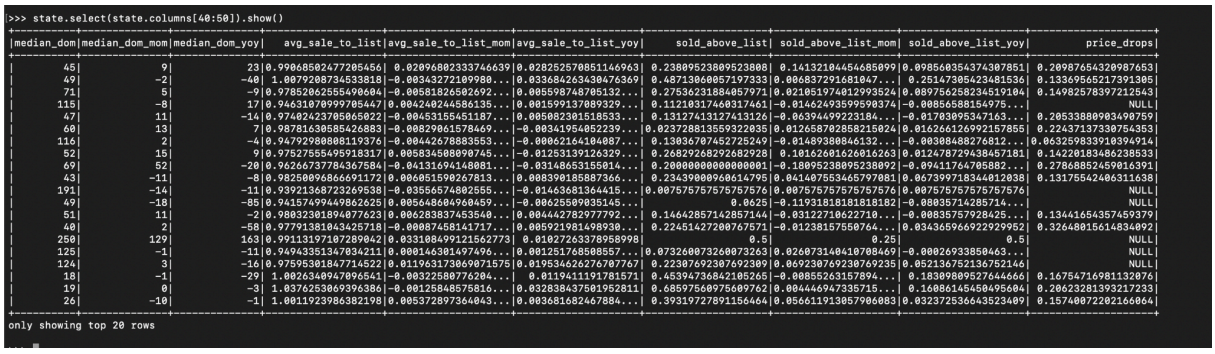
Part 1:

LINK TO DATASET:
https://www.kaggle.com/datasets/thuynyle/redfin-housing-market-data?select=us_national_market_tracker.tsv000

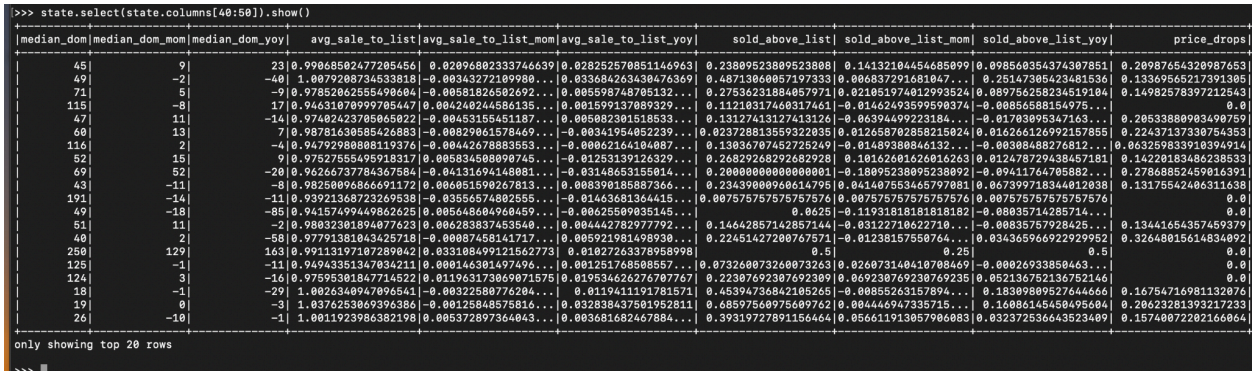
LINK TO SPREADSHEET:
https://docs.google.com/spreadsheets/d/1rwwvcn7_xmtw7XBHDbvBhglyAkcnsy-S0FTpkUQ_sok/edit#gid=0

Before Cleaning:



```
>>> state.select(state.columns[40:50]).show()
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|median_dom|median_dom_mom|median_dom_yoy|    avg_sale_to_list|avg_sale_to_list_mom|avg_sale_to_list_yoy|     sold_above_list| sold_above_list_mom| sold_above_list_yoy|         price_drops|
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|        45|            9|           23|0.990685024772054...|0.020968023337466...|0.028252570851146...| 0.23809523809523808| 0.14132104454685099|0.098560354374307...| 0.20987654320987653|
|        49|           -2|          -40| 1.007920873453381...|-0.00343272109980...|0.033684263430476...| 0.48713060057197333|0.006837291681047...| 0.25147305423481536| 0.13369565217391305|
|        71|            5|           -9|0.978520625554906...|-0.00581826502692...|0.005598748705132...| 0.27536231884057971|0.021051974012993524|0.089756258234519104| 0.14982578397212543|
|       115|           -8|           17|0.946310709997054...|0.004240244586135...|0.001599137089329...| 0.11210317460317461|-0.01462493599590374|-0.00856588154975...|                NULL|
|        47|           11|          -14|0.974024237050650...|-0.00453155451187...|0.005082301518533...| 0.13127413127413126|-0.06394499223184...|-0.01703095347163...| 0.20533880903490759|
|        60|           13|            7|0.978716305854268...|-0.00829861578469...|-0.00341954052239...|0.023728813559322035|0.012658702858215024|0.016266126992157855| 0.22437137330754353|
|       116|            2|           -4|0.947929880881193...|-0.00442678883553...|-0.00062164104087...| 0.13036707452725249|-0.01489380846132...|-0.00308480276812...|0.063259833910394914|
|        52|           15|            9|0.975275554959183...|0.005834508090745...|-0.01253139126329...| 0.26829268292682928| 0.18162601626016263|0.012478729438457181| 0.14228183486238533|
|        69|           52|          -20|0.962667377843675...|-0.04131694148081...|-0.03148653155014...| 0.20000000000000001|-0.18095238095238092|-0.09411764705882...| 0.27868852459016391|
|        43|          -11|           -8|0.982500968666911...|0.006051590267813...|0.008390185887366...| 0.23439000960614795|0.041407553465797081|0.067399718344012038| 0.13175542406311638|
|       191|          -14|          -11|0.939213687233695...|-0.03556574802555...|-0.01463681364415...|0.007575757575757576|0.007575757575757576|0.007575757575757576|                NULL|
|        49|          -18|          -85|0.941574999449862...|0.005648604960459...|-0.00625509035145...|              0.0625|-0.11931818181818182|-0.08035714285714...|                NULL|
|        51|           11|           -2|0.980323018940776...|0.006283837453540...|0.004442782977792...| 0.14642857142857144|-0.03122710622710...|-0.00835757928425...| 0.13441654357459379|
|        40|            2|          -58|0.977913810434257...|-0.00087458141717...|0.005921981498930...| 0.22451427200767571|-0.01238157550764...|0.034365966922929952| 0.32648015614834092|
|       250|          129|          163|0.991131971072899...|0.033108499121562773| 0.01027263378958998|                 0.5|                0.25|                 0.5|                NULL|
|       125|           -1|          -11|0.949433513470342...|0.000146301497496...|0.001251768508557...|0.07326007326007326...|0.026073140410708469|-0.00026933850463...|                NULL|
|       124|            3|          -16|0.975953018477145...|0.011963173069071575|0.019534626276707767| 0.22307692307692309|0.069230769230769235|0.052136752136752146|                NULL|
|        18|           -1|          -29| 1.002634094709654...|-0.00322580776204...|  0.0119411191781571| 0.45394736842105265|-0.00855263157894...| 0.18309809527644666| 0.16754716981132076|
|        19|            0|           -3| 1.037625306939638...|-0.00125848575816...|0.032838437501952811| 0.68597560975609762|0.004446947335715...| 0.16086145450495604| 0.20623281393217233|
|        26|          -10|           -1| 1.001192398638219...|0.005372897364043...|0.003681682467884...| 0.39319727891156464|0.056611913057906083|0.032372536643523409| 0.15740072202166064|
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 20 rows
```

After Cleaning:



```
>>> state.select(state.columns[40:50]).show()
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|median_dom|median_dom_mom|median_dom_yoy|    avg_sale_to_list|avg_sale_to_list_mom|avg_sale_to_list_yoy|     sold_above_list| sold_above_list_mom| sold_above_list_yoy|         price_drops|
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|        45|            9|           23|0.990685024772054...|0.020968023337466...|0.028252570851146963| 0.23809523809523808| 0.14132104454685099|0.098560354374307851| 0.20987654320987653|
|        49|           -2|          -40| 1.007920873453381...|-0.00343272109980...|0.033684263430476369| 0.48713060057197333|0.006837291681047...| 0.25147305423481536| 0.13369565217391305|
|        71|            5|           -9|0.978520625554906...|-0.00581826502692...|0.005598748705132...| 0.27536231884057971|0.021051974012993524|0.089756258234519104| 0.14982578397212543|
|       115|           -8|           17|0.946310709997054...|0.004240244586135...|0.001599137089329...| 0.11210317460317461|-0.01462493599590374|-0.00856588154975...|                 0.0|
|        47|           11|          -14|0.974024237050650...|-0.00453155451187...|0.005082301518533...| 0.13127413127413126|-0.06394499223184...|-0.01703095347163...| 0.20533880903490759|
|        60|           13|            7|0.978716305854268...|-0.00829861578469...|-0.00341954052239...|0.023728813559322035|0.012658702858215024|0.016266126992157855| 0.22437137330754353|
|       116|            2|           -4|0.947929880881193...|-0.00442678883553...|-0.00062164104087...| 0.13036707452725249|-0.01489380846132...|-0.00308480276812...|0.063259833910394914|
|        52|           15|            9|0.975275554959183...|0.005834508090745...|-0.01253139126329...| 0.26829268292682928| 0.18162601626016263|0.012478729438457181| 0.14228183486238533|
|        69|           52|          -20|0.962667377843675...|-0.04131694148081...|-0.03148653155014...| 0.20000000000000001|-0.18095238095238092|-0.09411764705882...| 0.27868852459016391|
|        43|          -11|           -8|0.982500968666911...|0.006051590267813...|0.008390185887366...| 0.23439000960614795|0.041407553465797081|0.067399718344012038| 0.13175542406311638|
|       191|          -14|          -11|0.939213687233695...|-0.03556574802555...|-0.01463681364415...|0.007575757575757576|0.007575757575757576|0.007575757575757576|                 0.0|
|        49|          -18|          -85|0.941574999449862...|0.005648604960459...|-0.00625509035145...|              0.0625|-0.11931818181818182|-0.08035714285714...|                 0.0|
|        51|           11|           -2|0.980323018940776...|0.006283837453540...|0.004442782977792...| 0.14642857142857144|-0.03122710622710...|-0.00835757928425...| 0.13441654357459379|
|        40|            2|          -58|0.977913810434257...|-0.00087458141717...|0.005921981498930...| 0.22451427200767571|-0.01238157550764...|0.034365966922929952| 0.32648015614834092|
|       250|          129|          163|0.991131971072899...|0.033108499121562773| 0.01027263378958998|                 0.5|                0.25|                 0.5|                 0.0|
|       125|           -1|          -11|0.949433513470342...|0.000146301497496...|0.001251768508557...|0.07326007326007326...|0.026073140410708469|-0.00026933850463...|                 0.0|
|       124|            3|          -16|0.975953018477145...|0.011963173069071575|0.019534626276707767| 0.22307692307692309|0.069230769230769235|0.052136752136752146|                 0.0|
|        18|           -1|          -29| 1.002634094709654...|-0.00322580776204...|  0.0119411191781571| 0.45394736842105265|-0.00855263157894...| 0.18309809527644666| 0.16754716981132076|
|        19|            0|           -3| 1.037625306939638...|-0.00125848575816...|0.032838437501952811| 0.68597560975609762|0.004446947335715...| 0.16086145450495604| 0.20623281393217233|
|        26|          -10|           -1| 1.001192398638219...|0.005372897364043...|0.003681682467884...| 0.39319727891156464|0.056611913057906083|0.032372536643523409| 0.15740072202166064|
+----------+-------------+-------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 20 rows
```

Code Snippets:

```
state = spark.read.csv("data/state_market_tracker.tsv000", sep='\t', header = True)

zip = spark.read.csv("data/zip_code_market_tracker.tsv000", sep='\t', header = True)

national = spark.read.csv("data/us_national_market_tracker.tsv000", sep='\t', header = True)

county = spark.read.csv("data/county_market_tracker.tsv000", sep='\t', header = True)

neighborhood = spark.read.csv("data/neighborhood_market_tracker.tsv000", sep='\t', header = True)

state = state.withColumn("period_begin", col("period_begin").cast(DateType()))
state = state.withColumn("period_end", col("period_end").cast(DateType()))
state = state.withColumn("median_sale_price", col("median_sale_price").cast(FloatType()))
state = state.withColumn("median_list_price", col("median_list_price").cast(FloatType())) (edited)
```

Our dataset includes different files (neighborhood, zip_code, state, national, and county). To clean the dataset, we created DataFrames for every file, and cast every value in the files to floats so that it is easier for us to analyze.

```
>>> state = state.fillna({
... 'period_duration': '0',  # Assuming it's a duration in string format like '30' for 30 days
...     'median_sale_price': 0.0,
...     'median_sale_price_mom': 0.0,
...     'median_sale_price_yoy': 0.0,
...     'median_list_price': 0.0,
...     'median_list_price_mom': 0.0,
...     'median_list_price_yoy': 0.0,
...     'median_ppsf': 0.0,
...     'median_ppsf_mom': 0.0,
...     'median_ppsf_yoy': 0.0,
...     'median_list_ppsf': 0.0,
...     'median_list_ppsf_mom': 0.0,
...     'median_list_ppsf_yoy': 0.0,
...     'homes_sold': 0.0,
...     'homes_sold_mom': 0.0,
...     'homes_sold_yoy': 0.0,
...     'pending_sales': 0.0,
...     'pending_sales_mom': 0.0,
...     'pending_sales_yoy': 0.0,
...     'new_listings': 0.0,
...     'new_listings_mom': 0.0,
...     'new_listings_yoy': 0.0,
...     'inventory': 0.0,
...     'inventory_mom': 0.0,
...     'inventory_yoy': 0.0,
...     'months_of_supply': 0.0,
...     'months_of_supply_mom': 0.0,
...     'months_of_supply_yoy': 0.0,
...     'median_dom': 0.0,  # median days on market
...     'median_dom_mom': 0.0,
...     'median_dom_yoy': 0.0,
...     'avg_sale_to_list': 0.0,
...     'avg_sale_to_list_mom': 0.0,
```

To clean our dataset, we filled all NULL values with zero using the code above.

Describe the Data:

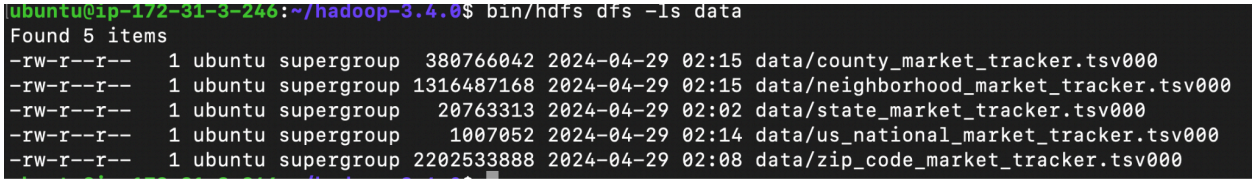
- Number of records:
    - county.count() 563123
    - neighborhood.count( ) 9002882
    - state.count() 27079

    - national.count() 1321
    - zip.count() 5310673
- Length of each record/attributes: 59 columns
- Record Details: housing market data for a specific region(county, neighborhood, state, zip code, national) within a given time period.
- Significant attributes:
    - Location: region_type_id, region_type, region, city, state, state_code
    - Time: period_begin, period_end, period_duration, months_of_supply
    - Price: median_sale_price, median_sale_price_mom, median_sale_price_yoy, median_price, average_sale_list, price_drops, sold_above_list
    - Housing Data: home_sold, new_listing, pending_sales, inventory
- example data

```
>>> data.select([c for c in data.columns if c in ['period_begin','state','median_sale_price','pending_sales']]).show()
+------------+--------------+-----------------+-------------+
|period_begin|         state|median_sale_price|pending_sales|
+------------+--------------+-----------------+-------------+
|  2019-10-01|      Oklahoma|           162200|           47|
|  2021-07-01|       Vermont|           317900|          909|
|  2016-08-01| New Hampshire|           200100|          200|
|  2013-04-01|   Mississippi|           129500|          403|
|  2019-12-01|      Missouri|           152000|         NULL|
|  2019-07-01|    New Mexico|           385500|          207|
|  2015-01-01|      New York|           219500|         1771|
|  2014-02-01|      Columbia|           678800|           75|
|  2018-10-01|      Columbia|           620000|           18|
|  2020-03-01|          Ohio|           156100|          855|
|  2012-04-01|    New Mexico|           312700|           32|
|  2012-02-01|      Delaware|           105000|           16|
|  2017-12-01|      Maryland|           185600|          344|
|  2015-10-01| Massachusetts|           360400|         3668|
|  2014-08-01|        Hawaii|           408300|            3|
|  2014-07-01|South Carolina|           117800|          244|
|  2013-12-01| New Hampshire|           166000|           58|
|  2021-10-01|       Georgia|           295300|         1069|
|  2021-07-01| Massachusetts|           660500|          678|
|  2020-03-01|     Minnesota|           232700|          844|
+------------+--------------+-----------------+-------------+
only showing top 20 rows

>>>
```

Running time evaluation:

The following files were uploaded into the virtual machine:



1. neighborhood_market_tracker.tsv000
    a. This file took the longest to upload at around 5-7 minutes. This file does have a size of 4,801,822 KB, making it the largest file in the dataset. This was a cause of it taking longer than the other 4 files to upload.
2. zip_code_market_tracker.tsv000
    a. This file took about 3 minutes to upload, coming in at 2,957,839 KB. Because of the size of this file, it did take a long time to upload, but its size did not affect the speed at which we were able to retrieve data from the file.
3. county_market_tracker.tsv000
    a. This file took 2 minutes to upload as it does have a smaller file size compared to the previous two coming in at 371,842 KB.
4. state_market_tracker.tsv000
    a. This file took half a minute to upload as it is significantly smaller compared to the other files at 20,277 KB.
5. us_national_market_tracker.tsv000
    a. This file took a few seconds to upload as it is the smallest file in the dataset. This file has a size of 984KB, making it upload significantly quicker than the other files.

