November 11, 2025

SiteScan

Part A: Letter of Transmittal

November 11, 2025
Board of Directors
BuySum
1100 W Ave, New York, 00000

Dear Board Committee members,

This letter of transmittal pertains to a proposal for expanding business operations. BuySum has expressed a desire to expand store operations into new territories, and expansion has been delayed or halted due to uncertainty and ambivalence regarding where to branch out. Factors contributing to this have been: overhead restrictions, limitations of demographic and economic datasets, and a lack of methodology and expertise regarding financial forecasting. Engineering teams, marketing teams, and CFOs have gathered and met to discuss next-step strategies. Deliberations have pointed to a software solution that utilizes a machine learning foundation. As a tenured Machine Learning engineer, with backgrounds in data engineering, data analytics, and software development, I was tasked with coordinating with my team to develop a solution strategy.

The solution proposed is a web application, named *'SiteScan'*, whose primary function is to automate data research and construct computing models that can reduce large local economic and demographic datasets into a digestible report that provides a general idea of how well a new store could potentially perform in a given location through the use of machine learning models. This proposal would benefit BuySum by providing an automated tool capable of generating forecasts in seconds, which could reduce the time required for data analysis and research by marketing and financing teams. The solution is also extensible and scalable; for example, while the datasets currently used in our forecasting model are regional, they can be extended nationally if the company ever decides to expand outside of the state.

The data collected to construct this web application is divided into five fields of demographic and economic regional data: population, mean income, mean home value, mean commute time, and poverty population, which cover the trends of the last 12 years. The datasets comprise over 200 records, which will be subdivided and grouped by zip code, with each group containing approximately 12 records that map to 12 years of trend history.

The hypothesis of this project is that store profit performance in a future site can be inferred and projected through data analysis of the site's economic and demographic trends. The objective of the project is to construct a computational model that utilizes machine learning to study, interpret, and project hypothetical performance trends for a new store, ensuring a mathematically and logically defensible outcome.

The development methodology of this project is in 9 phases:
1. Background research for machine learning model and algorithm selection
2. Data collection from publicly available repositories.
3. Environment set up for project scaffolding.
4. Data modeling for dataset scaffolding.
5. Data munging to transform the data into a usable format.
6. Training the unsupervised machine learning model on the dataset.
7. Training the supervised machine learning model on the dataset.
8. Integrating the two machine learning models.
9. Building a frontend for a user interface.
10. Wiring the machine learning models to the frontend.

It’s estimated that the project should have a production-ready application within 30 days of development time, so a budget list modeled around that timeframe has been provided:

one-time expenses:

items	**Number of units**	**Cost/unit (in USD)**	**Cost sum (in USD, monthly)**
employee workstations	6	1000	6000

recurring, monthly expenses:

items	**Number of units**	**Cost/unit (in USD)**	**Cost sum (in USD, monthly)**
data engineer	2	7000	14000
AI engineer	2	9200	18400
IT staff	2	6200	12400
office space	n/a	2000	2000
total	n/a	n/a	46800

total expenses = (one time items + monthly expenses) = $47,400

With an initial overhead of ~$50,000. Stakeholders should see an ROI that covers that amount within 30 days if we factor in the man-hours saved from data analysis activities, with company-wide financial windfalls around a 5% increase per month. Furthermore, the web application will also bypass any financial obligations tied to legal responsibilities or permits because all Data Commons information is public-facing and devoid of any PII.

More details of the inner workings and logic foundation of the product will be provided in the business documents. We’re excited for your review and collaboration.

Sincerely,

Lawrence Sanzogni, Machine Learning Engineer

Part B: Project Proposal Plan

Project Summary

This executive summary is a recap and technical elaboration of the machine learning solution project presented in the letter of transmittal. To reiterate:

The client, *BuySum*, seeks to expand store operations into new territories, and uncertainty and ambivalence regarding where to expand have caused business delays or halts. They've reported that overhead restrictions, limitations of demographic and economic datasets, and a lack of methodology and expertise regarding financial forecasting have been large contributors to this stagnancy. They've expressed a wish to decide on two new locations and secure the contracting rights of those locations within the next 90 days.

The solution proposed to aid in their decision-making is a web application named *'SiteScan'*, whose primary function is to automate data research and construct computing models that can reduce large local economic and demographic datasets into a digestible report that provides a general idea of how well a new store could potentially perform in a given location through the use of machine learning models. This proposal would benefit our client by providing an automated tool capable of generating forecasts in seconds, thereby reducing the days of data analysis and research required by marketing and financing teams.

The application's core functionality will be built on a machine learning model and will require test data for training. Datasets will be procured from the Data Commons open source data repository, and the datasets will consist of the following five demographic and economic categories, as table fields: mean income, mean home value, mean commute time, and poverty population; each category will be mapped to 12 years of historical data (presented as table rows). The dataset tables containing these categories will be subdivided into zip code groups, resulting in a zip code data frame with 12 records and 5 fields.

The methodology behind this project's design and development was covered in the letter of transmittal, but to reiterate, the development of this project is divided into 9 phases:

1. Background research for machine learning model and algorithm selection: Research data and test code snippets will be kept in a Google Colab document. This helps incrementally build the data modeling components for the application before setting up the app environment.
2. Data collection from publicly available repositories: The datasets mentioned previously will be fetched from the DataCommons repository using DataCommons's API methods.
3. Environment setup for project scaffolding: After testing the data collection and modeling code modules, setting up the project environment is the next step. A Django/Python app tech stack will be chosen for implementation.
4. Data modeling for dataset scaffolding: In the project environment, classes and modules will be added to save the datacommons API payload into local CSV files.
5. Data munging to transform the data into a usable format: Classes and modules will be added to discretize and normalize the data categories from the raw data CSV files.
6. Training the unsupervised machine learning model on the dataset. The normalized and discretized datasets will be fed into the unsupervised ML, and the model will be trained to look for associations in the datasets.
7. Training the supervised machine learning model on the dataset. The supervised ML will be fed the normalized datasets and perform linear regression computations. These regression lines will then be plotted on a *matplotlib* graph.
8. Integrating the two machine learning models: Once the two models have been verified to work independently, the unsupervised model's findings will be fed into the supervised model. This ULM's findings will act as Mathematical weights that represent the level of influence that each category has on the regression line.

9. Building a frontend for the user interface: A simple yet comprehensive dashboard will be built using the Django templating engine. This dashboard will contain and support all required data visualizations.

10. Wiring the machine learning models to the frontend: Once both MLs have completed their training. The regression line plot will be converted to an HTML string, saved in a buffer file, and then served to the webpage, where it will be rendered.

