variable even after using statistical modeling. Overall, we gained a lot from this project and it has furthered our interest in data science.

In [ ]:

```

```