Cleaning the data:

The longest part of the process of cleaning the data was figuring out which commands to use to filter and clean the data appropriately. After that, when using the commands that we used to filter the data, we were able to use these commands and they worked almost instantaneously with little to no wait time when calling functions and commands to filter the data. This process went smoothly and did not take as long as it took to upload the files when it comes to response time and runtime.

Part 2 (REVISED):

Analysis and Creating DataFrames For ML(Nate and Harjyot):

For our data analysis, we are analyzing and preprocessing our dataset based on the 8 topics in our project proposal: Predictive Housing Market Analysis, Communal Socioeconomic Class, Socioeconomic Potential Prediction, Industry Potential Prediction, Housing Market Trajectory, Price Deduction Analysis, Population Prediction, and Analysis of Time Spent on the Market. Using pyspark dataframes and sql operations, we used the schema of our cleaned dataset and used the columns that apply to each of the 8 topics listed before and showing the executions time.

Schema:

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import functions as F
>>> spark = SparkSession.builder.appName("zip_eda").getOrCreate()
tput_cleaned/zip_code_market.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
df.printSchema()
```

```
>>> df.printSchema()
root
 |-- period_begin: date (nullable = true)
 |-- period_end: date (nullable = true)
 |-- period_duration: integer (nullable = true)
 |-- region_type: string (nullable = true)
 |-- region_type_id: integer (nullable = true)
 |-- table_id: integer (nullable = true)
 |-- is_seasonally_adjusted: string (nullable = true)
 |-- region: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- state_code: string (nullable = true)
 |-- property_type: string (nullable = true)
 |-- property_type_id: integer (nullable = true)
 |-- median_sale_price: double (nullable = true)
 |-- median_sale_price_mom: double (nullable = true)
 |-- median_sale_price_yoy: double (nullable = true)
 |-- median_list_price: double (nullable = true)
 |-- median_list_price_mom: double (nullable = true)
 |-- median_list_price_yoy: double (nullable = true)
 |-- median_ppsf: double (nullable = true)
 |-- median_ppsf_mom: double (nullable = true)
 |-- median_ppsf_yoy: double (nullable = true)
 |-- median_list_ppsf: double (nullable = true)
 |-- median_list_ppsf_mom: double (nullable = true)
 |-- median_list_ppsf_yoy: double (nullable = true)
 |-- homes_sold: double (nullable = true)
 |-- homes_sold_mom: double (nullable = true)
 |-- homes_sold_yoy: double (nullable = true)
 |-- pending_sales: double (nullable = true)
 |-- pending_sales_mom: double (nullable = true)
 |-- pending_sales_yoy: double (nullable = true)
 |-- new_listings: double (nullable = true)
 |-- new_listings_mom: double (nullable = true)
 |-- new_listings_yoy: double (nullable = true)
 |-- inventory: double (nullable = true)
 |-- inventory_mom: double (nullable = true)
 |-- inventory_yoy: double (nullable = true)
 |-- months_of_supply: double (nullable = true)
 |-- months_of_supply_mom: double (nullable = true)
 |-- months_of_supply_yoy: double (nullable = true)
 |-- median_dom: double (nullable = true)
 |-- median_dom_mom: double (nullable = true)
 |-- median_dom_yoy: double (nullable = true)
 |-- avg_sale_to_list: double (nullable = true)
 |-- avg_sale_to_list_mom: double (nullable = true)
```

```
 |-- sold_above_list_yoy: double (nullable = true)
 |-- price_drops: double (nullable = true)
 |-- price_drops_mom: double (nullable = true)
 |-- price_drops_yoy: double (nullable = true)
 |-- off_market_in_two_weeks: double (nullable = true)
 |-- off_market_in_two_weeks_mom: double (nullable = true)
 |-- off_market_in_two_weeks_yoy: double (nullable = true)
 |-- parent_metro_region: string (nullable = true)
 |-- parent_metro_region_metro_code: integer (nullable = true)
 |-- last_updated: timestamp (nullable = true)
```

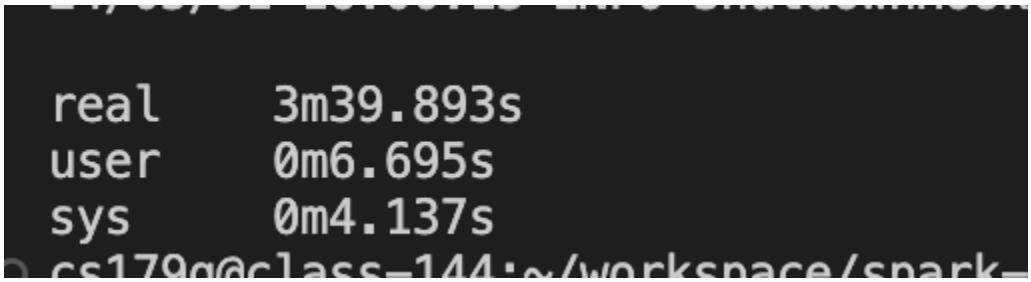
Preprocessing for ML (Nate):

To make it easier to perform machine learning on our dataset, the first step in this part of the project was to preprocess our dataset. This was done by creating a list of the names of columns that Gurman and Ibrahim need to perform ML for the different topics. Then using the select function from pyspark.sql filling in a new dataframe with the columns. Finally, I wrote these dataframes into csv files into our virtual machine.

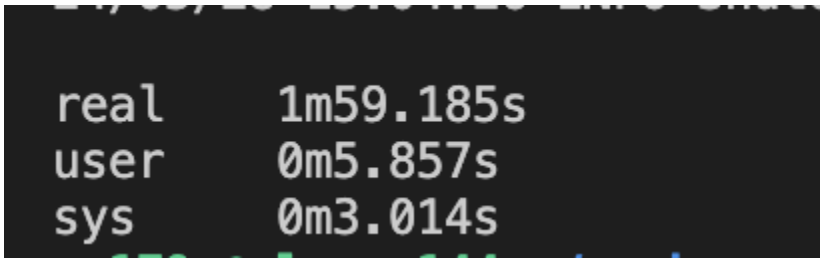
```
>>> columns_needed = ['period_begin', 'period_end', 'region_type', 'region', 'city', 'state', 'state_code', 'property_type', 'parent_metro_region', 'last_updated', 'price_drops', 'med
ian_sale_price', 'median_dom']
>>> filtered_df = df.select(columns_needed)
>>> filtered_df.printSchema()
root
 |-- period_begin: date (nullable = true)
 |-- period_end: date (nullable = true)
 |-- region_type: string (nullable = true)
 |-- region: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- state_code: string (nullable = true)
 |-- property_type: string (nullable = true)
 |-- parent_metro_region: string (nullable = true)
 |-- last_updated: timestamp (nullable = true)
 |-- price_drops: double (nullable = true)
 |-- median_sale_price: double (nullable = true)
 |-- median_dom: double (nullable = true)
```

```
>>> filetered_df.show(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'filetered_df' is not defined
>>> filtered_df.show(1)
+------------+----------+-----------+--------------------+----+--------+----------+-------------+-------------------+--------------------+-----------+-----------------+----------+
|period_begin|period_end|region_type|              region|city|   state|state_code|property_type|parent_metro_region|        last_updated|price_drops|median_sale_price|median_dom|
+------------+----------+-----------+--------------------+----+--------+----------+-------------+-------------------+--------------------+-----------+-----------------+----------+
|  2017-07-01|2017-09-30|   zip code|Zip Code: 60201|NULL|Illinois|        IL|    Townhouse|        Chicago, IL|2022-01-09 14:29:56|        0.0|         434000.0|      11.0|
+------------+----------+-----------+--------------------+----+--------+----------+-------------+-------------------+--------------------+-----------+-----------------+----------+
```

Time with 1 executor:



Time with 2 executors:



Housing Market Trajectory(Nate):

For the housing market trajectory, I created a new dataframe that takes the columns that will be used for machine learning to predict the trajectory of the housing market. We found that our dataset has a city column, however it is left with only null values so we removed this from the schema for housing market trajectory. After this, I then stored the dataframe into a csv file which will then be used as data for our machine learning predictors.

Housing Trajectory DataFrame Schema:

```
root
 |-- median_sale_price: double (nullable = true)
 |-- median_list_price: double (nullable = true)
 |-- median_ppsf: double (nullable = true)
 |-- period_begin: date (nullable = true)
 |-- homes_sold: double (nullable = true)
 |-- region: string (nullable = true)
 |-- period_end: date (nullable = true)
 |-- region_type: string (nullable = true)
 |-- region: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- state_code: string (nullable = true)
 |-- property_type: string (nullable = true)
 |-- parent_metro_region: string (nullable = true)
 |-- last_updated: timestamp (nullable = true)
```

Example Rows:

```
>>> housing_trajectory_df.show(5)
+-----------------+----------------+------------------+------------+----------+----------+-----------+--------------------+--------+----------+--------------------+-------------------+-------------------+
|median_sale_price|median_list_price|       median_ppsf|period_begin|homes_sold|period_end|region_type|              region|   state|state_code|       property_type|parent_metro_region|       last_updated|
+-----------------+----------------+------------------+------------+----------+----------+-----------+--------------------+--------+----------+--------------------+-------------------+-------------------+
|         434000.0|        339000.0| 251.0239445494644|  2017-07-01|      12.0|2017-09-30|   zip code|Zip Code: 60201|Illinois|        IL|           Townhouse|        Chicago, IL|2022-01-09 14:29:56|
|         489950.0|        489000.0|286.03777589872607|  2019-06-01|     140.0|2019-08-31|   zip code|Zip Code: 22310|Virginia|        VA|     All Residential|     Washington, DC|2022-01-09 14:29:56|
|         279000.0|        373500.0|             111.6|  2016-05-01|       1.0|2016-07-31|   zip code|Zip Code: 48208|Michigan|        MI|Multi-Family (2-4...|        Detroit, MI|2022-01-09 14:29:56|
|         450000.0|        475000.0| 143.3658771625477|  2013-03-01|      62.0|2013-05-31|   zip code|Zip Code: 20132|Virginia|        VA|     All Residential|     Washington, DC|2022-01-09 14:29:56|
|         125000.0|        124900.0| 83.26530612244898|  2014-10-01|       7.0|2014-12-31|   zip code|Zip Code: 35758| Alabama|        AL|           Townhouse|     Huntsville, AL|2022-01-09 14:29:56|
+-----------------+----------------+------------------+------------+----------+----------+-----------+--------------------+--------+----------+--------------------+-------------------+-------------------+
only showing top 5 rows
```

Time with 1 Executors:

```
24/05/31 16:06:24 INFO ShutdownHookMa

real    3m38.311s
user    0m6.114s
sys     0m1.967s
```

Time with 2 Executors:

```
real    2m18.631s
user    0m6.200s
sys     0m3.228s
```

Code Snippet:

```
#preprocessing dataset for machine learning
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import avg
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("preprocess_data").getOrCreate()
file_path = "/home/cs179g/workspace/output_cleaned/zip_code_market.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
df.printSchema()


columns_needed = ['period_begin', 'period_end', 'region_type', 'region', 'state', 'city', 'state_code', 'property_type', 'parent_metro_region',
filtered_df = df.select(columns_needed)
filtered_df.printSchema()
filtered_df.show(1)

#save dataframe as csv
filtered_df.write.csv("/home/cs179g/workspace/ml_analysis/filtered_data.csv", header=True, mode="overwrite")
```

For analysis, I computed the averages of median sale price and list price by state as well as the median price per square foot by state using groupBy and avg functions from pyspark.sql. Each of these are saved into there own data frames. The tables below shows the averages for sale price, list price, and price per square foot by both state and region(zip code) to help understand the housing market in different areas. The numbers below give important metrics that can be be used to see which areas are more affordable than others and get a better understanding of the housing market.

Average Sale Price by State:

```
>>> avg_sale_price_by_state = housing_trajectory_df.groupBy("state").avg("median_sale_price").orderBy("state")
>>> avg_sale_price_by_state.show()
+----------+-----------------------+
|     state|avg(median_sale_price)|
+----------+-----------------------+
|   Alabama|    152415.71673781888|
|    Alaska|    273094.24960842327|
|   Arizona|    231833.78904675628|
|  Arkansas|    146275.44628609615|
|California|     624561.7461420788|
|  Colorado|    321555.13908904855|
|  Columbia|     661767.1058630552|
|Connecticut|     278741.4093003135|
|  Delaware|    213819.75302167935|
|   Florida|     240894.929121823|
|   Georgia|    192695.60296629145|
|    Hawaii|     608897.7751445764|
|     Idaho|     246432.4648843754|
|  Illinois|    200930.83015565743|
|   Indiana|     147607.2316483019|
|      Iowa|    164309.74747971844|
|    Kansas|    204310.75669652905|
|  Kentucky|    162039.20242587602|
| Louisiana|     188898.972364187|
|     Maine|    244040.68155058287|
+----------+-----------------------+
only showing top 20 rows
```

The average sale price by state lists the average median sale price for homes in each state listed in alphabetical order.

Average Sale Price by Region:

```
>>> avg_sale_price_by_region = housing_trajectory_df.groupBy("region").avg("median_sale_price").orderBy("region")
>>> avg_sale_price_by_region.show()
+--------------+-----------------------+
|        region|avg(median_sale_price)|
+--------------+-----------------------+
|Zip Code: 00501|              251701.0|
|Zip Code: 01001|    236885.70569620252|
|Zip Code: 01002|     396270.9090909091|
|Zip Code: 01005|    204883.30405405405|
|Zip Code: 01007|     335404.9438202247|
|Zip Code: 01008|    228954.54545454544|
|Zip Code: 01009|    172107.14285714287|
|Zip Code: 01010|    293987.20930232556|
|Zip Code: 01011|    198071.42857142858|
|Zip Code: 01012|     525637.9310344828|
|Zip Code: 01013|    223019.91525423728|
|Zip Code: 01020|    200243.32795698923|
|Zip Code: 01022|    178468.33333333334|
|Zip Code: 01026|    297047.61904761905|
|Zip Code: 01027|              334349.0|
|Zip Code: 01028|    337298.94366197183|
|Zip Code: 01030|     289285.4166666667|
|Zip Code: 01031|    164263.59504132232|
|Zip Code: 01032|              319062.5|
|Zip Code: 01033|     320043.6046511628|
+--------------+-----------------------+
only showing top 20 rows
```

The average sale price by region shows the average median sale price by zip code, this is useful information to get a better understanding of housing prices within certain neighborhoods.

Average List Price by State:

```
>>> avg_list_price_by_state = housing_trajectory_df.groupBy("state").avg("median_list_price").orderBy("state")
>>> avg_list_price_by_state.show()
+-----------+-----------------------+
|      state|avg(median_list_price)|
+-----------+-----------------------+
|    Alabama|    152265.01702360372|
|     Alaska|     267471.2582666203|
|    Arizona|    236933.40004437132|
|   Arkansas|    138861.91918107812|
| California|     622835.0809863504|
|   Colorado|     352045.3574093419|
|   Columbia|     643569.4202136608|
|Connecticut|    290048.64848815126|
|   Delaware|    214272.16777514867|
|    Florida|    255015.82276751168|
|    Georgia|    196652.13267095265|
|     Hawaii|     627640.7165520475|
|      Idaho|     241120.3618163465|
|   Illinois|     206452.3769741858|
|    Indiana|    147716.27705537278|
|       Iowa|    155701.37192794084|
|     Kansas|    202419.56102282446|
|   Kentucky|     163521.3881594147|
|  Louisiana|    187262.63833049402|
|      Maine|    229335.02018042293|
+-----------+-----------------------+
only showing top 20 rows
```

This is the average listing price of homes by state, which can be used to compare with the sale price to see if there is a increase or decrease in the sale price compared to a homes listing price.

Average List Price by Region:

```
>>> avg_list_price_by_region = housing_trajectory_df.groupBy("region").avg("median_list_price").orderBy("region")
>>> avg_list_price_by_region.show()
+---------------+-----------------------+
|         region|avg(median_list_price)|
+---------------+-----------------------+
|Zip Code: 00501|    25713.333333333332|
|Zip Code: 01001|    209481.68987341772|
|Zip Code: 01002|    323503.63636363635|
|Zip Code: 01005|     208152.6554054054|
|Zip Code: 01007|     278056.1685393258|
|Zip Code: 01008|     272527.2727272727|
|Zip Code: 01009|    157571.42857142858|
|Zip Code: 01010|      266203.488372093|
|Zip Code: 01011|    202526.19047619047|
|Zip Code: 01012|    179567.24137931035|
|Zip Code: 01013|    200231.91525423728|
|Zip Code: 01020|    172735.47311827957|
|Zip Code: 01022|    175329.62962962964|
|Zip Code: 01026|    390080.95238095237|
|Zip Code: 01027|              317934.0|
|Zip Code: 01028|    306191.42253521126|
|Zip Code: 01030|    169323.95833333334|
|Zip Code: 01031|    144865.48140495867|
|Zip Code: 01032|           263268.75|
|Zip Code: 01033|    216610.46511627908|
+---------------+-----------------------+
only showing top 20 rows
```

This is the average listing price of homes by region(zip code), which can be used to compare with the sale price to see if there is an increase or decrease in the sale price compared to a home's listing price.

Median Price per Square Foot by State:

```
>>> median_ppsf_by_state = housing_trajectory_df.groupBy("state").agg({"median_ppsf": "median"}).orderBy("state")
>>> median_ppsf_by_state.show()
+----------+--------------------+
|     state|median(median_ppsf)|
+----------+--------------------+
|   Alabama|  81.32497618380499|
|    Alaska| 156.07014719297646|
|   Arizona|  128.0214861235452|
|  Arkansas|  79.86111111111111|
|California| 315.40222872277894|
|  Colorado| 156.97166921238167|
|  Columbia| 461.16611661166115|
|Connecticut|  131.7222600408441|
|  Delaware| 117.02728127939793|
|   Florida| 122.81561081495403|
|   Georgia|  90.42865813252078|
|    Hawaii| 409.76489445763815|
|     Idaho| 118.26911398864188|
|  Illinois| 104.92435313804185|
|   Indiana|  77.96052631578948|
|      Iowa| 107.93162776780372|
|    Kansas|  101.6566265060241|
|  Kentucky|  90.28855103940428|
| Louisiana| 103.00737412463434|
|     Maine| 126.20837808807734|
+----------+--------------------+
only showing top 20 rows
```

Median Price per Square Foot by Region:

```
>>> median_ppsf_by_region = housing_trajectory_df.groupBy("region").agg({"median_ppsf": "median"}).orderBy("region")
>>> median_ppsf_by_region.show()
+----------+--------------------+
|    region|median(median_ppsf)|
+----------+--------------------+
|Zip Code: 00501|                 0.0|
|Zip Code: 01001|   152.2744795682344|
|Zip Code: 01002|    227.169384057971|
|Zip Code: 01005| 120.57220708446866|
|Zip Code: 01007| 189.49228441004266|
|Zip Code: 01008| 200.81967213114754|
|Zip Code: 01009| 128.83928745159847|
|Zip Code: 01010| 132.87904599659285|
|Zip Code: 01011| 140.14618511569734|
|Zip Code: 01012| 197.72209567198178|
|Zip Code: 01013| 159.66777764933153|
|Zip Code: 01020| 147.53146551724137|
|Zip Code: 01022| 174.63235294117646|
|Zip Code: 01026| 170.47880451357122|
|Zip Code: 01027| 220.82018927444796|
|Zip Code: 01028| 167.13238101096135|
|Zip Code: 01030| 149.67763485208917|
|Zip Code: 01031|   98.87800240029841|
|Zip Code: 01032|   250.5446623093682|
|Zip Code: 01033| 169.04761904761904|
+----------+--------------------+
only showing top 20 rows
```

The median price per square foot is a variable that gives the amount of money it costs per square foot of a home. While this doesn't give information regarding the type of home, it does show the value of property in each state and given zip code. For example, in Indiana, the average price per square foot of a home is $77 per square foot which is low compared to California which is $315 per square foot.

Code Snippet:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg

# Create SparkSession
spark = SparkSession.builder.appName("housing_market_trajectory").getOrCreate()

# Load data
file_path = "/home/cs179g/workspace/output_cleaned/zip_code_market.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

# Repartition DataFrame to 2 partitions
df = df.repartition(2)

# Preprocessing: Select required columns
housing_trajectory_columns = ['median_sale_price', 'median_list_price', 'median_ppsf', 'period_begin', 'homes_sold', 'period_end', 'region_type', 'region', 'state', 'state_code', 'property_type', 'parent_metro_region', 'last_updated']
housing_trajectory_df = df.select(housing_trajectory_columns)

# Cache the DataFrame for reuse
housing_trajectory_df.cache()

# Save the DataFrame as CSV (optional)
housing_trajectory_df.write.csv("/home/cs179g/workspace/ml_analysis/housing_trajectory.csv", header=True, mode="overwrite")

# Analysis: Average sale price by state
avg_sale_price_by_state = housing_trajectory_df.groupBy("state").agg(avg("median_sale_price").alias("avg_median_sale_price")).orderBy("state")

# Write results to CSV
avg_sale_price_by_state.write.csv("/home/cs179g/workspace/data/average_sale_price_by_state.csv", header=True, mode="overwrite")

# Analysis: Average sale price by region
avg_sale_price_by_region = housing_trajectory_df.groupBy("region").agg(avg("median_sale_price").alias("avg_median_sale_price")).orderBy("region")
avg_sale_price_by_region.write.csv("/home/cs179g/workspace/data/average_sale_price_by_region.csv", header=True, mode="overwrite")

# Analysis: Average list price by state
avg_list_price_by_state = housing_trajectory_df.groupBy("state").agg(avg("median_list_price").alias("avg_median_list_price")).orderBy("state")
avg_list_price_by_state.write.csv("/home/cs179g/workspace/data/average_list_price_by_state.csv", header=True, mode="overwrite")

# Analysis: Average list price by region
avg_list_price_by_region = housing_trajectory_df.groupBy("region").agg(avg("median_list_price").alias("avg_median_list_price")).orderBy("region")
avg_list_price_by_region.write.csv("/home/cs179g/workspace/data/average_list_price_by_region.csv", header=True, mode="overwrite")

# Analysis: Median price per square foot by state
median_ppsf_by_state = housing_trajectory_df.groupBy("state").agg(avg("median_ppsf").alias("avg_median_ppsf")).orderBy("state")
median_ppsf_by_state.write.csv("/home/cs179g/workspace/data/median_ppsf_by_state.csv", header=True, mode="overwrite")

# Analysis: Median price per square foot by region
median_ppsf_by_region = housing_trajectory_df.groupBy("region").agg(avg("median_ppsf").alias("avg_median_ppsf")).orderBy("region")
median_ppsf_by_region.write.csv("/home/cs179g/workspace/data/median_ppsf_by_region.csv", header=True, mode="overwrite")

# Analysis: Average price per state by property type
average_df = housing_trajectory_df.groupBy("state", "property_type").agg(avg("median_sale_price").alias("avg_median_sale_price"), avg("median_list_price").alias("avg_median_list_price"), avg("median_ppsf").alias("avg_median_ppsf"))
average_df.write.csv("/home/cs179g/workspace/data/averages_by_prop_type_state.csv", header=True, mode="overwrite")

# Stop SparkSession
spark.stop()

```

Sale Price by Property Type and State:

For this part of analysis, I created a new dataframe called average_df. This dataframe includes the property type column and using functions to compute the averages for the different types of property that can be purchased.

```
>>> average_df = housing_trajectory_df.groupBy("state", "property_type").agg(avg("median_sale_price").alias("avg_median_sale_price"),avg("median_list_price").alias("avg_median_list_pr
ice"),avg("median_ppsf").alias("avg_median_ppsf"))
>>> average_df.show()
```

state	property_type	avg_median_sale_price	avg_median_list_price	avg_median_ppsf
Wisconsin	Condo/Co-op	175418.9510777235	161555.13291038436	117.22852812474487
New Jersey	All Residential	355375.6649818789	377362.75361533544	192.9971622863287
Nevada	Single Family Res...	318553.53441160807	333020.3776956869	160.84953385660603
Massachusetts	Multi-Family (2-4...	499198.9856035012	458122.3465333231	191.0469314058446
Oregon	Townhouse	300706.6052383402	257983.82167352538	200.67965738564024
Connecticut	Townhouse	665122.6678004535	465870.2709750567	342.38070274717455
Alabama	All Residential	152578.6650816666	157232.35238176468	81.55348365682369
Oregon	All Residential	295276.26733058196	299387.7781864944	176.26167378869158
Hawaii	Multi-Family (2-4...	832672.609437751	455083.8298192771	354.5325781918101
Colorado	Single Family Res...	354847.5105164972	398209.8543053625	156.64082420625854
Mississippi	Single Family Res...	135096.35603534512	141716.68922974324	72.5569239417085
Rhode Island	Single Family Res...	305781.75968629285	316644.9887042023	168.241827113768
Pennsylvania	Condo/Co-op	199432.8655380596	175656.4061043638	171.2402294920816
North Carolina	Townhouse	209710.70768619518	197104.86025428015	144.38775267243597
Texas	Townhouse	216953.94208969935	206056.16985652482	122.07063834430613
Maine	Condo/Co-op	229924.5268329554	202971.69551524313	199.152877626292
Kentucky	Single Family Res...	164693.36820592135	170556.23295956696	87.92685846493103
New Hampshire	Condo/Co-op	220225.19614791987	198644.11253210067	159.92147924122455
Alaska	Multi-Family (2-4...	368506.07887189294	348887.36424474185	136.37964546960896
Louisiana	Condo/Co-op	155931.44885427586	153477.02486146474	135.95989888874124

only showing top 20 rows

The table above shows the first 20 rows of the same metrics calculated before by property type to show how much one would pay on average in a particular state for each property. This provides more information that can be used to analyze the trends in the housing market with more context.

Price Deduction and Time on Market Analysis (Nate):

Table for Analysis:

median_sale_price	median_list_price	median_ppsf	period_begin	period_end	region	state	property_type	homes_sold	median_dom	last_updated
434000.0	339000.0	251.0239445494644	2017-07-01	2017-09-30	Zip Code: 60201	Illinois	Townhouse	12.0	11.0	2022-01-09 14:29:56
489950.0	489000.0	286.03777589872607	2019-06-01	2019-08-31	Zip Code: 22310	Virginia	All Residential	140.0	18.5	2022-01-09 14:29:56
279000.0	373500.0	111.6	2016-05-01	2016-07-31	Zip Code: 48208	Michigan	Multi-Family (2-4...	1.0	16.0	2022-01-09 14:29:56
450000.0	475000.0	143.3658771625477	2013-03-01	2013-05-31	Zip Code: 20132	Virginia	All Residential	62.0	28.0	2022-01-09 14:29:56
125000.0	124900.0	83.26530612244898	2014-10-01	2014-12-31	Zip Code: 35758	Alabama	Townhouse	7.0	207.0	2022-01-09 14:29:56
149951.0	170000.0	111.57068452380952	2019-08-01	2019-10-31	Zip Code: 40109	Kentucky	Single Family Res...	1.0	94.0	2022-01-09 14:29:56
375000.0	383750.0	174.59138187221396	2019-04-01	2019-06-30	Zip Code: 85207	Arizona	Single Family Res...	311.0	45.5	2022-01-09 14:29:56
188600.0	199450.0	112.5776397515528	2016-01-01	2016-03-31	Zip Code: 85742	Arizona	All Residential	143.0	85.5	2022-01-09 14:29:56
93500.0	0.0	116.875	2013-04-01	2013-06-30	Zip Code: 95930	California	Single Family Res...	1.0	0.0	2022-01-09 14:29:56
132000.0	131900.0	82.95509369762837	2015-07-01	2015-09-30	Zip Code: 66030	Kansas	Townhouse	6.0	68.0	2022-01-09 14:29:56
133000.0	299000.0	133.5740072202166	2019-08-01	2019-10-31	Zip Code: 21218	Maryland	Condo/Co-op	8.0	35.0	2022-01-09 14:29:56
390500.0	399000.0	219.11425187248702	2021-03-01	2021-05-31	Zip Code: 37211	Tennessee	Single Family Res...	230.0	17.0	2022-01-09 14:29:56
185950.0	292650.0	104.68670105929041	2016-03-01	2016-05-31	Zip Code: 77336	Texas	Single Family Res...	50.0	72.5	2022-01-09 14:29:56
824000.0	929450.0	386.39583333333337	2019-04-01	2019-06-30	Zip Code: 08243	New Jersey	Townhouse	42.0	107.0	2022-01-09 14:29:56
44400.0	0.0	37.0	2012-08-01	2012-10-31	Zip Code: 48650	Michigan	All Residential	1.0	136.0	2022-01-09 14:29:56
525000.0	0.0	210.0	2019-08-01	2019-10-31	Zip Code: 95776	California	Multi-Family (2-4...	1.0	53.0	2022-01-09 14:29:56
555000.0	0.0	342.17016029593094	2020-04-01	2020-06-30	Zip Code: 70165	Louisiana	Condo/Co-op	1.0	336.0	2022-01-09 14:29:56
154000.0	173000.0	81.01851851851852	2013-02-01	2013-04-30	Zip Code: 56308	Minnesota	All Residential	83.0	205.0	2022-01-09 14:29:56
237000.0	249920.0	185.09615384615384	2016-08-01	2016-10-31	Zip Code: 98204	Washington	All Residential	111.0	12.0	2022-01-09 14:29:56
40000.0	36000.0	42.3728813559322	2014-05-01	2014-07-31	Zip Code: 21217	Maryland	All Residential	77.0	49.5	2022-01-09 14:29:56

The table above shows the columns that will be used for this part of the analysis. The column median_dom stands for the median days on market.

Price Deduction:

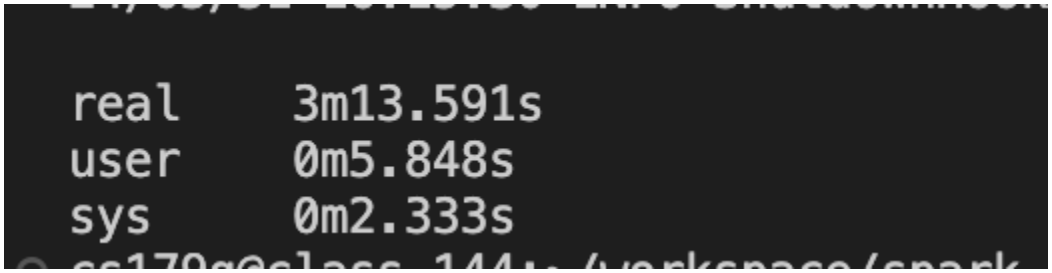
state	avg_price_change
Utah	2436.5706177072757
Hawaii	18742.941407471084
Minnesota	-1143.545110199794
Ohio	-12102.825851804342
Oregon	-10482.709905474983
Arkansas	-7413.527105018023
Texas	3245.3115254059235
Pennsylvania	-6101.306268771125
Connecticut	11307.239187837793
Nebraska	-7538.1202036524055
Vermont	-697.489354177845
Nevada	6953.183917148949
Washington	-6342.860653927637
Illinois	5521.5468185283835
Oklahoma	-1694.3151790141078
Delaware	452.41475346933555
Alaska	-5622.991341802994
New Mexico	-17888.819049951027
West Virginia	4016.964118198874
Missouri	-2428.9177486789704

This table shows the average price change by state. A positive price change indicates that the sale price was lower than the list price, meaning that properties were selling for less than their listed price.

region	avg_price_change
Zip Code: 60111	-73357.69230769231
Zip Code: 37353	-97230.39805825242
Zip Code: 03835	-6937.424657534247
Zip Code: 60643	12521.97191011236
Zip Code: 61849	3643.6170212765956
Zip Code: 27571	871.1770270270271
Zip Code: 92282	20945.72314049587
Zip Code: 48460	6860.0
Zip Code: 17363	-9696.94688221709
Zip Code: 18077	31043.005597014926
Zip Code: 64165	-24513.122641509435
Zip Code: 86303	-15749.470175438597
Zip Code: 90015	40386.858227848104
Zip Code: 95688	19940.86046511628
Zip Code: 35242	12679.573093220339
Zip Code: 11411	12045.384879725087
Zip Code: 74843	-139844.44444444444
Zip Code: 27513	3776.597888675624
Zip Code: 89149	12357.770127118643
Zip Code: 97473	-15782.263473053892

This provides the same information as the table above and calculates average price change by region. Both the tables show important information regarding the fluctuation in price change. This shows that either the prices for property are increasing or decreasing based on location.