The deliverables associated with the design and implementation of this application reflect it's data pipeline and are the following:

1. Set up a running Django, Python execution environment, using Python version 3.1.14. Extenrnal Python libraries required to execute the project are *matplotlib, pandas, scipy mlxtend.*

2. Build a *Database* class module with the following methods:
    - Can successfully fetch and return all required datasets from the Datacommons API
    - Write the fetched datasets into CSV files and read the CSV files.
    - Convert the datasets into visual plots.

3. Build a *Pandas* class module that leverages pandas library methods for efficient data modeling and data munging. The methods of this class should:
    - Normalize the datasets by converting all trends into percentages.
    - Discretize the datasets into 'true' or 'false' values based on their behavior (e.g., increasing, decreasing, flat).
    - Group the datasets by zipcode to prevent data bleeding.
    - Save the normalized and discretized datasets into their own, separate CSV files.

4. Build an *Unsupervised* class module that leverages the *mlxtend* Python library methods for association training. The methods of this class should:
    - Read the discretized dataset CSV files.
    - Bin the discretized datasets into itemsets.
    - Perform apriori algorithmic computations on the discretized data sets to calculate the 'support', 'confidence', and 'lift' of the items in the itemsets.

5. Develop a Supervised Learning class module that utilizes the SciPy library for regression training. The methods of this class should:

    - Extract a group from the normalized CSV file.
    - Compute the regression line and get the slope for each data category in the group.
    - Compute the values of the projected *store profits* for each year by applying the following formula:

$$store\_profits_i = population_i * populationSlope + mean\_income_i * mean\_incomeSlope + mean\_home\_value_i * mean\_home\_valueSlope + commute\_time_i * commute\_timeSlope + poverty\_population_i * poverty\_populationSlope$$

    - Compute the regression line of the new store profit values.

6. Build a *view* to wire the data retrieved from the Supervised model and inject it into the Django HTML template.

The implementation plan for this product will be divided into phases:

- A prototyping phase that runs computations on regional data (the selected location is for the dataset is Austin, TX). The anticipated outcome is a meta-linear regression that accurately parallels the trends and relationships of the five data categories.
- A statewide phase that extends the dataset and trains the MLs on city-level associations. The anticipated outcomes mirror the previous phase but on a larger scale.
- A national phase that extends the dataset further to train the MLs on county-level associations. The anticipated outcomes mirror the previous phase but on a larger scale.
- A dry test for stakeholders to determine if financial forecast evaluation metrics are sufficient. An optimistic anticipated outcome is that the prototype meets expectations,
- An implementation phase that hinges on board approval.

Validation of the product meeting requirements will be through the agile development SDLC model. Once v1.0.0 has been tested through all phases listed in the implementation plan, the product will undergo a dry run, allowing stakeholders to gain a firsthand look at its functionality. Feedback will be collected iteratively, and updates will be applied cyclically as needed.

Timelines for each deliverable are illustrated in the table below:

Milestone or deliverable	**Project Dependencies**	**Resources**	**Start and End Date (weekends factored in)**	**Duration**
Fetch modules that collect all required data	Data Commons API methods for retrieving data, A Data Commons API key for permission to retrieve data, and an active Google Colab document that runs a Python environment for testing data requests.	The Data Commons API documentation, Google Colab documentation, Macbook Pro M1	12/01/2025 - 12/05/2025	5 days
Project Environment set up	An IDE environment, Python, and pip locally installed, the Django	The PyCharm IDE, Mac OS version 15.6.1, Python version 3.1.14 installed locally, the pip Python package manager	12/08/2025 - 12/10/2025	2 days
Modeled Data	Scripts that can save the Data Commons data payload locally, and modules that can provide schemas for formatting the data	All previous resources + the *pandas* Python library, which also has built-in CSV read and write methods	12/11/2025 - 12/15/2025	3 days

Munged Data	Scripts and modules that can normalize and discretize the data in the CSV files for machine model training	All previous resources + the pandas Python library	12/16/2025-12/19/2025	4 days
An Unsupervised Machine learning model that has completed association training	Libraries, scripts, and modules that can bin discretized datasets and run the apriori algorithm on the bins	All previous resources + the All previous All previous resources + the mlxtend Python library	12/22/2025 - 12/26/2025	5 days
A Supervised Machine learning model that can compute the meta regression line and generate the necessary UI plots	Libraries, scripts, and modules that can perform linear regression and draw data into graphics.	All previous resources + the matplotlib and scipy Python libraries	12/29/2025-12/31/2026	3 days
A user interface that can query the machine models in the backend, pass a zipcode as an input parameter, and render the computed graphs from the supervised learning model on a web browser.	Scripts and modules that can wire backend data to the frontend.	All previous resources + the built-in Django library	1/1/2025 - 1/3/2025	3 days

Part C: Application

All application source code files can be downloaded from the GitHub repository (https://github.com/Lawsan92/SiteScan/). They will also be included as a ZIP attachment in this submission.

Part D: Post-implementation Report

This post-implementation report will provide a detailed review of the inner workings of the finished product's components, including mathematical computations, data modeling methods, all versions of the datasets, machine learning algorithms, and frontend rendering and integration scaffolding. It will tie all of these descriptions in a development timeline. It will also review whether the finished product met its deliverables or if adjustments were needed.

Solution Summary

To recap, the business requirement for this project is to program a web application that can produce a financial forecast of how well a new store could hypothetically perform in an area, by performing a series of data computations on the following local dataset trends: population, mean income, mean home value, mean commute time, and poverty population. We'll begin with the evolution of our datasets.

Data Summary

The initial datasets were retrieved from the Data Commons open-source repositories using its public API methods that were written for a Python implementation. A sample fetch method requires the following parameters:

- Dcids: These are the “category” parameters.
- A Year list.
- A zip code.