Time with 1 Executor:



Time with 2 Executors:

```
real    1m57.120s
user    0m5.868s
sys     0m2.782s
```

Code Snippet:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import avg
from pyspark.sql import functions as F

spark = SparkSession.builder.appName("price_deduction_analysis").getOrCreate()
file_path = "/home/cs179g/workspace/output_cleaned/zip_code_market.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
df.printSchema()

#price deduction analysis code
#(ran in pyspark shell)
#median_dom = median days on market
price_deduct_columns = ["median_sale_price", "median_list_price", "price_drops", "median_ppsf", "period_begin", "period_end", "region", "state", "property_type", "homes_sold", "median_dom", "last_updated"]
price_deduction_df = df.select(price_deduct_columns)
price_deduction_df.cache()
price_deduction_df.show()

#price change by state
price_deduction_df = price_deduction_df.withColumn("price_change", col("median_list_price") - col("median_sale_price"))
avg_price_change_by_state = price_deduction_df.groupBy("state").agg(avg("price_change").alias("avg_price_change"))
avg_price_change_by_state.show()
avg_price_change_by_state.write.csv("/home/cs179g/workspace/data/avg_price_change_by_state.csv", header=True, mode="overwrite")


#price change by region
avg_price_change_by_region = price_deduction_df.groupBy("region").agg(avg("price_change").alias("avg_price_change"))
avg_price_change_by_region.show()
avg_price_change_by_region.write.csv("/home/cs179g/workspace/data/avg_price_change_by_region.csv", header=True, mode="overwrite")

spark.stop()

```

Time on Market:

For the time on market analysis, I created two dataframes that use the median_dom (median days on market) column to show how long properties stay on the market before being sold. The two tables show the median days on market by year and property type per year. These dataframes were then written to a csv file to be used for our database.

year_month	median_dom
2012-1	112.0
2012-10	91.0
2012-11	92.0
2012-12	94.0
2012-2	106.5
2012-3	98.0
2012-4	89.5
2012-5	86.0
2012-6	85.0
2012-7	87.0
2012-8	88.0
2012-9	89.5
2013-1	94.0
2013-10	79.5
2013-11	83.5
2013-12	88.0
2013-2	89.0
2013-3	81.0
2013-4	75.0
2013-5	72.0

only showing top 20 rows

The table above shows the median of the number of days properties stayed on the market before they were sold and the given year. Our dataset has housing market data from 2012 - 2022.

Days on Market By State and Property Type:

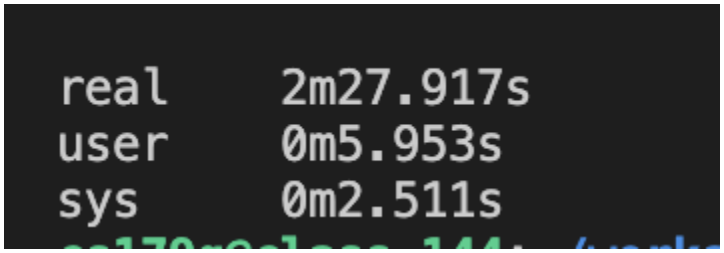
state	property_type	median_dom
Alabama	All Residential	88.5
Alabama	Condo/Co-op	79.0
Alabama	Multi-Family (2-4...	86.0
Alabama	Single Family Res...	88.5
Alabama	Townhouse	76.0
Alaska	All Residential	90.0
Alaska	Condo/Co-op	88.0
Alaska	Multi-Family (2-4...	94.0
Alaska	Single Family Res...	87.0
Arizona	All Residential	52.0
Arizona	Condo/Co-op	45.0
Arizona	Multi-Family (2-4...	51.5
Arizona	Single Family Res...	52.0
Arizona	Townhouse	44.0
Arkansas	All Residential	81.5
Arkansas	Condo/Co-op	80.0
Arkansas	Multi-Family (2-4...	67.0
Arkansas	Single Family Res...	81.0
Arkansas	Townhouse	76.5
California	All Residential	34.0

only showing top 20 rows

This table provides more information and gives the number of days on market by property type in each state. The table gives more context as to how long certain properties stay on the market before they are sold.

Time with 1 Executor:

Time with 2 Executors:



Code Snippet:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import avg
from pyspark.sql import functions as F
from pyspark.sql.functions import col, year, month

spark = SparkSession.builder.appName("time_on_market_analysis").getOrCreate()
file_path = "/home/cs179g/workspace/output_cleaned/zip_code_market.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
df.printSchema()

#showing the median days on market for properties by year
monthly_data = df.withColumn("year_month", F.concat(year("period_begin"), F.lit("-"), month("period_begin")))
monthly_stats = monthly_data.groupBy("year_month").agg(avg("price_drops").alias("avg_price_drops"),median("median_dom").alias("median_dom")).orderBy("year_month")
monthly_stats = monthly_stats.filter(col("avg_price_drops").isNull() | (col("avg_price_drops") == 0))
monthly_stats = monthly_stats.drop("avg_price_drops")
monthly_stats = monthly_stats.dropna(subset=["year_month", "median_dom"])
monthly_stats.show()
monthly_stats.write.csv("/home/cs179g/workspace/data/monthly_stats", header = True, mode = "overwrite")

#shows the median days on market of each property type by state
state_property_stats = df.groupBy("state", "property_type").agg(median("median_dom").alias("median_dom")).orderBy("state", "property_type")
state_property_stats = state_property_stats.dropna(subset=["state", "property_type", "median_dom"])
state_property_stats.show()
state_property_stats.write.csv("/home/cs179g/workspace/data/median_dom_by_property_type_state.csv", header = True, mode = "overwrite")

spark.stop()
```

SQL Examples:

Below are images showing all the tables that were entered into our database. Again, these tables come from the analysis above. And an example of what one of the tables looks like.

```
mysql> SHOW tables;
+-----------------------------------+
| Tables_in_group4_db               |
+-----------------------------------+
| average_list_price_by_region      |
| average_list_price_by_state       |
| average_sale_price_by_region      |
| average_sale_price_by_state       |
| avg_median_prices                 |
| avg_price_change                  |
| avg_price_change_region           |
| median_dom                        |
| median_dom_state_by_type          |
| median_prices_state_prop          |
| property_stats                    |
+-----------------------------------+
11 rows in set (0.01 sec)
```

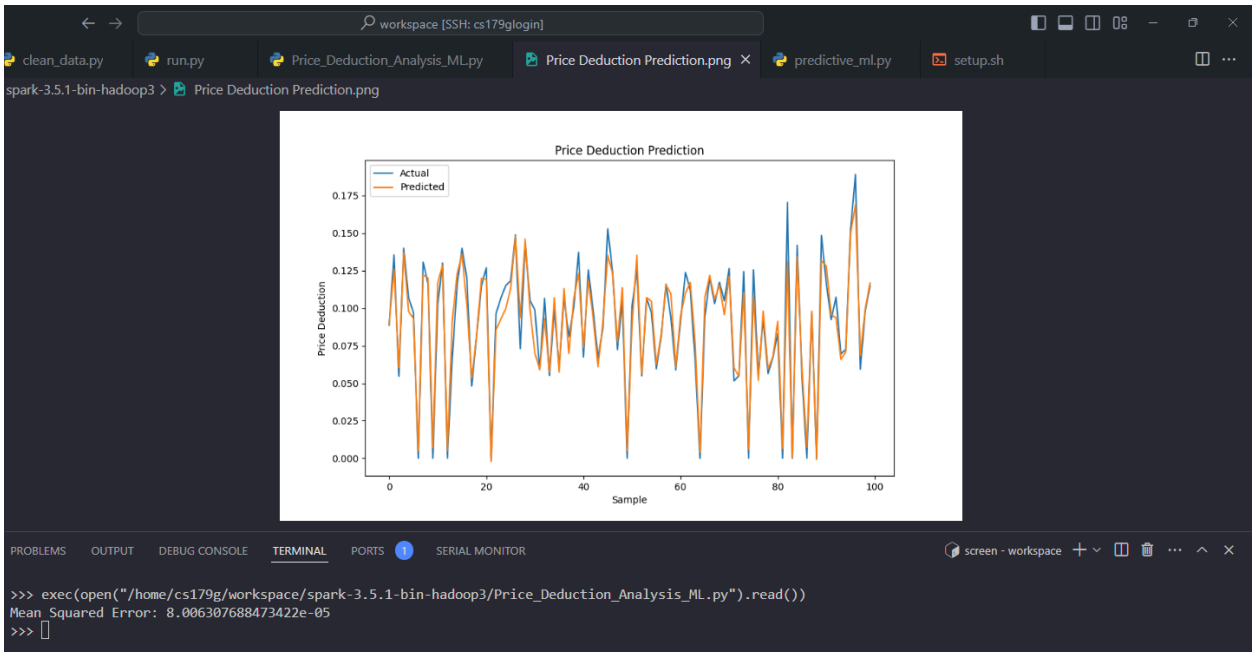
```
mysql> SELECT * FROM median_prices_state_prop;
+----------------+--------------------------+-----------------------+-----------------------+-----------------+
| state          | property_type            | avg_median_sale_price | avg_median_list_price | avg_median_ppsf |
+----------------+--------------------------+-----------------------+-----------------------+-----------------+
| New Jersey     | All Residential          |     355375.6649818789 |     377362.7536153354 |     192.9971622863 |
| Massachusetts  | Multi-Family (2-4 Unit)  |     499198.9856035012 |     458122.3465333231 |     191.0469314058 |
| Wisconsin      | Condo/Co-op              |     175418.9510777235 |     161555.1329103844 |     117.2285281247 |
| Nevada         | Single Family Residential |     318553.5344116081 |     333020.3776956869 |     160.8495338566 |
| Oregon         | Townhouse                |     300706.6052383402 |     257983.8216735254 |     200.6796573856 |
| Connecticut    | Townhouse                |     665122.6678004535 |     465870.2709750567 |     342.3807027472 |
| Oregon         | All Residential          |     295276.2673305820 |     299387.7781864944 |     176.2616737887 |
| Alabama        | All Residential          |     152578.6650816666 |     157232.3523817647 |      81.5534836568 |
| Hawaii         | Multi-Family (2-4 Unit)  |     832672.6094377510 |     455083.8298192771 |     354.5325781918 |
| Colorado       | Single Family Residential |     354847.5105164972 |     398209.8543053625 |     156.6408242063 |
| Mississippi    | Single Family Residential |     135096.3560353451 |     141716.6892297432 |      72.5569239417 |
| Rhode Island   | Single Family Residential |     305781.7596862929 |     316644.9857042023 |     168.2418271138 |
| Pennsylvania   | Condo/Co-op              |     199432.8655380596 |     175656.4061043638 |     171.2402294921 |
| North Carolina | Townhouse                |     209710.7076861952 |     197104.8602542802 |     144.3877526724 |
| Texas          | Townhouse                |     216953.9420896994 |     206056.1698565248 |     122.0706383443 |
| Maine          | Condo/Co-op              |     229924.5268329554 |     202971.6955152431 |     199.1528776263 |
| Kentucky       | Single Family Residential |     164693.3682059214 |     170556.2329595670 |      87.9268584649 |
| New Hampshire  | Condo/Co-op              |     220225.1961479199 |     198644.1125321007 |     159.9214792412 |
| Alaska         | Multi-Family (2-4 Unit)  |     368506.0788718929 |     348887.3642447419 |     136.3796454696 |
| New Hampshire  | Multi-Family (2-4 Unit)  |     234775.3057006092 |     188131.7338990427 |     196.0528711434 |
| Louisiana      | Condo/Co-op              |     155931.4488542759 |     153477.0248614647 |     135.9598988887 |
| Pennsylvania   | Multi-Family (2-4 Unit)  |     163489.4293804864 |     136776.6601799704 |     160.4803023540 |
| Michigan       | All Residential          |     158083.8891643479 |     166999.6361967912 |     105.6403649598 |
| New Jersey     | Multi-Family (2-4 Unit)  |     334592.8500227489 |     308248.6820531911 |     156.8588801941 |
| Michigan       | Townhouse                |     167694.3828125000 |      69968.4787946429 |     105.7139692660 |
| Indiana        | Townhouse                |     200409.9356435644 |     122155.4801980198 |     103.4376108322 |
| New Mexico     | All Residential          |     327875.7606721162 |     311131.8303814714 |     163.3195586340 |
| Utah           | All Residential          |     307284.4919110212 |     318204.1655993708 |     135.8940888832 |
| South Carolina | Single Family Residential |     239379.8383145091 |     250517.3225456125 |     115.9164484118 |
| Tennessee      | All Residential          |     203933.3430628158 |     209204.2199281605 |     125.4132424544 |
| Iowa           | Single Family Residential |     164558.0175196115 |     160084.4084049309 |     109.0458899238 |
| New York       | Townhouse                |     538832.0930034129 |     397214.3519733963 |     259.0435622978 |
| Rhode Island   | Condo/Co-op              |     236210.8810717372 |     243764.5638720830 |     173.2007464599 |
| Connecticut    | All Residential          |     298159.5872169746 |     320721.0035595222 |     149.0583366106 |
| Minnesota      | Condo/Co-op              |     161651.2938099390 |     152183.1324007292 |     144.3611318328 |
| Tennessee      | Condo/Co-op              |     191266.3596996133 |     180297.5402913931 |     134.1744300422 |
| South Carolina | Townhouse                |     200083.8393283864 |     195705.0730096843 |     124.1010086451 |
| Arizona        | Condo/Co-op              |     155470.8892450364 |     152379.9660173025 |     136.4202335100 |
| Arizona        | All Residential          |     250331.3101076374 |     262741.1316593094 |     137.7612983011 |
| Arkansas       | Multi-Family (2-4 Unit)  |     161855.1541115584 |     138181.5544853364 |      65.5541303312 |
| Washington     | Multi-Family (2-4 Unit)  |     440982.1499531689 |     389898.7963471745 |     185.1797540487 |
| Rhode Island   | All Residential          |     293605.6352023121 |     306096.0709826589 |     163.5747017615 |
| Missouri       | Townhouse                |     142120.9707941588 |     127723.7221444289 |     114.7881064351 |
| Utah           | Condo/Co-op              |     224518.4946088679 |     222835.6519419078 |     169.4893933978 |
| Vermont        | Single Family Residential |     252095.0048898203 |     262211.6749539595 |     131.6915227038 |
| Ohio           | Single Family Residential |     145799.8445285222 |     149334.8422943495 |      83.8028518242 |
| Pennsylvania   | Townhouse                |     188980.1019171158 |     183564.5787873403 |     114.1180483355 |
| North Carolina | Condo/Co-op              |     173380.0261540886 |     165619.2833622184 |     133.7637624705 |
| West Virginia  | Multi-Family (2-4 Unit)  |     166009.6181192661 |     108128.6972477064 |      67.7133668092 |
| Ohio           | Condo/Co-op              |     215352.5814672753 |     210630.7761675806 |      98.9033634209 |
```

Example script to push data from csv into MYSQL:

```python
import csv
import mysql.connector

# Connect to MySQL
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='group4',
    database='group4_db'
)
cursor = conn.cursor()

# Create MySQL table
create_table_sql = """
CREATE TABLE IF NOT EXISTS average_list_price_by_state (
    state VARCHAR(255),
    avg_median_list_price DECIMAL(20, 10)
)
"""
cursor.execute(create_table_sql)

# Open the CSV file
with open('/home/cs179g/workspace/data/average_list_price_by_state.csv', 'r') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader)  # Skip header row
    for row in csvreader:
        # Extract values from the row
        state = row[0]
        avg_median_list_price_str = row[1]
        # Skip rows with empty values in the second column
        if avg_median_list_price_str:
            avg_median_list_price = float(avg_median_list_price_str)  # Convert to float
            # Construct the INSERT INTO statement
            sql = "INSERT INTO average_list_price_by_state (state, avg_median_list_price) VALUES (%s, %s)"
            values = (state, avg_median_list_price)
            # Execute the INSERT INTO statement
            cursor.execute(sql, values)

# Commit changes and close connection
conn.commit()
cursor.close()
conn.close()