Code fig.

```
from datacommons_client.client import DataCommonsClient

zip = '78735'

def API_fetch():

 api_key = "AIzaSyCTI4Xz-UW_G2Q2RfknhcfdAnTHq5X5XuI" # Replace with your API key
 client = DataCommonsClient(api_key=api_key)
 response = client.observation.fetch(
     variable_dcids=[
         "Count_Person",
         "Mean_Income_Household",
         'Median_HomeValue_HousingUnit_OccupiedHousingUnit_OwnerOccupied',
         'Mean_CommuteTime_Person_Years16Onwards_WorkCommute_Employed_WorkedOutsideOfHome',
         'Count_HousingUnit',
         'Count_Person_BelowPovertyLevelInThePast12Months',
         ],
     date='',
     entity_dcids=[f'zip/{zip}']
 )
  return response
```

The format from the original payload is a JSON object:

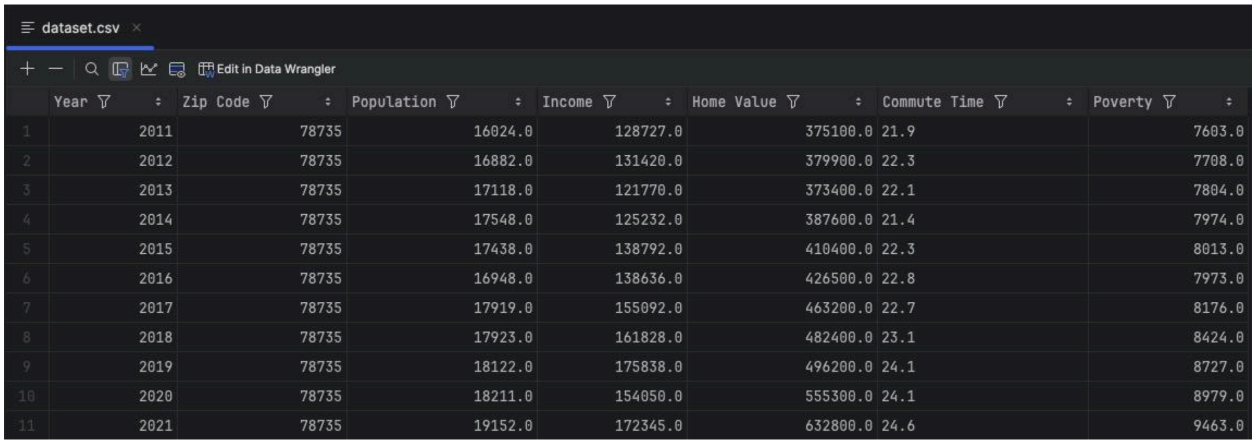
Code fig.

```
response: {
  "byVariable": {
    "Mean_Income_Household": {
      "byEntity": {
        "zip/78735": {
          "orderedFacets": [
            {
              "earliestDate": "2011",
              "facetId": "1107922769",
              "latestDate": "2023",
              "obsCount": 13,
              "observations": [
                {
                  "date": "2011",
                  "value": 128727.0
                },
               ...
                {
                  "date": "2023",
                  "value": 193891.0
                }
              ]
            }
          ]
        }
      }
    },
           "Median_HomeValue_HousingUnit_OccupiedHousingUnit_OwnerOccupied":{...},
"Mean_CommuteTime_Person_Years16Onwards_WorkCommute_Employed_WorkedOutsideOfHome": {...},
"Count_HousingUnit": { ...},
"Count_Person_BelowPovertyLevelInThePast12Months": {...},
"Count_Person": {...},
  "facets": {...}
```

As demonstrated, this format is heavily nested, with numerous attributes, and querying it would necessitate elaborate queries.

The custom *Database* Python class contains a **model_data** method which transforms the data set into the following format that's saves it into a CSV file:

dataset.csv Fig.


dataset.csv
Edit in Data Wrangler

Year	Zip Code	Population	Income	Home Value	Commute Time	Poverty
2011	78735	16024.0	128727.0	375100.0	21.9	7603.0
2012	78735	16882.0	131420.0	379900.0	22.3	7708.0
2013	78735	17118.0	121770.0	373400.0	22.1	7804.0
2014	78735	17548.0	125232.0	387600.0	21.4	7974.0
2015	78735	17438.0	138792.0	410400.0	22.3	8013.0
2016	78735	16948.0	138636.0	426500.0	22.8	7973.0
2017	78735	17919.0	155092.0	463200.0	22.7	8176.0
2018	78735	17923.0	161828.0	482400.0	23.1	8424.0
2019	78735	18122.0	175838.0	496200.0	24.1	8727.0
2020	78735	18211.0	154050.0	555300.0	24.1	8979.0
2021	78735	19152.0	172345.0	632800.0	24.6	9463.0

model_data fig.

```python
# Database.py
def model_data(self):
   print('cleaning and modeling data...')
   data_model = {}
   data_keys = [
       "Count_Person",
       "Mean_Income_Household",
       'Median_HomeValue_HousingUnit_OccupiedHousingUnit_OwnerOccupied',
       'Mean_CommuteTime_Person_Years16Onwards_WorkCommute_Employed_WorkedOutsideOfHome',
       'Count_HousingUnit',
       'Count_Person_BelowPovertyLevelInThePast12Months',
   ]
   zip_keys = self.zip_keys

   for key in data_keys:
       for dataset in self.data.byVariable[key]:
           for zip_key in zip_keys:
               filtered_dataset = dataset[1][f'zip/{zip_key}'].orderedFacets[0].observations
               for i, entry in enumerate(filtered_dataset):

                   def missing_entry():
                       if i < len(filtered_dataset) - 1:
                           if int(filtered_dataset[i + 1].date) !=
int(filtered_dataset[i].date) + 1:
                               dummy_data = type('Observation', (object,), {
                                   'date': int(filtered_dataset[i].date) + 1,
                                   'value': 'null'
                               })
                               filtered_dataset.insert(i + 1, dummy_data)
                       return
                   missing_entry()
```

```
                    year = entry.date
                    value = entry.value
                    model_key = '[' + str(year) + '|' + str(zip_key) + ']'
                    if model_key not in data_model:
                        data_model[model_key] = []
                    data_model[model_key].append([key, value])
    self.data = data_model
    print(f'data model created...number of zip codes: {self.data_size}')
```

The next phase involved preparing the data in a format that could be used by both supervised and unsupervised learning models through normalization and discretization.

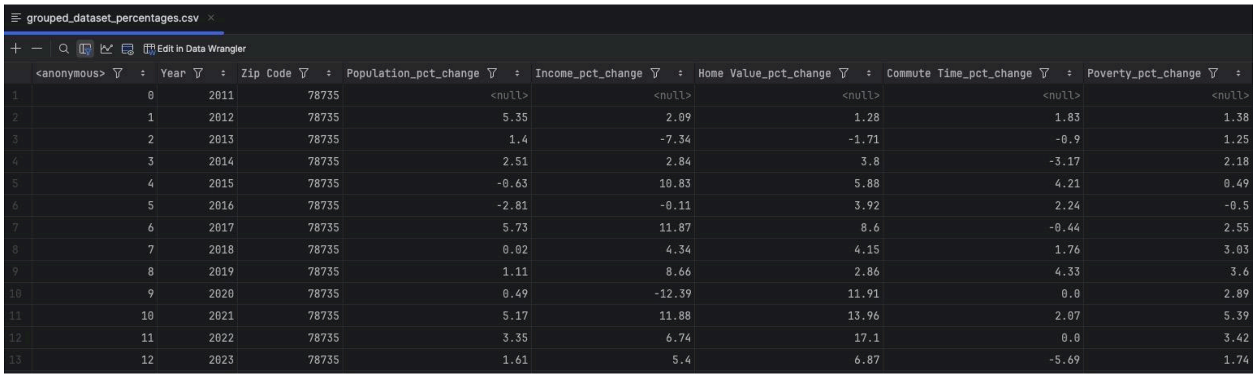
Normalizing the data was achieved by loading the CSV file into the *PandaDataframe* class and converting all trends experienced by each data category into percentage movements using the *to_percentages* class method.

to_percentages code fig.

```
# Pandas.py
def to_percentages(self):
   print('Converting numeric columns to percentage changes by Zip Code...')
   df = self.dataframe.copy()
   numeric_cols = df.select_dtypes(include='number').columns

   # Loop through all numeric columns
   for col in numeric_cols:
       # Compute pct_change() within each Zip Code group
       df[f'{col}_pct_change'] = df.groupby('Zip Code')[col].pct_change() * 100
   df = df.round(2)

   self.dataframe = df

   df = df.drop(['Population','Income', 'Home Value','Commute Time','Poverty',
'Year_pct_change', 'Zip Code_pct_change'], axis=1)
   df.to_csv(os.path.join(self.base_path,'grouped_dataset_percentages.csv'))
   df = df.drop(
       ['Year', 'Zip Code'], axis=1)
   df = df.drop([0])
   self.dataframe = df
   print('saving percentages to csv file...')
```

The normalized data was then saved into its own *dataset_percentages.csv* file.

dataset_percentages.csv fig.


grouped_dataset_percentages.csv

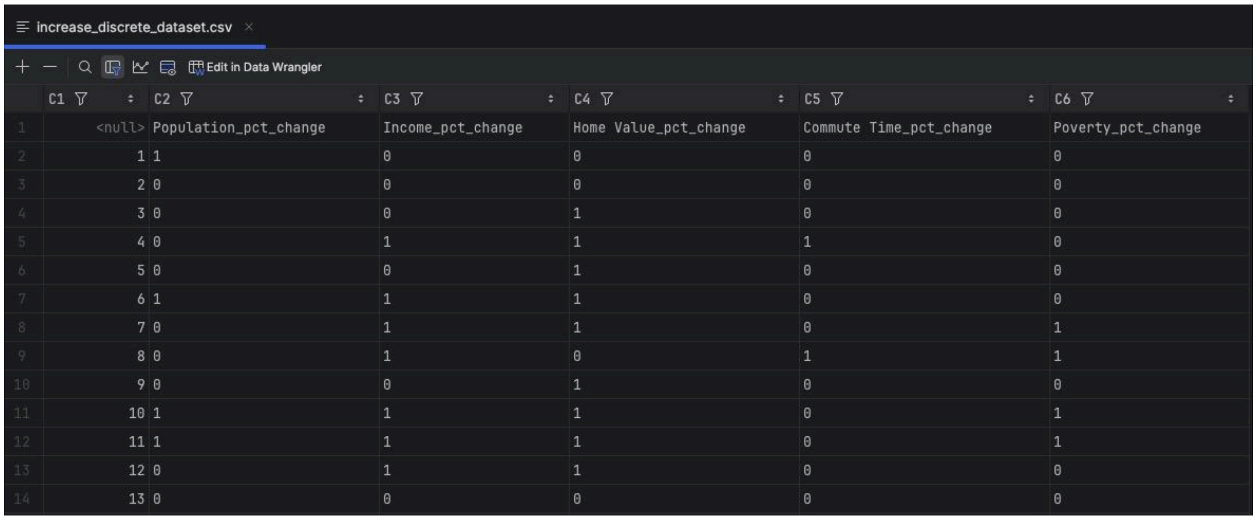
<anonymous>	Year	Zip Code	Population_pct_change	Income_pct_change	Home Value_pct_change	Commute Time_pct_change	Poverty_pct_change
0	2011	78735	<null>	<null>	<null>	<null>	<null>
1	2012	78735	5.35	2.09	1.28	1.83	1.38
2	2013	78735	1.4	-7.34	-1.71	-0.9	1.25
3	2014	78735	2.51	2.84	3.8	-3.17	2.18
4	2015	78735	-0.63	10.83	5.88	4.21	0.49
5	2016	78735	-2.81	-0.11	3.92	2.24	-0.5
6	2017	78735	5.73	11.87	8.6	-0.44	2.55
7	2018	78735	0.02	4.34	4.15	1.76	3.03
8	2019	78735	1.11	8.66	2.86	4.33	3.6
9	2020	78735	0.49	-12.39	11.91	0.0	2.89
10	2021	78735	5.17	11.88	13.96	2.07	5.39
11	2022	78735	3.35	6.74	17.1	0.0	3.42
12	2023	78735	1.61	5.4	6.87	-5.69	1.74

After normalization, the data needed to be discretized into '0' and '1' values, as the apriori algorithm required a 2D matrix of '1s' and '0s' to perform its computations. This was accomplished using the *discretize* method from the *PandaDataFrame* class. As a reminder, two discrete datasets were required for association training: one for increasing trends and another for decreasing trends.

discretize fig.

```
def discretize(self):
   print('converting continuous variables to discrete variables...')
   cont_cols = self.dataframe.columns

   def discretize_increase():
       increase_data = self.dataframe.copy()
       for i, col in enumerate(cont_cols):
           increase_data[col] = (increase_data[col] >= 3).astype(int)
       # save data to csv
       print('saving increasing discrete dataframe to csv file...')
       increase_data.to_csv(os.path.join(self.increase_path,
'increase_discrete_dataset.csv'))
       return increase_data

   self.increased_dataframe = discretize_increase()

   def discretize_decrease():
       decrease_data = self.dataframe.copy()
       for i, col in enumerate(cont_cols):
           decrease_data[col] = (decrease_data[col] <= -3).astype(int)
       print('saving decreasing discrete dataframe to csv file...')
       decrease_data.to_csv( os.path.join(self.decrease_path,
'decrease_discrete_dataset.csv'))
       return decrease_data

   self.decreased_dataframe =  discretize_decrease()
```

increase_discrete_dataset.csv



C1	C2	C3	C4	C5	C6
<null>	Population_pct_change	Income_pct_change	Home Value_pct_change	Commute Time_pct_change	Poverty_pct_change
1	1	0	0	0	0
2	0	0	0	0	0
3	0	0	1	0	0
4	0	1	1	1	0
5	0	0	1	0	0
6	1	1	1	0	0
7	0	1	1	0	1
8	0	1	0	1	1
9	0	0	1	0	0
10	1	1	1	0	1
11	1	1	1	0	1
12	0	1	1	0	0
13	0	0	0	0	0

Note: there also exists a decrease_discrete_dataset.csv file.