```

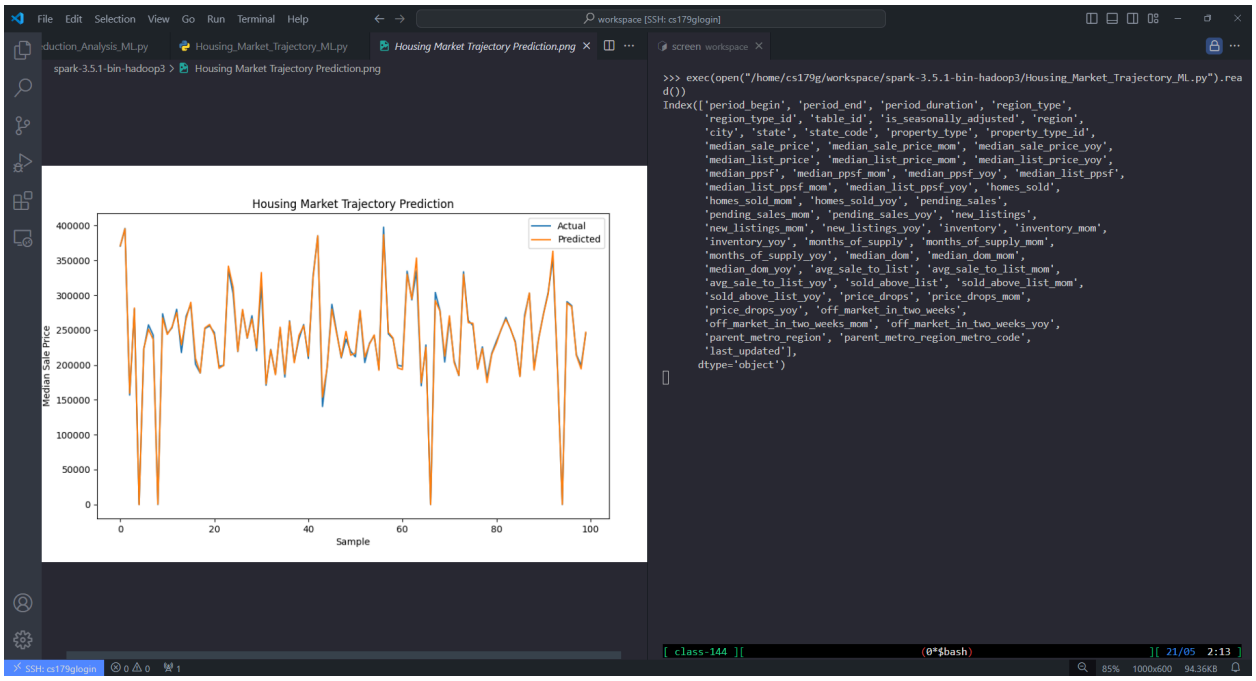
ML(Gurman and Ibrahim):

spark**:**



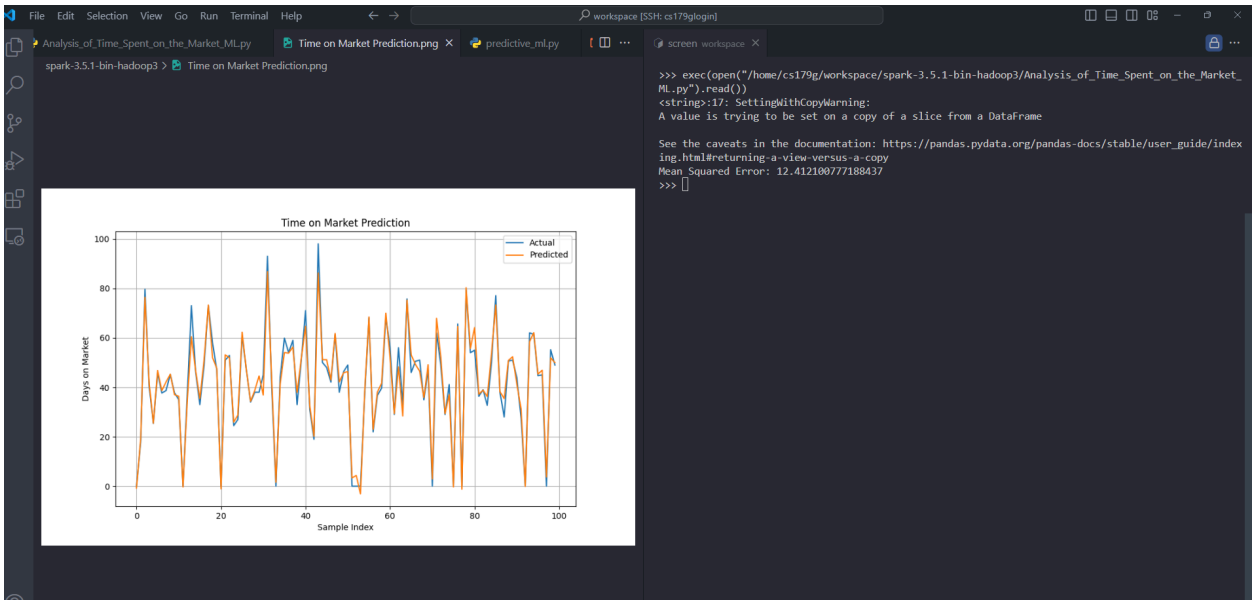
Based on the data in our datasets, we were able to make predictions on what will be the housing market price deduction trajectory over the next 100 samples while also displaying the actual recorded price deductions found in the dataset.

spark:



Based on the data in our datasets, we were able to make predictions on what will be the overall median sales price relative to the national housing market trajectory over the next 100 samples while also displaying the actually recorded price deductions found in the dataset.

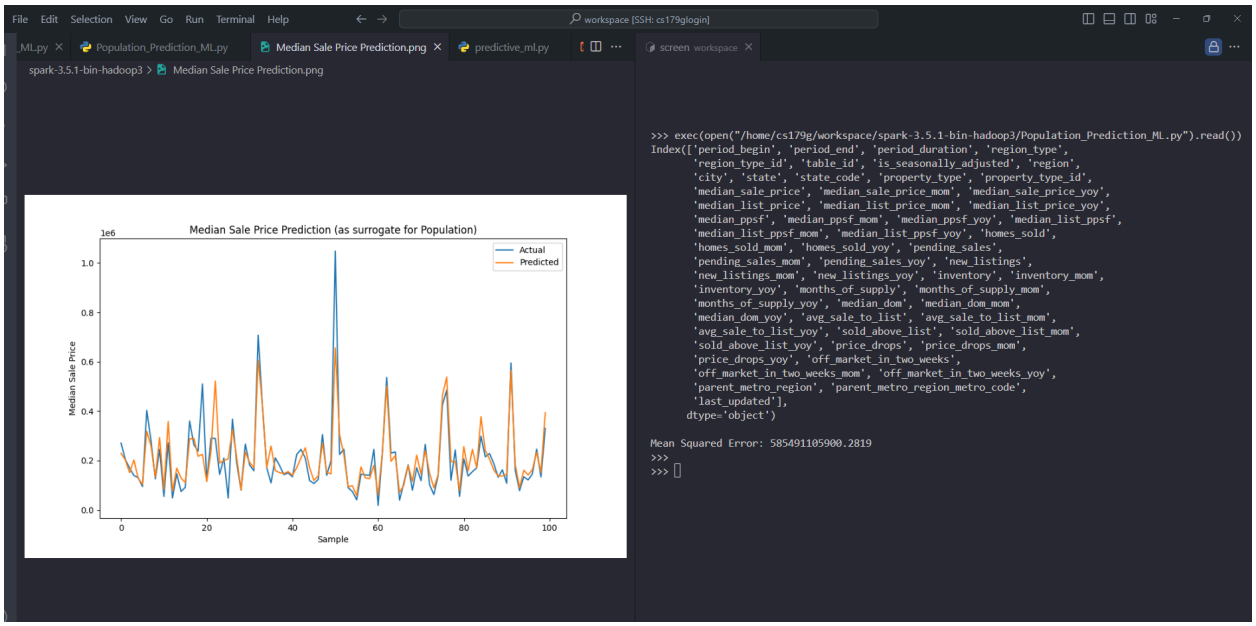
spark:



Based on the data in our datasets, we were able to make predictions on how long homes stay on the market over the next 100 samples while also displaying the actually recorded price deductions found in the dataset over the first 100 samples.

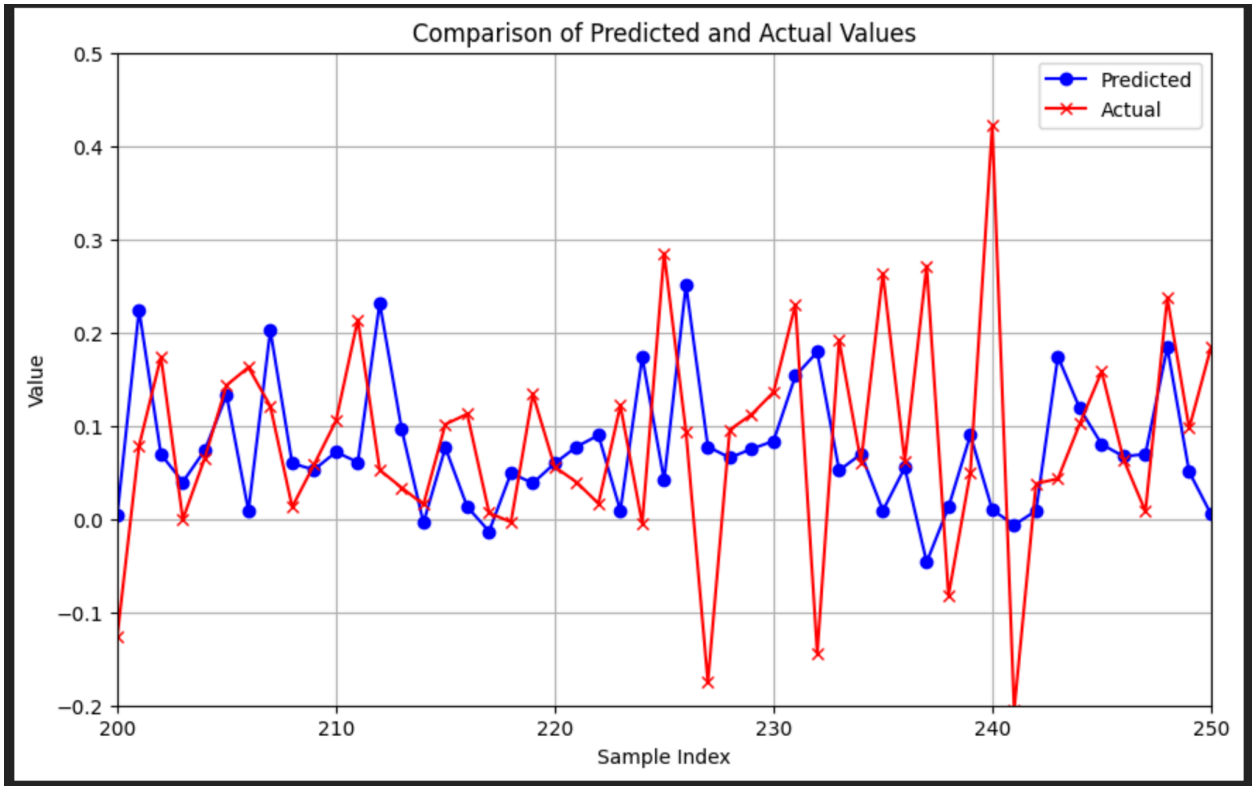
Population Prediction ML (using median sale price as a estimator)

spark:



Based on the data in our datasets, we were able to make predictions on populations using median sales price as a decision variable across multiple target variables. This predicts proportions of population increases in the dataset.

Predictive Housing Market Analysis ML (Gurman)



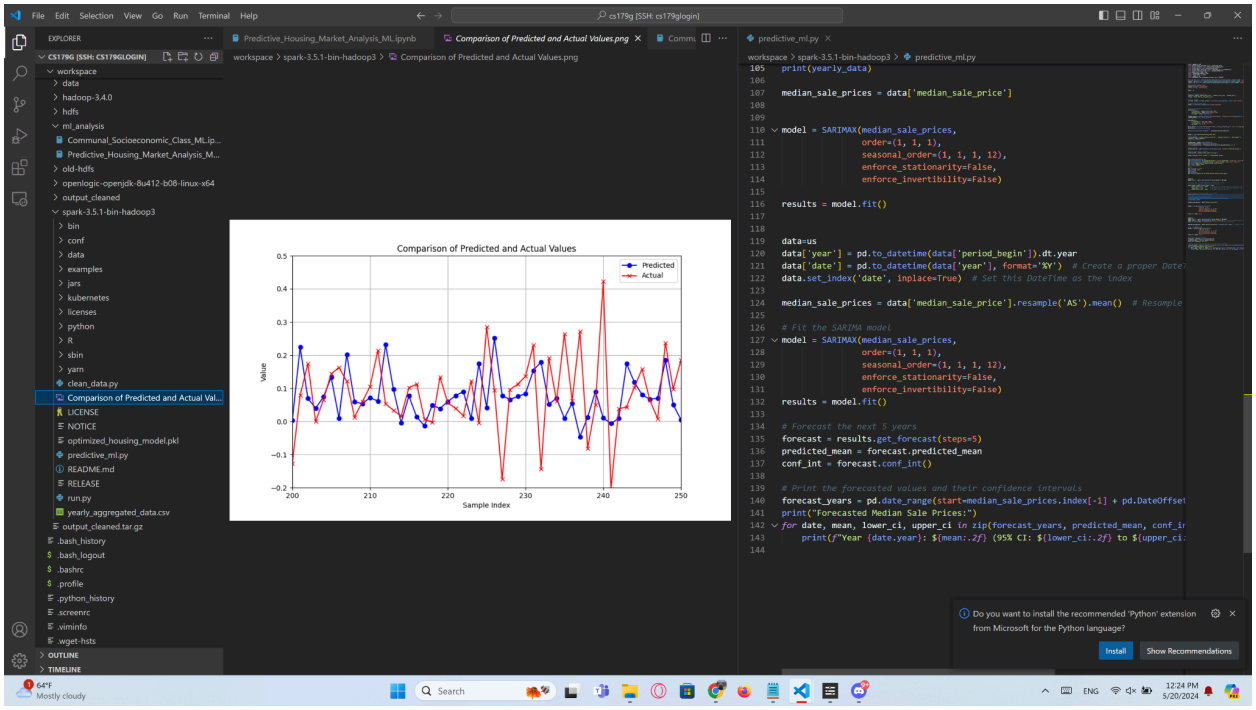
(gridsearch)

This illustrates the predicted model median price change yoy vs actual, the model is based on a different part of the set as the actual and shows a clear similar trend that we can expect for the next n number of samples.

```
[10]  ✓  0.0s

...   Forecasted Median Sale Prices:
      Year 2022: $320577.28 (95% CI: $317805.48 to $323349.09)
      Year 2023: $320577.28 (95% CI: $316657.36 to $324497.21)
      Year 2024: $426301.19 (95% CI: $421635.54 to $430966.84)
      Year 2025: $446270.77 (95% CI: $441210.17 to $451331.38)
      Year 2026: $459085.60 (95% CI: $454024.99 to $464146.20)
      C:\Users\gurma\AppData\Local\Temp\ipykernel_11252\3460894636.
```

spark:

```
screen data ×



           * * *

Machine precision = 2.220D-16
 N =            5     M =           10
 This problem is unconstrained.

At X0         0 variables are exactly at the bounds

At iterate    0    f= -0.00000D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5      0      1      0     0     0   0.000D+00  -0.000D+00
  F =  -0.0000000000000000

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
Forecasted Median Sale Prices:
Year 2022: $320577.28 (95% CI: $317805.48 to $323349.09)
Year 2023: $320577.28 (95% CI: $316657.36 to $324497.21)
Year 2024: $426301.19 (95% CI: $421635.54 to $430966.84)
Year 2025: $446270.77 (95% CI: $441210.17 to $451331.38)
Year 2026: $459085.60 (95% CI: $454024.99 to $464146.20)
>>>
```