Every row of 1s and 0s represents a bin where 0 = no item, and 1 = an item.

Machine Learning

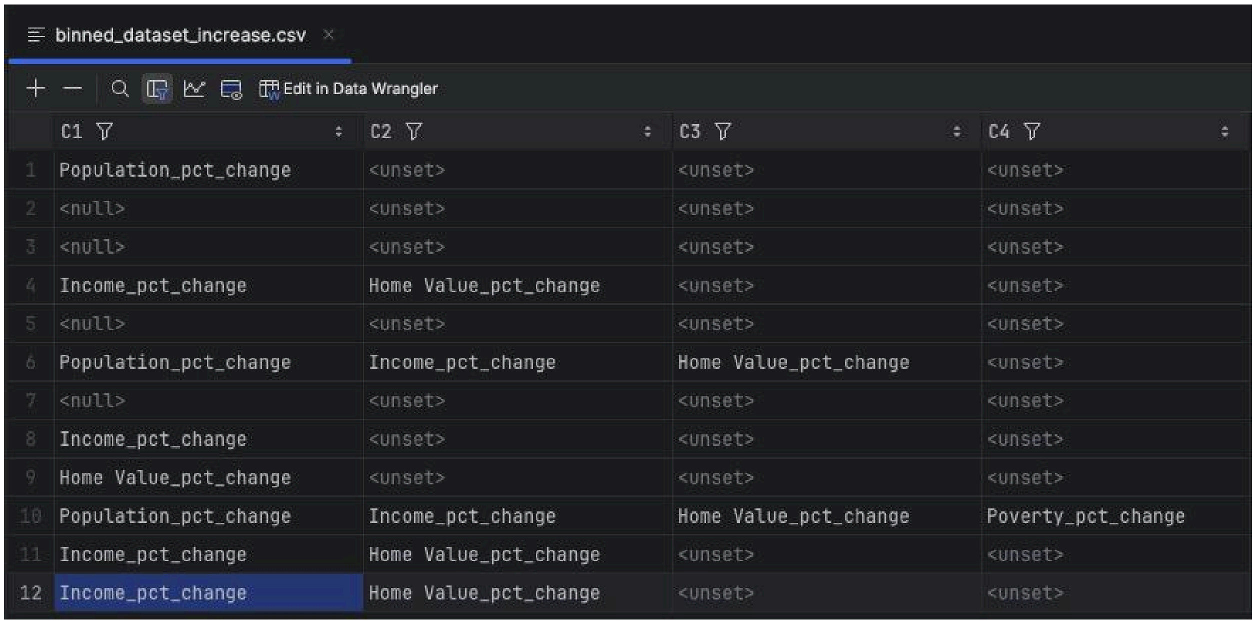
There are two main class modules responsible for the machine learning computations: a *Supervised_Model* class, which implements predictive methods, and an *Unsupervised_Model* class, implementing a descriptive method.

Unsupervised Learning Model

The unsupervised model's primary function was to identify trend associations and relationships between the five data categories by implementing the Apriori algorithm. This algorithm scans the rows of a table containing discrete items and bins them, creating itemsets. Each itemset can be seen as a 'transaction', where items are 'purchased' together. Single items appearing frequently in the total rows have a high *support* value. Items that frequently show up together in these 'bins' have a strong *confidence*, and items that frequently lead to the appearance of other items have a high *lift*.

Here is a visualization how how a matrix of 1s and 0s would translate to bins:

binned_dataset_increase.csv



	C1	C2	C3	C4
1	Population_pct_change	<unset>	<unset>	<unset>
2	<null>	<unset>	<unset>	<unset>
3	<null>	<unset>	<unset>	<unset>
4	Income_pct_change	Home Value_pct_change	<unset>	<unset>
5	<null>	<unset>	<unset>	<unset>
6	Population_pct_change	Income_pct_change	Home Value_pct_change	<unset>
7	<null>	<unset>	<unset>	<unset>
8	Income_pct_change	<unset>	<unset>	<unset>
9	Home Value_pct_change	<unset>	<unset>	<unset>
10	Population_pct_change	Income_pct_change	Home Value_pct_change	Poverty_pct_change
11	Income_pct_change	Home Value_pct_change	<unset>	<unset>
12	Income_pct_change	Home Value_pct_change	<unset>	<unset>

As stated previously, "every row of 1s and 0s represents a bin where 0 = no item, and 1 = an item."

An aprioiri algorithm needs parameters (or rules) in order to know what to look for. The rules for this study were to count values that:

- Had a trend threshold of -3|+3 %. Meaning, only trends showing a dip or spike of 3% were considered
- Had a minimum support of 0.01 percent. This is a very low frequency, and it ties in with why changes needed to be made to the machine learning model.

Below are figures for the class method, and the results from the computation:

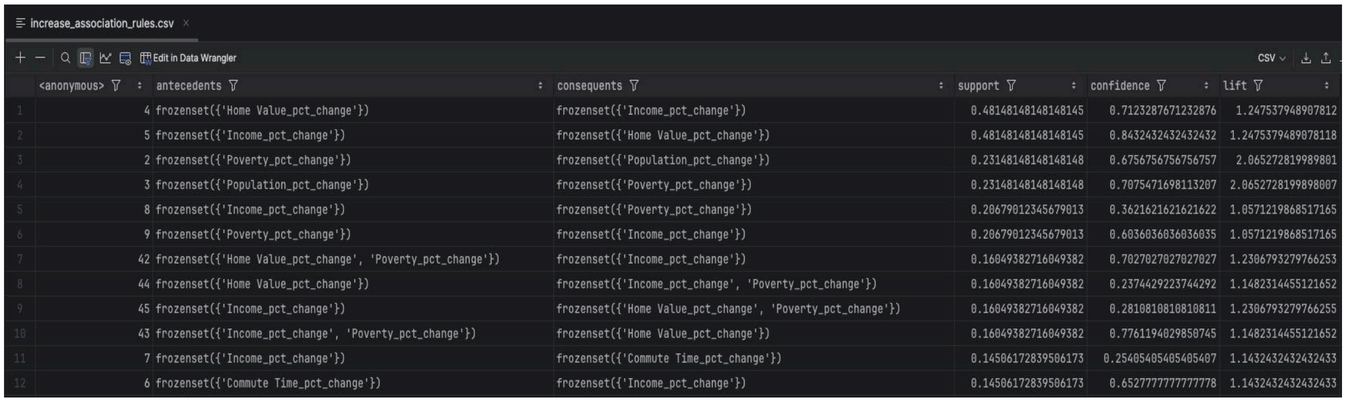
association_rules code fig.

```
#Unsupervised_Model.py
def association_rules(self):

   def increase_association_rules():
       print('generating increase association rules...')

       frequent_itemsets = apriori(
           self.increase_data,
           min_support=0.01,
           use_colnames=True
       )

       rules = association_rules(
           frequent_itemsets,
           metric="lift",
           min_threshold=1.0
       )

       rules = rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
       rules.sort_values(by='support', ascending=False, inplace=True)
       print('saving increase association rules to csv file...')
       rules.to_csv(os.path.join(self.increase_path, 'increase_association_rules.csv'))

   increase_association_rules()

   def decrease_association_rules():
     ...
   decrease_association_rules()
```

increase_association_rules.csv fig

Supervised Learning Model

This machine learning model handles predictive methods for the data, and is responsible for computing both the hypothetical data plots and the regression line of the *store_profits* dataset. These outputs are computed by using scipy's linear regression method and a custom formula that takes inspiration from the multiple linear regression formula:



	<anonymous>	antecedents	consequents	support	confidence	lift
1	4	frozenset({'Home Value_pct_change'})	frozenset({'Income_pct_change'})	0.48148148148148145	0.7123287671232876	1.247537948907812
2	5	frozenset({'Income_pct_change'})	frozenset({'Home Value_pct_change'})	0.48148148148148145	0.8432432432432432	1.2475379489078118
3	2	frozenset({'Poverty_pct_change'})	frozenset({'Population_pct_change'})	0.23148148148148148	0.6756756756756757	2.065272819989801
4	3	frozenset({'Population_pct_change'})	frozenset({'Poverty_pct_change'})	0.23148148148148148	0.7075471698113207	2.0652728199898007
5	8	frozenset({'Income_pct_change'})	frozenset({'Poverty_pct_change'})	0.20679012345679013	0.3621621621621622	1.0571219868517165
6	9	frozenset({'Poverty_pct_change'})	frozenset({'Income_pct_change'})	0.20679012345679013	0.6036036036036035	1.0571219868517165
7	42	frozenset({'Home Value_pct_change', 'Poverty_pct_change'})	frozenset({'Income_pct_change'})	0.16049382716049382	0.7027027027027027	1.2306793279766253
8	44	frozenset({'Home Value_pct_change'})	frozenset({'Income_pct_change', 'Poverty_pct_change'})	0.16049382716049382	0.2374429223744292	1.1482314455121652
9	45	frozenset({'Income_pct_change'})	frozenset({'Home Value_pct_change', 'Poverty_pct_change'})	0.16049382716049382	0.2810810810810811	1.2306793279766255
10	43	frozenset({'Income_pct_change', 'Poverty_pct_change'})	frozenset({'Home Value_pct_change'})	0.16049382716049382	0.7761194029850745	1.1482314455121652
11	7	frozenset({'Income_pct_change'})	frozenset({'Commute Time_pct_change'})	0.14506172839506173	0.25405405405405407	1.1432432432432433
12	6	frozenset({'Commute Time_pct_change'})	frozenset({'Income_pct_change'})	0.14506172839506173	0.6527777777777778	1.1432432432432433

Meta-slope formula fig.

$$store\_profits_i = population_i * populationSlope + mean\_income_i * mean\_incomeSlope + mean\_home\_value_i * mean\_home\_valueSlope + commute\_time_i * commute\_timeSlope + poverty\_population_i * poverty\_populationSlope$$

i = datapoint year

This formula requires three layers of computation:

1st Linear Regression Layer

This calculates the linear regression line for each data category to extract that category's slope. The slopes are then saved in a dictionary:

get_slopes code fig.

```python
# Supervised_Model.py
def get_slopes(self):
   data = self.data
   x = data['Year']
   y = data['Population_pct_change']

   for key, dataset in data.items():
       if key == 'Year' or key == 'Zip Code':
           continue
       y = dataset
       slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
```

```
        self.slopes[key] = slope
    print('fitting linear regression model...')

#output...
slopes: {'Population_pct_change': np.float64(0.023356643356643367), 'Income_pct_change':
np.float64(0.4513636363636364), 'Home Value_pct_change': np.float64(1.1343356643356644),
'Commute Time_pct_change': np.float64(-0.17664335664335667), 'Poverty_pct_change':
np.float64(0.24734265734265737)}
```

Theoretical data points computation layer

The calculated slopes fill the roles of the 'weights' in the meta-slope formula. This formula is then iteratively called to produce the new stope profits data points:

find_y code fig.

```
def find_y(self):
    print('finding y values...')
    slopes = self.slopes
    y = []
    for key, dataset in self.data.items():
        if key == 'Year' or key == 'Zip Code':
            continue
        for i, item in enumerate(dataset):
            try:
                y[i] = y[i] + item * slopes[key]
            except IndexError as e:
                y.append(0)
    self.y = y

#output...
model.y:  [np.float64(2.4133751748251746), np.float64(-4.784565734265735),
np.float64(6.691514685314686), np.float64(10.935691258741262),
np.float64(3.877593356643357), np.float64(15.82141993006993), np.float64(7.104967132867133),
np.float64(7.278576923076923), np.float64(8.632362587412588),
np.float64(22.165051048951053), np.float64(23.28524265734266),
np.float64(11.665726573426575)]
```