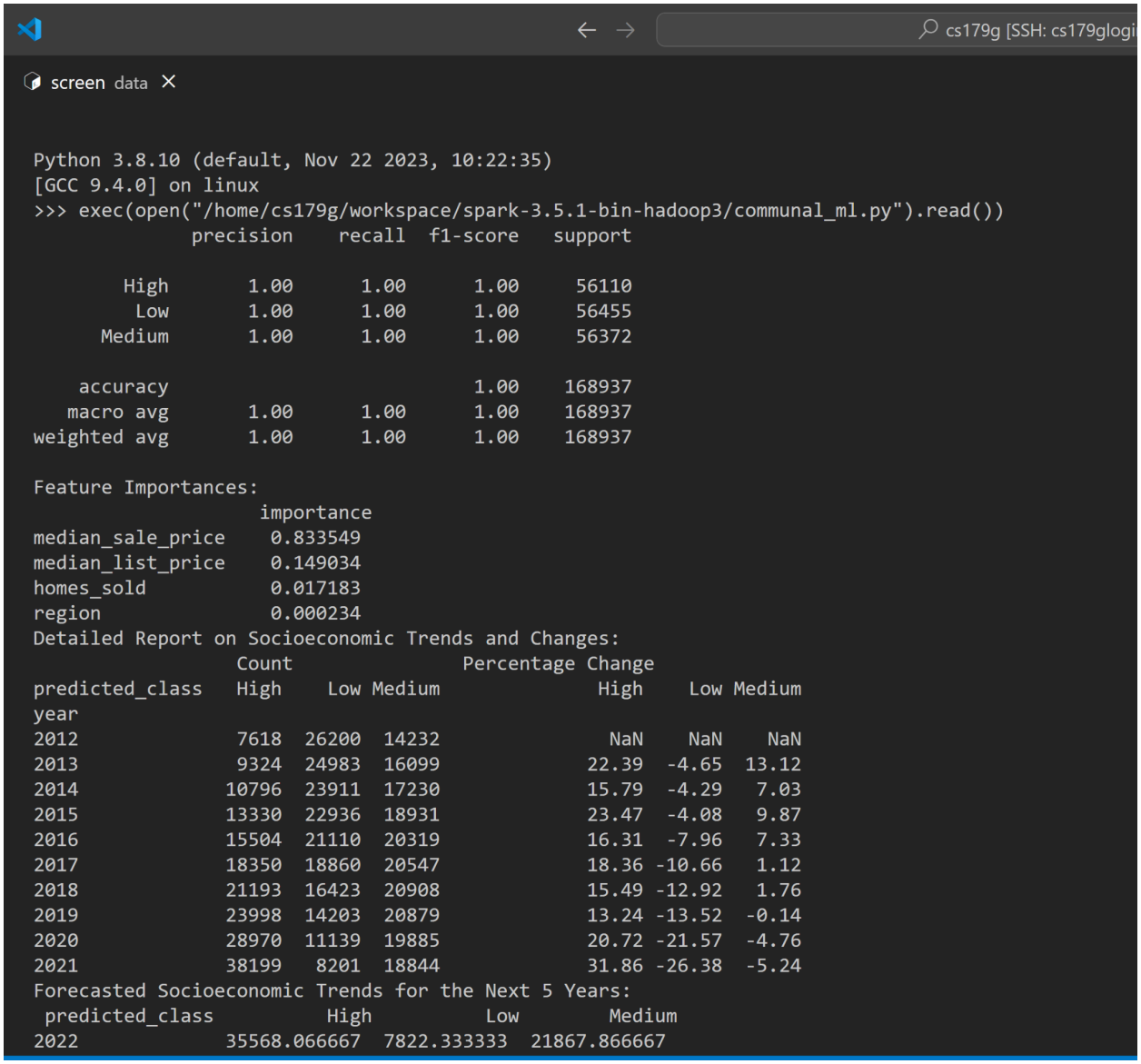


(SARIMA model forecast)

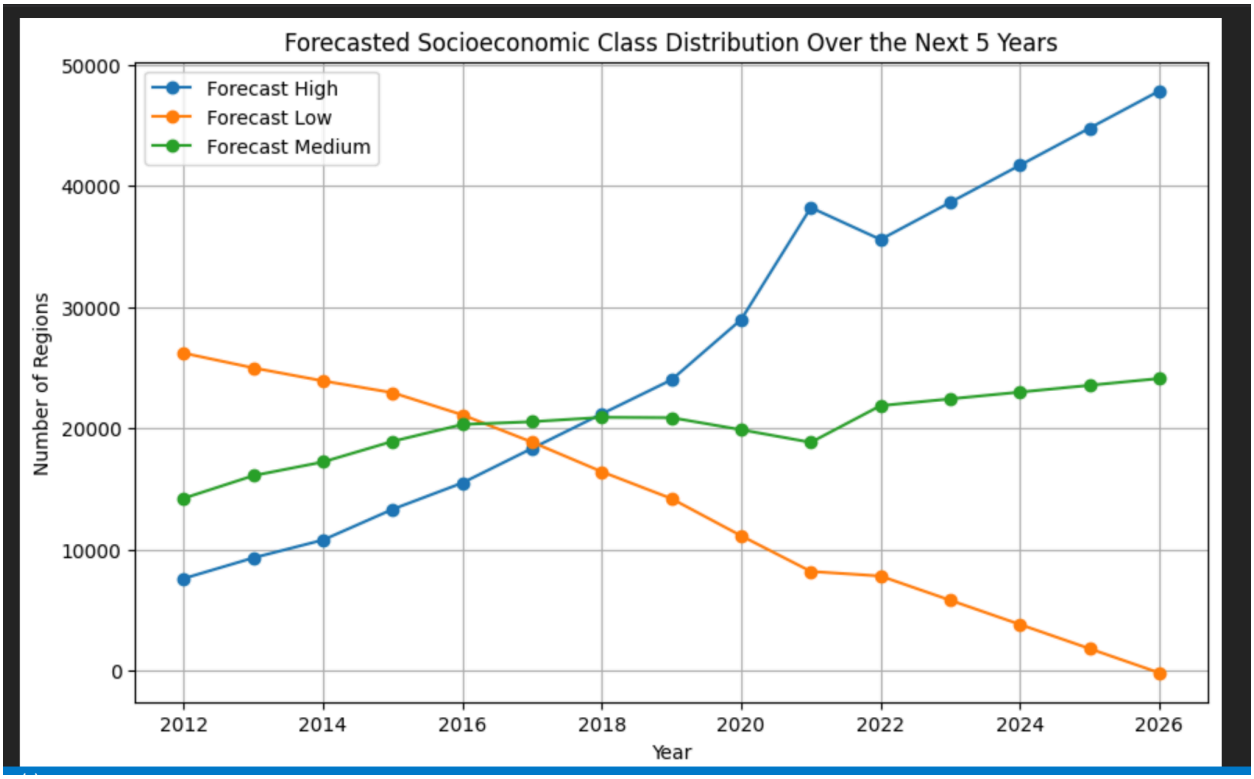
Here we have a forecast of projected median sales prices for the next few years. This follows the trend of the graph shown above.

Communal Socioeconomic Class ML and (Gurman)

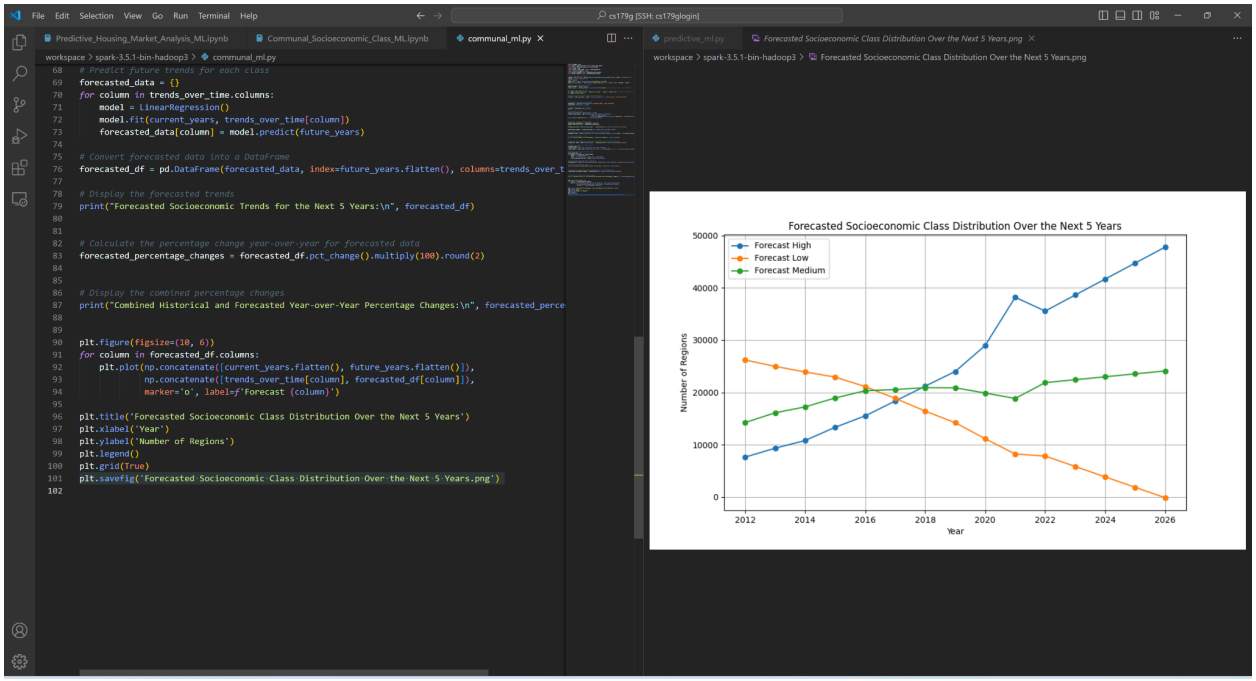
spark:


cs179g [SSH: cs179glogi
screen data
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
>>> exec(open("/home/cs179g/workspace/spark-3.5.1-bin-hadoop3/communal_ml.py").read())
precision recall f1-score support
High 1.00 1.00 1.00 56110
Low 1.00 1.00 1.00 56455
Medium 1.00 1.00 1.00 56372
accuracy 1.00 168937
macro avg 1.00 1.00 1.00 168937
weighted avg 1.00 1.00 1.00 168937
Feature Importances:
importance
median_sale_price 0.833549
median_list_price 0.149034
homes_sold 0.017183
region 0.000234
Detailed Report on Socioeconomic Trends and Changes:
Count Percentage Change
predicted_class High Low Medium High Low Medium
year
2012 7618 26200 14232 NaN NaN NaN
2013 9324 24983 16099 22.39 -4.65 13.12
2014 10796 23911 17230 15.79 -4.29 7.03
2015 13330 22936 18931 23.47 -4.08 9.87
2016 15504 21110 20319 16.31 -7.96 7.33
2017 18350 18860 20547 18.36 -10.66 1.12
2018 21193 16423 20908 15.49 -12.92 1.76
2019 23998 14203 20879 13.24 -13.52 -0.14
2020 28970 11139 19885 20.72 -21.57 -4.76
2021 38199 8201 18844 31.86 -26.38 -5.24
Forecasted Socioeconomic Trends for the Next 5 Years:
predicted_class High Low Medium
2022 35568.066667 7822.333333 21867.866667

```
2015             13330  22936  18931             23.47  -4.08   9.87
2016             15504  21110  20319             16.31  -7.96   7.33
2017             18350  18860  20547             18.36 -10.66   1.12
2018             21193  16423  20908             15.49 -12.92   1.76
2019             23998  14203  20879             13.24 -13.52  -0.14
2020             28970  11139  19885             20.72 -21.57  -4.76
2021             38199   8201  18844             31.86 -26.38  -5.24
Forecasted Socioeconomic Trends for the Next 5 Years:
 predicted_class          High          Low        Medium
2022             35568.066667  7822.333333  21867.866667
2023             38629.860606  5827.012121  22427.951515
2024             41691.654545  3831.690909  22988.036364
2025             44753.448485  1836.369697  23548.121212
2026             47815.242424  -158.951515  24108.206061
Combined Historical and Forecasted Year-over-Year Percentage Changes:
 predicted_class  High     Low  Medium
2022              NaN     NaN     NaN
2023             8.61  -25.51    2.56
2024             7.93  -34.24    2.50
2025             7.34  -52.07    2.44
2026             6.84 -108.66    2.38
>>>
[ class-144 ][                                                    (0*$bash
```



spark:



(linear regression)

This linear regression model forecasts how the trends will continue in terms of the growth and decline of the lower middle and upper classes, from the recorded current data projecting to the future based off of trends.

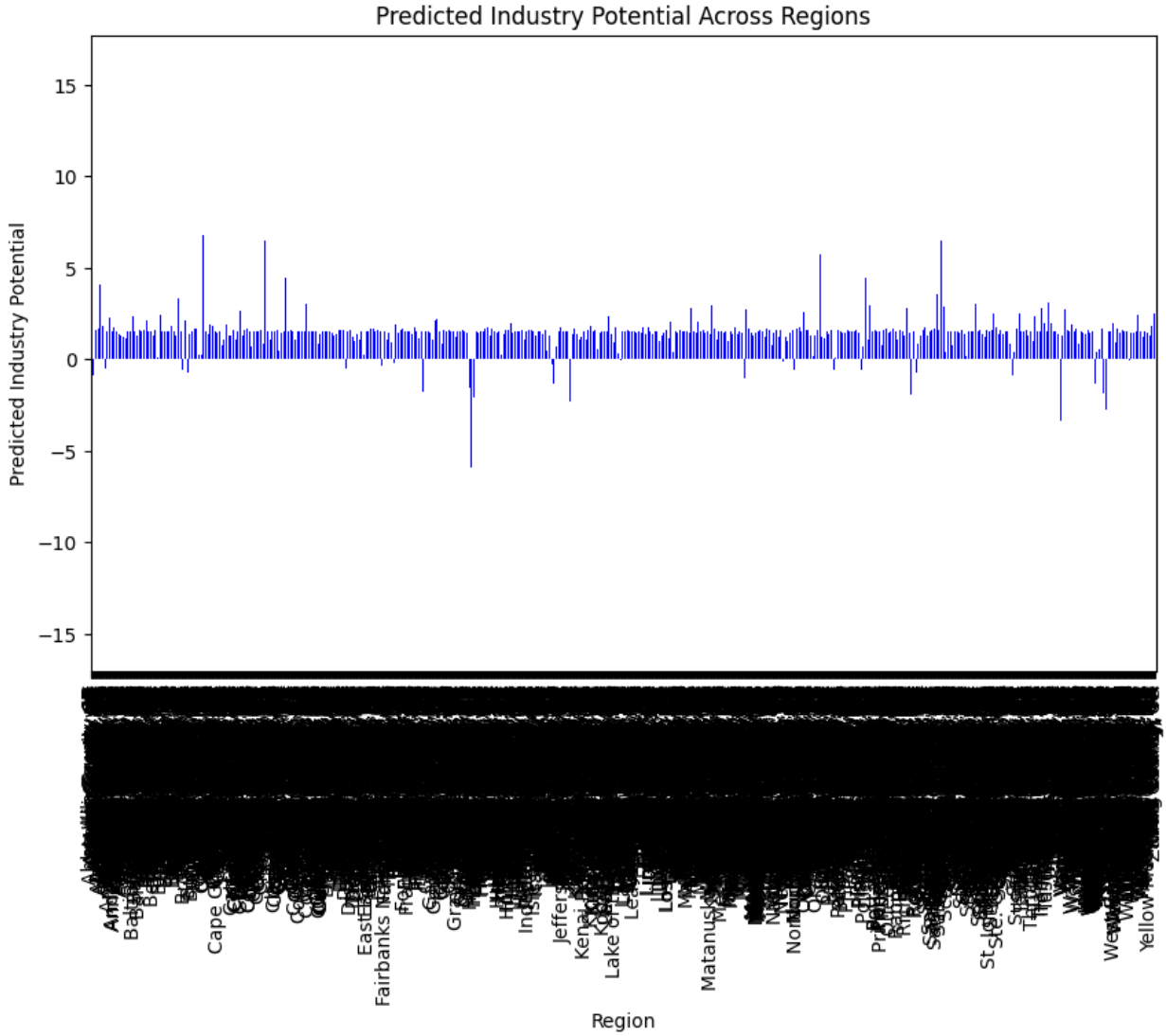
Socioeconomic Potential Prediction ML (Gurman)

```
...   year                      2012     2013     2014     2015     2016     2017  \
      region
      Abbeville County, SC      32.0     60.0     82.0     90.0    114.0     96.0
      Ada County, ID         14796.0  16746.0  16282.0  19150.0  21686.0  21972.0
      Adair County, IA          94.0     92.0    126.0     68.0     78.0     96.0
      Adair County, OK          66.0    102.0     90.0     94.0     92.0    128.0
      Adams County, CO       12726.0  15810.0  16591.0  17886.0  17872.0  18166.0

      year                      2018     2019     2020     2021  \
      region
      Abbeville County, SC      92.0    130.0    160.0    194.0
      Ada County, ID         22117.0  24468.0  26320.0  22898.0
      Adair County, IA         104.0     78.0    116.0    118.0
      Adair County, OK         112.0    176.0    190.0    200.0
      Adams County, CO       17546.0  19053.0  19996.0  20633.0

      year                  predicted_industry_potential
      region
      Abbeville County, SC                      1.550706
      Ada County, ID                           -4.300062
      Adair County, IA                          1.525083
      Adair County, OK                          1.545104
      Adams County, CO                         -0.860727
```

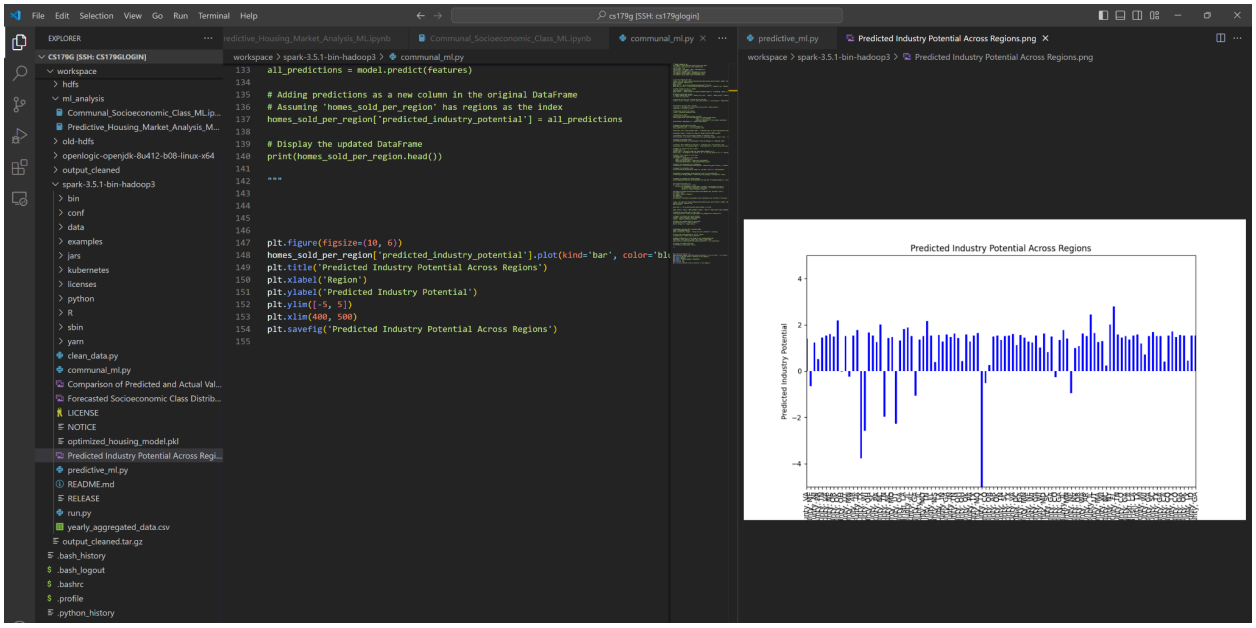
spark:

```
>>>
>>> exec(open("/home/cs179g/workspace/spark-3.5.1-bin-hadoop3/communal_ml.py").read())
year                   2012     2013     2014     2015     2016     2017     2018     2019     2020     2021  predicted_industry_potential
region
Abbeville County, SC     32.0     60.0     82.0     90.0    114.0     96.0     92.0    130.0    160.0    194.0                      1.550706
Ada County, ID        14796.0  16746.0  16282.0  19150.0  21686.0  21972.0  22117.0  24468.0  26320.0  22898.0                     -4.300062
Adair County, IA         94.0     92.0    126.0     68.0     78.0     96.0    104.0     78.0    116.0    118.0                      1.525083
Adair County, OK         66.0    102.0     90.0     94.0     92.0    128.0    112.0    176.0    190.0    200.0                      1.545104
Adams County, CO      12726.0  15810.0  16591.0  17886.0  17872.0  18166.0  17546.0  19053.0  19996.0  20633.0                     -0.860727
>>>
[ class-144 ][                                                    (0*$bash)
```

Industry Potential Prediction ML (gurman)



(linear regression)



This shows the industry potential of each city and region, the image before the one above includes all regions and therefore is difficult to read due to the size of the dataset but the above image is zoomed in to show one range. This uses linear regression to determine which regions are expected to grow and which are expected to decline based off of past and current trends from info such as sale prices and number of sales per region etc.

Part 3 Web Interface:

For our web interface, we decided to implement an interactive map with dropdown menus that would allow users to select from a certain table in our MYSQL database based on the information they want to see. Also, we have another page that contains the information from our Machine Learning Algorithms which also works with the map.

Set Up Web Application Using Flasks (Nate):

To set up our web application our group chose to use Flask for our backend which involves routing and querying from the MYSQL database. Then we decided to use javascript, html, and css for our frontend. For the setup, I made a new directory and organized it between

backend and frontend. Then, using github I initialized a remote repository within our VM to allow for all of us to be able to work on the project at the same time.

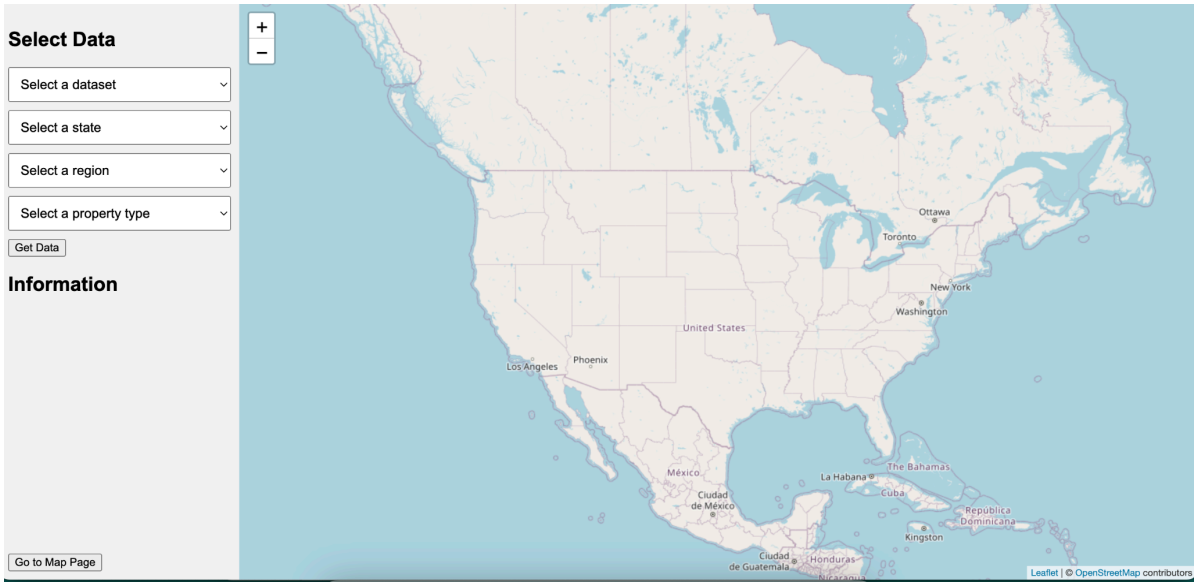
LINK TO REPO: https://github.com/gsinghd/group4project

Frontend Map Implementation(Nate):

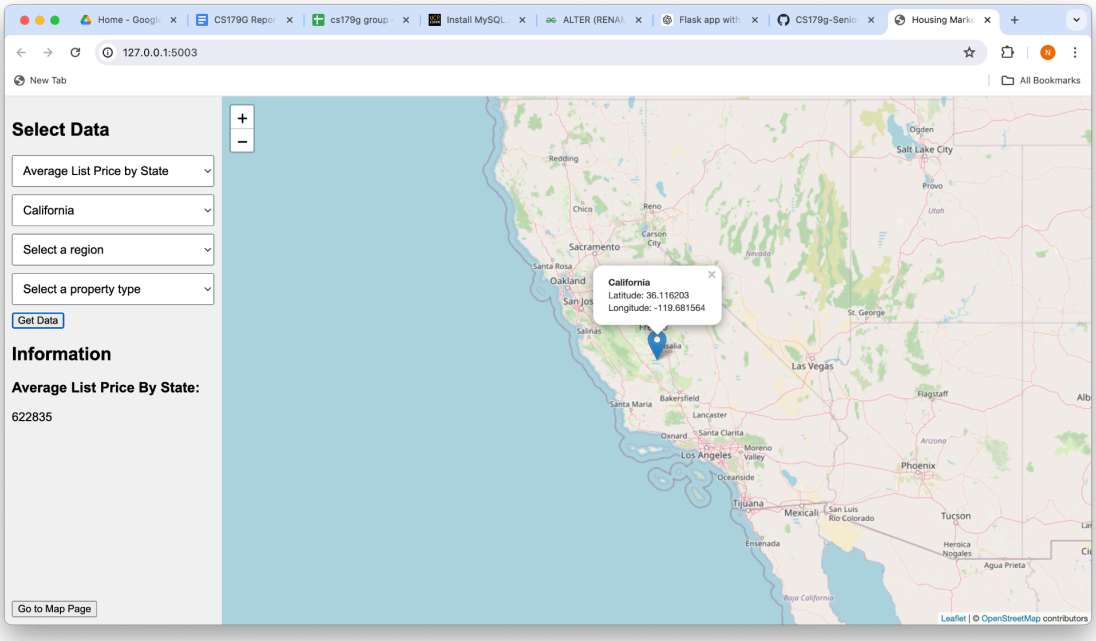
For our interactive map we utilized Leaflet.js which is a free open source JavaScript Library specifically for interactive maps. I added code to both the index.html and script.js to load Leaflet.js into our app, and then a function which initializes the map to show a view of the whole United States when first loading up the web application. Next, for our map interaction the user is prompted to select a dataset.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Housing Market Analysis</title>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
</head>
<body onload="initMap()">
    <div class="sidebar">
        <h2>Select Data</h2>
        <select id="datasetSelect">
            <option value="" disabled selected>Select a dataset</option>
        </select>
        <select id="stateSelect">
            <option value="" disabled selected>Select a state</option>
        </select>
        <select id="regionSelect">
            <option value="" disabled selected>Select a region</option>
        </select>
        <select id="typeSelect">
            <option value="" disabled selected>Select a property type</option>
        </select>
        <button onclick="populateData()">Get Data</button>
        <div>
            <h2>Information</h2>
            <div id="dataResponse"></div>
        </div>
    </div>
    <div class="main-content">
        <div id="map"></div>
        <div id="data-container"></div>
    </div>
    <button onclick="openMapPage()" style="position: fixed; left: 10px; bottom: 10px;">Go to Map Page</button>

    <script src="/static/script.js"></script>
</body>
</html>
```

Depending on the dataset, the user then selects either the state, region, or property type which they would like to see data for. Then data appears below the information header. Below is a screenshot of what our web application looks like.



Below is another screenshot with an example of how our interactive map moves based on the location selected by the user. You will see that based on the location of either state or region(zip code) the map will move to the chosen location on the map so the viewer can view the housing market data of a certain state or region.



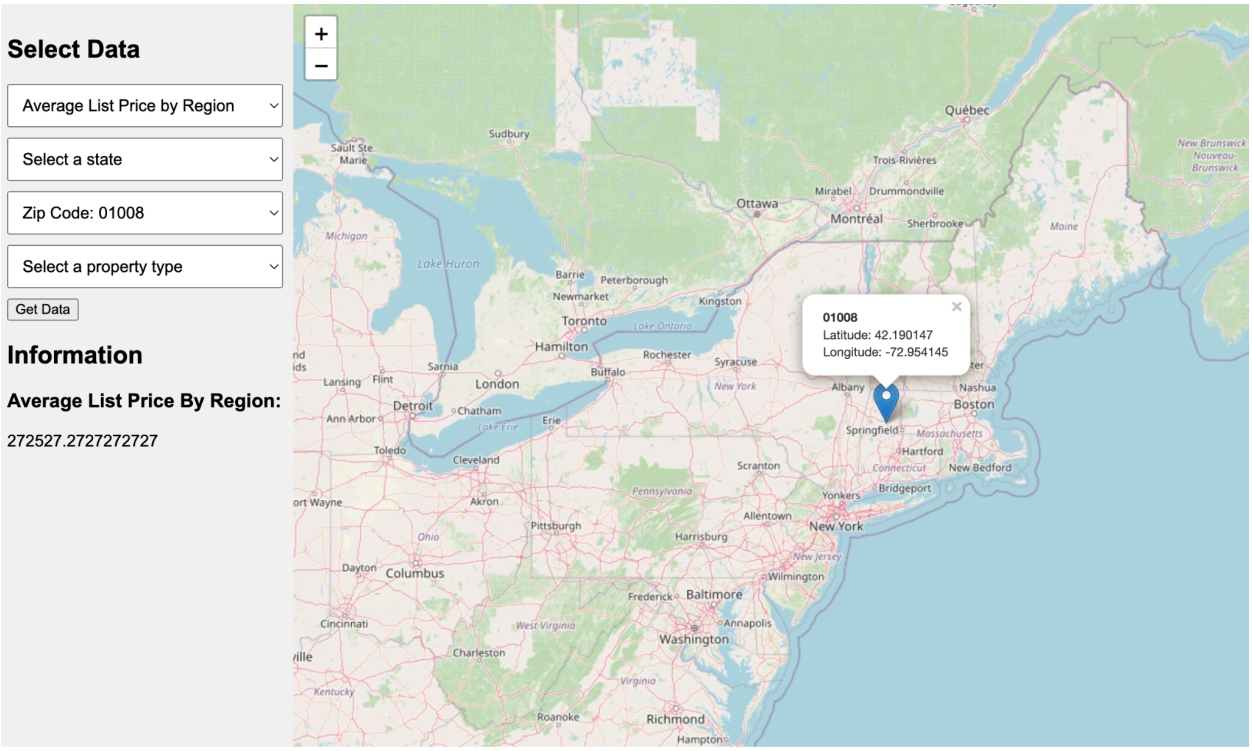
Code Snippet of Map Initialization and Populating the dropdown menu:

```
// script.js
var map;

function initMap() {
    // Initialize the map
    map = L.map('map').setView([39.8283, -98.5795], 4);

    // Add the OpenStreetMap tile layer
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(map);
}

const tableDisplayNames = {
    "average_list_price_by_region": "Average List Price by Region",
    "average_list_price_by_state": "Average List Price by State",
    "average_sale_price_by_region": "Average Sale Price by Region",
    "average_sale_price_by_state": "Average Sale Price by State",
    "avg_price_change_by_state": "Average Price Change by State",
    "avg_price_change_region": "Average Price Change by Region",
    "median_dom_state_by_type": "Median Days on Market by State and Property Type",
    "median_prices_state_prop_type": "Median Prices by State and Property Type"
};

const excludedDatasets = [
    "industry_potential_by_region",
    "regional_growth_by_region",
    "temp_table"
];

function openMapPage() {
    window.open('/us-map', '_blank');
}

function populateDatasetSelect() {
    const datasetSelect = document.getElementById('datasetSelect');

    fetch('/datasets')
        .then(response => response.json())
        .then(datasets => {
            datasets.forEach(dataset => {
                if (!excludedDatasets.includes(dataset)) {
                    const displayName = tableDisplayNames[dataset] || dataset;
                    const option = document.createElement('option');
                    option.value = dataset;
                    option.text = displayName;
                    datasetSelect.add(option);
                }
            });
        })
        .catch(error => console.error('Error:', error));
}

```

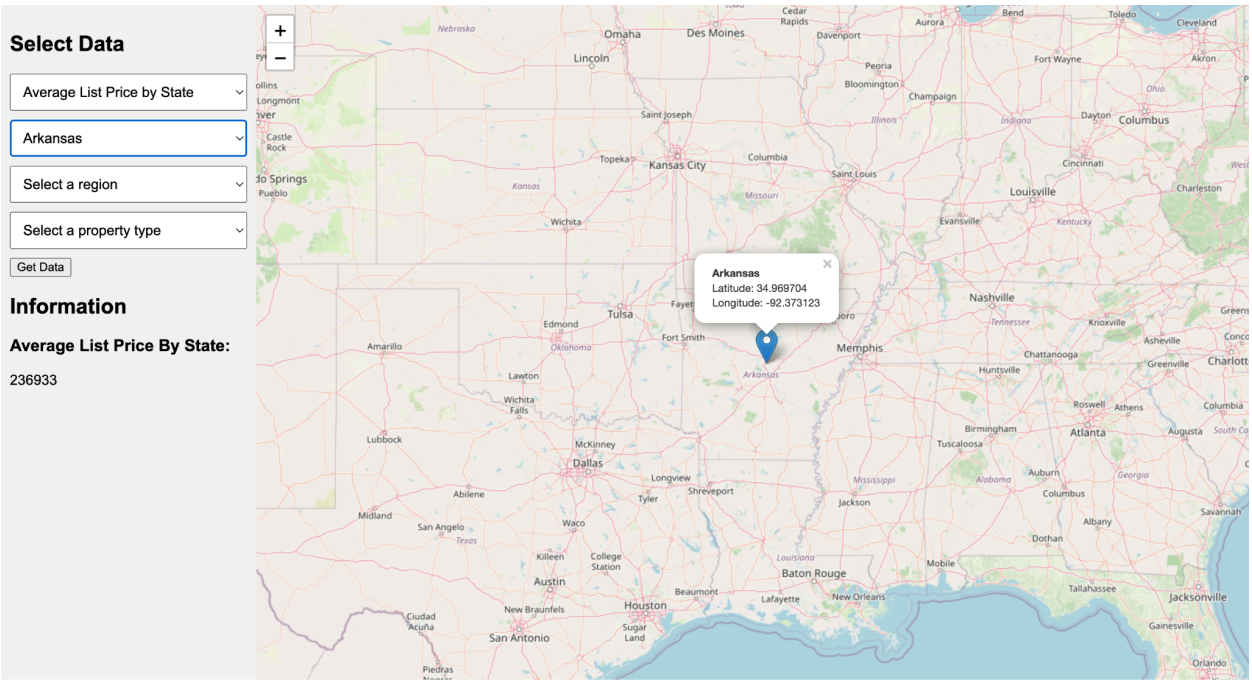
Backend Map Implementation(Ibrahim):

For the backend implementation of the map, we were able to route the data to longitude and latitude through an api. We used a ZipCode API and made an API ourselves for the states by just loading in the longitudes and latitudes of each state. This allowed for the map to move to the selected zipcode/region/state when prompted to do so and display the data for that state/region/zipcode.

ZipCode/Region Example:



State Example:



Using these APIs, we were able to make the map more interactive with the user, allowing for both and informational and visual aspect to be in effect for the user.

We had also included a pin marker that would navigate through the map and display the state/zipcode/region and the longitude and latitude of the selected location.

The “Get Data” button seen in the map above is the executor that we set to execute the command initiated by the user and fetch the data requested. This is done within our functions relative to data population and fetching found in our “script.js” file, as well as the querying found in our “app.py” file.

Map Updating Based Off Of User Choice (Ibrahim):

For Region:

```
function setRegionLocation(){
    var selectElement = document.getElementById('regionSelect');
    var selectedRegion = selectElement.value.slice(-5);
    fetch("/region_location/" + selectedRegion)
        .then(response => response.json())
        .then(data => {
            console.log(data.lat)
            map.eachLayer(function (layer) {
                if (layer instanceof L.Marker) {
                    map.removeLayer(layer);
                }
            });
            var marker = L.marker([data.lat, data.lng]).addTo(map);
            marker.bindPopup("<b>" + selectedRegion + "</b><br>Latitude: " + data.lat + "<br>Longitude: " + data.lng).openPopup();
            map.setView([ data.lat, data.lng], 6);
        })
        .catch(error => console.error('Error:', error));
}
```

For State:

```
function setStateLocation(){
    var selectElement = document.getElementById('stateSelect');
    var selectedState = selectElement.value;
    fetch("/state_location/" + selectedState)
        .then(response => response.json())
        .then(data => {
            map.eachLayer(function (layer) {
                if (layer instanceof L.Marker) {
                    map.removeLayer(layer);
                }
            });
            var marker = L.marker([data.latitude, data.longitude]).addTo(map);
            marker.bindPopup("<b>" + selectedState + "</b><br>Latitude: " + data.latitude + "<br>Longitude: " + data.longitude).openPopup();
            map.setView([ data.latitude, data.longitude], 6);
        })
        .catch(error => console.error('Error:', error));
}
```

Using this code, we were able to fetch the latitude and longitude of the region/state selected using the APIs we used. This includes “ZipCodeAPI” and an API that I manually made in the python script that includes the longitude and latitude of each state in the United States. This API looks like the following:

```
@app.route('/state_location/<state>')
def get_location(state):
    states = {
    "Alabama": {"latitude": 32.806671, "longitude": -86.79113},
    "Alaska": {"latitude": 61.370716, "longitude": -152.404419},
    "Arizona": {"latitude": 33.729759, "longitude": -111.431221},
    "Arkansas": {"latitude": 34.969704, "longitude": -92.373123},
    "California": {"latitude": 36.116203, "longitude": -119.681564},
    "Columbia": {"latitude": 39.2037, "longitude": -76.8610},
    "Colorado": {"latitude": 39.059811, "longitude": -105.311104},
    "Connecticut": {"latitude": 41.597782, "longitude": -72.755371},
    "Delaware": {"latitude": 39.318523, "longitude": -75.507141},
    "Florida": {"latitude": 27.766279, "longitude": -81.686783},
    "Georgia": {"latitude": 33.040619, "longitude": -83.643074},
    "Hawaii": {"latitude": 21.094318, "longitude": -157.498337},
    "Idaho": {"latitude": 44.240459, "longitude": -114.478828},
    "Illinois": {"latitude": 40.349457, "longitude": -88.986137},
    "Indiana": {"latitude": 39.849426, "longitude": -86.258278},
    "Iowa": {"latitude": 42.011539, "longitude": -93.210526},
    "Kansas": {"latitude": 38.5266, "longitude": -96.726486},
    "Kentucky": {"latitude": 37.66814, "longitude": -84.670067},
    "Louisiana": {"latitude": 31.169546, "longitude": -91.867805},
    "Maine": {"latitude": 44.693947, "longitude": -69.381927},
    "Maryland": {"latitude": 39.063946, "longitude": -76.802101},
    "Massachusetts": {"latitude": 42.230171, "longitude": -71.530106},
    "Michigan": {"latitude": 43.326618, "longitude": -84.536095},
    "Minnesota": {"latitude": 45.694454, "longitude": -93.900192},
    "Mississippi": {"latitude": 32.741646, "longitude": -89.678696},
    "Missouri": {"latitude": 38.456085, "longitude": -92.288368},
    "Montana": {"latitude": 46.921925, "longitude": -110.454353},
    "Nebraska": {"latitude": 41.12537, "longitude": -98.268082},
    "Nevada": {"latitude": 38.313515, "longitude": -117.055374},
    "New Hampshire": {"latitude": 43.452492, "longitude": -71.563896},
    "New Jersey": {"latitude": 40.298904, "longitude": -74.521011},
    "New Mexico": {"latitude": 34.840515, "longitude": -106.248482},
    "New York": {"latitude": 42.165726, "longitude": -74.948051},
    "North Carolina": {"latitude": 35.630066, "longitude": -79.806419},
    "North Dakota": {"latitude": 47.528912, "longitude": -99.784012},
    "Ohio": {"latitude": 40.388783, "longitude": -82.764915},
    "Oklahoma": {"latitude": 35.565342, "longitude": -96.928917},
    "Oregon": {"latitude": 44.572021, "longitude": -122.070938},
    "Pennsylvania": {"latitude": 40.590752, "longitude": -77.209755},
    "Rhode Island": {"latitude": 41.680893, "longitude": -71.51178},
    "South Carolina": {"latitude": 33.856892, "longitude": -80.945007},
    "South Dakota": {"latitude": 44.299782, "longitude": -99.438828},
    "Tennessee": {"latitude": 35.747845, "longitude": -86.692345},
    "Texas": {"latitude": 31.054487, "longitude": -97.563461},
    "Utah": {"latitude": 40.150032, "longitude": -111.862434},
    "Vermont": {"latitude": 44.045876, "longitude": -72.710686},
    "Virginia": {"latitude": 37.769337, "longitude": -78.169968},
    "Washington": {"latitude": 47.400902, "longitude": -121.490494},
    "West Virginia": {"latitude": 38.491226, "longitude": -80.954063},
    "Wisconsin": {"latitude": 44.268543, "longitude": -89.616508},
    "Wyoming": {"latitude": 42.755966, "longitude": -107.30249}
    }
    return states[state]
```

Using this we were able to update the map location using the latitude and longitude found by the User’s input into the state/region menu.

Querying/Routing(Ibrahim):

An example of how we routed the data:

```python
@app.route('/region_data/<dataset>/<region>')
def get_region_data(dataset, region):
    try:
        # Connect to MySQL database
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()

        # Query
        query = f"SELECT * FROM {dataset} WHERE region = \"{region}\""

        # Fetch the distinct states from the selected dataset
        cursor.execute(query)
        data = [row[1] for row in cursor.fetchall()]

        # Close the connection
        cursor.close()
        conn.close()

        return jsonify(data)
    except Exception as e:
        return jsonify({'error': 'Failed to fetch states', 'details': str(e)}), 500
```

We essentially followed the same template for all the data that was routed. We used these routes to fetch data from the datasets within our database as well as data from the APIs we used to fetch and display the location of the selected state/zipcode/region.

This was used extensively in fetching data from the datasets inside the ‘group4_db’ database, locating the rows and columns of the queried data, then displaying that data on screen.

Example of how data was fetched based on the dataset selected:

```
if (selectedDataset.includes('type')) {
    fetch(`/types/${selectedDataset}`)
        .then(response => response.json())
        .then((types) => {
            types.forEach(type => {
                console.log("Fetched types:", types);
                const option = document.createElement('option');
                option.value = type;
                option.text = type;
                typeSelect.add(option);
            });
        })
        .catch(error => console.error('Error:', error));
}
```

This is an example of how the data was fetched found in our script.js file.

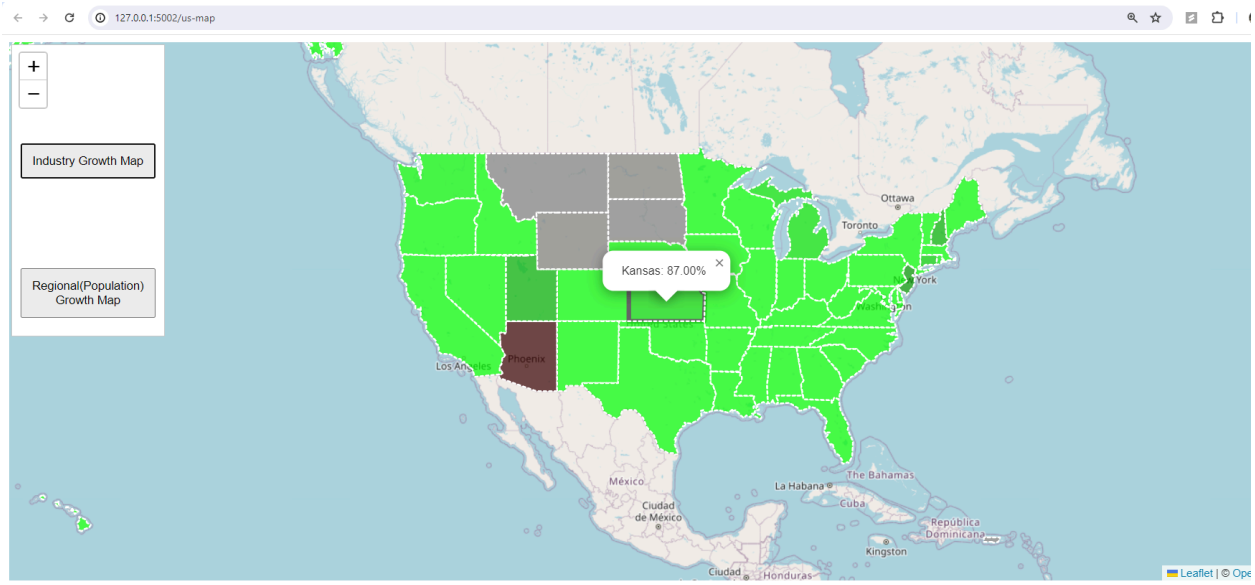
Steps to how we routed, queried, and handled errors (as applied to all routes and queries):

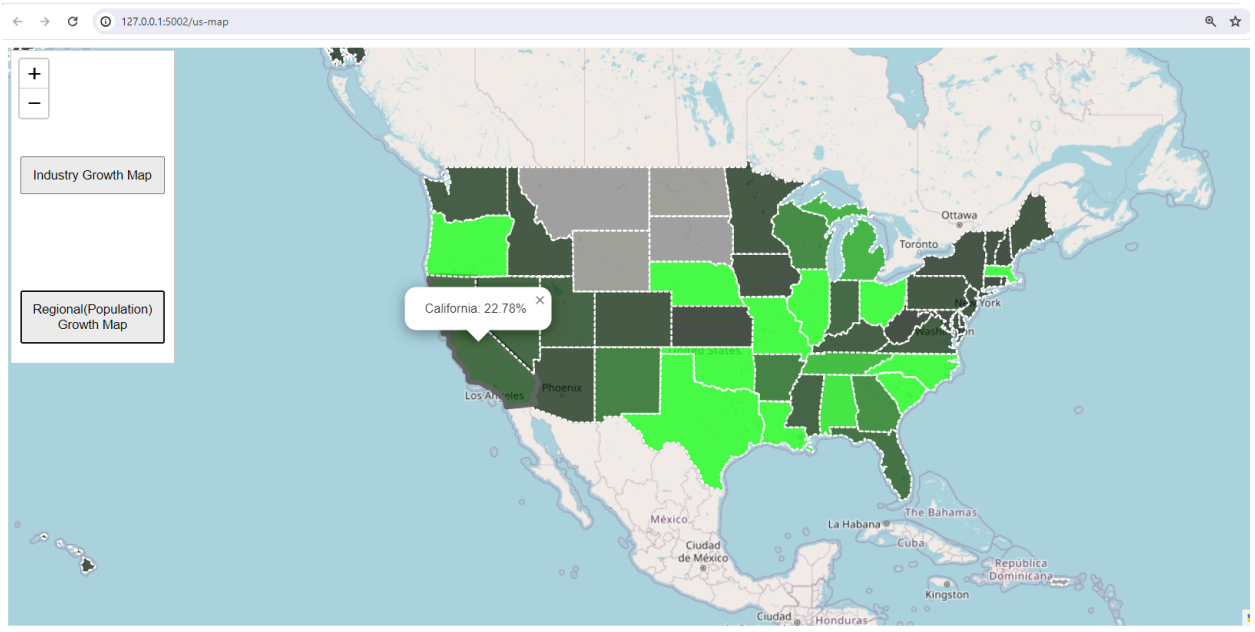
- **Routing**: Sets up a route that triggers the function when a specific URL is accessed.
- **Querying**: Connects to our MySQL "group4_db" database, then we construct and execute a query based on parameters selected by the user, and fetch the results found in that location selected by the user within the dataset in the database.
- **Error Handling**: Catches and handles errors/exceptions, as seen in the last line of code above.

ML Frontend(Gurman and Harjyot):

For the frontend implementation of the Machine Learning (ML) results, we wrote the HTML and JavaScript code to seamlessly integrate the ML output with the interactive map. The ML data is visually represented using color coordination to differentiate various data points which can be

seen below for the growth and decline, making it easier for users to interpret the results. This interface provides an intuitive and engaging way for users to explore the insights generated by the ML algorithms. In the first picture you can see the industry growth projects overlaid on the map where lighter green is highest and darker green to red is decreasing to negative projected growth. The second picture shows projected population growth in the future, it follows a similar green to red color guide, where lighter green states are and will experience more population growth and the darker ones will see decline.





ML Backend(Gurman and Harjyot):

We developed the backend for the ML component, which involved setting up two key routes in Flask: one for retrieving the ML growth data and another for handling the map's data integration. The ML backend processes the data and ensures it is accurately fetched from the MySQL database, then it formats the data appropriately for the frontend display. By implementing efficient querying and routing mechanisms, we ensured smooth data flow between the backend

and the interactive map, providing a cohesive user experience.

```python
def state_industry_growth_data():
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor(dictionary=True)

    try:
        cursor.execute("""
            SELECT SUBSTRING_INDEX(region, ',', -1) AS state, AVG(predicted_industry_potential) AS a
            FROM industry_potential_by_region
            GROUP BY SUBSTRING_INDEX(region, ',', -1)
        """)
        rows = cursor.fetchall()
        # Normalize state names if necessary, and round averages for simplicity
        growth_data = {row['state'].strip(): round(row['avg_growth'], 2) for row in rows}
        return jsonify(growth_data)  # Ensure the variable name matches what's being returned
    except Exception as e:
        # Ensure the error message is helpful and the variable names are correct
        return jsonify({'error': 'Database query failed', 'details': str(e)}), 500
    finally:
        cursor.close()
        conn.close()

@app.route('/state-regional-growth-data')
def state_regional_growth_data():
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor(dictionary=True)

    try:
        cursor.execute("""
            SELECT SUBSTRING_INDEX(region, ',', -1) AS state, AVG(predicted_regional_growth) AS avg_
            FROM regional_growth_by_region
            GROUP BY SUBSTRING_INDEX(region, ',', -1)
        """)
        rows = cursor.fetchall()
        growth_data = {row['state'].strip(): round(row['avg_growth'], 4) for row in rows}
        return jsonify(growth_data)
    except Exception as e:
        return jsonify({'error': 'Database query failed', 'details': str(e)}), 500
    finally:
        cursor.close()
        conn.close()
```