2nd linear regression and plotting layer

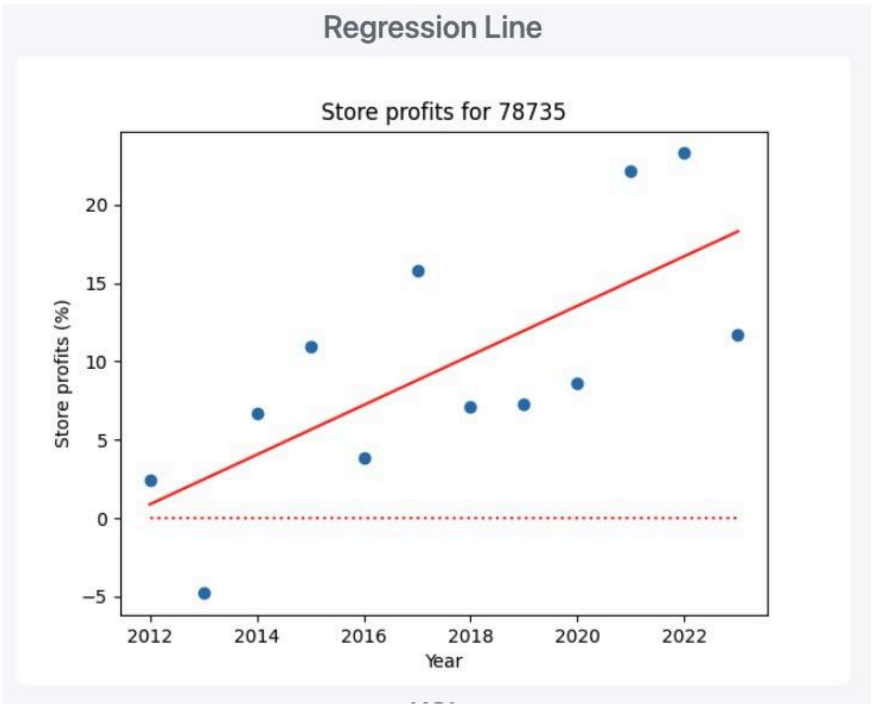
The *plot_linear_regression* method from the *Supervised_Model* class then takes the new values and runs through the same *scipy* linear regression formula. The datapoints and regression line are then plotted to a *matplotlib* line figure. The figure is then saved to a buffer file as a PNG, then converted to an HTML string.

plot_linear_regression code fig.

```
def plot_linear_regression(self):
    print('plotting linear regression...')
    data = self.data
```

```
x = data['Year']
y = self.y
base_value = [i * 0 for i in range(len(x))]

slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
print('plotting linear regression model...')
plt.scatter(x, y)
plt.title(f'Store profits for {self.zip}')
plt.xlabel('Year')
plt.ylabel('Store profits (%)')
plt.plot(x, intercept + slope * x, 'r')
plt.plot(x, base_value, ':r')

buffer_file = BytesIO()
plt.savefig( buffer_file, format='png')
graph_buffer_file_base64 = base64.b64encode(buffer_file.getvalue())
graph_buffer_file_html = graph_buffer_file_base64.decode('utf-8')
graph_html = '<img src=\'data:image/png;base64,{}\'>'.format(graph_buffer_file_html)
plt.close()
return graph_html
```

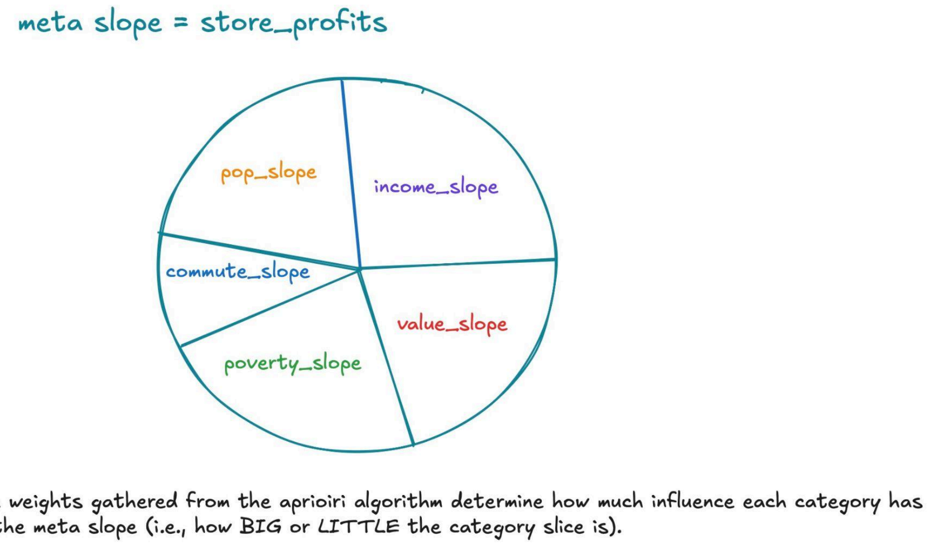
regression line graph fig.



Validation

The methods for validating the hypothesis of computing a defensible regression line for a hypothetical dataset were altered during the undertaking of this project. The original model involved assigning weights to each data category. These weights would determine the level of contribution that each category had to the composite *store profit* dataset's slope.

research graph fig.


meta slope = store_profits
pop_slope
income_slope
commute_slope
value_slope
poverty_slope
The weights gathered from the aprioiri algorithm determine how much influence each category has on the meta slope (i.e., how BIG or LITTLE the category slice is).

These weights would've been calculated by using the support, lift, and confidence values from each data category as operands. The exact distribution of weights (i.e, support vs. confidence vs. lift) would involve assuming what is more relevant, but the core model had a scaling system where the most difficult to prove metric would have the most weight:

- Support = x1.0
- Confidence = x1.2
- Lift = x1.8
- Total weight = support x confidence x lift

E.g.

The *increase_association_rules.csv fig* demonstrated that increasing *home values* have a .048 score, a .71 score, and a 1.21 score. So the result would be: 0.48 * .71 * 1.21

These weights would then be injected in the *meta-slope formula* as additional variables operands, resulting in a different formula:

meta-slope formula ver.1 fig

```
meta-slope = population_weight( population_i * population_slope) + meanIncome_weight( meanIncome_i * meanIncome_slope) + ...
```

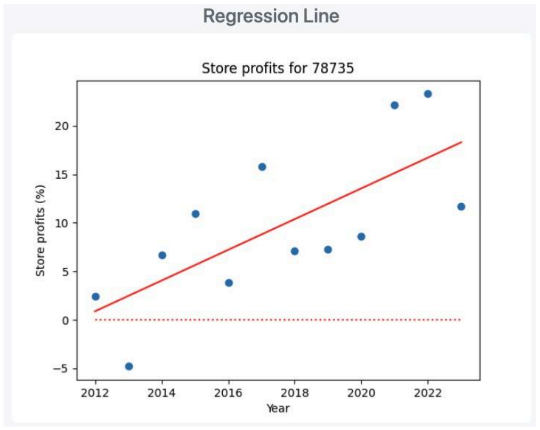
This formula needed to be changed because association training was determined to be unsuitable for this dataset and problem type for the following reasons:

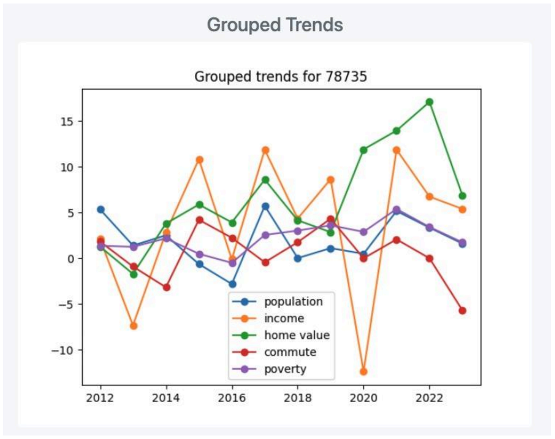
1. The datasets are too small. Although our dataset comprised 25 zip codes, totaling 325 rows of data, each zip code's dataset needed to be treated as a separate entity, and the apriori algorithm had to run on each group, rather than all zip code datasets as one merged entity. Doing so would've polluted the results, similar to merging association findings from multiple supermarket store locations, which would defeat the purpose. Each supermarket has its own unique clientele, just as each zip code has its own unique demographic and economic trends.
2. The trends aren't volatile enough or don't show any contrast. Reviewing the 25 zip code datasets, it was observed that the trends within each category followed a similar direction to those of every other. For example, an increase in population would be accompanied by an increase in poverty, which would also be associated with an increase in home values, and so on. This demonstrates that the categories are scaling, indicating a plateaued economy that is neither growing nor declining, rendering the task of determining the weight of each category's contribution impossible.

This discovery led to dropping the apriori altogether, and creating the current meta-slope equation, which treats the level of contribution from each category uniformly, a justification that is particularly applicable in an economy neither in decline nor in growth.

The accuracy of the web application's current readings was determined in two layers:

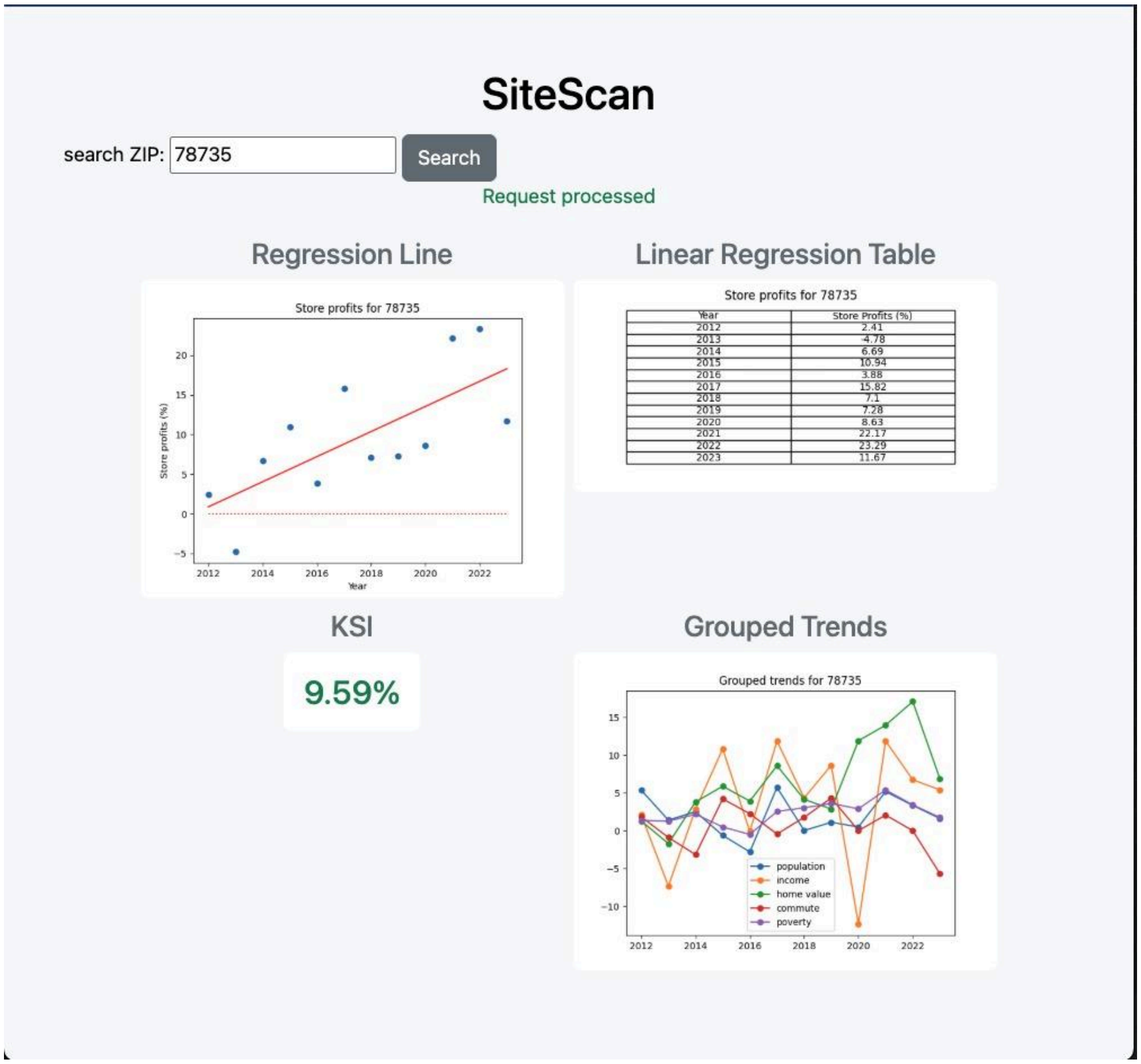
1. Using the linear regression lines as the basis for calculating the meta-slope. Because regression lines, by design, are intended to determine the most likely and accurate slope for a dataset, this seemed to be the safest math-based foundation on which to base the accuracy of our predictions. *Scipy*'s linear regression method applies Ordinary Least Squares (OLS) regression by default, minimizing the SSE (sum of squared error) between all datapoints and the regression line to ensure accuracy in forecasting.
2. Visual analysis, the dashboard has a 'grouped trends' plot showing the trends from all five categories, as well as the regression plot for the store profits. It was determined that the plots in the store profits graph accurately mirror the trends from the grouped datasets in all tested zip code datasets. Below is an example using the 78735 zip code group:





Visualizations

The final product results in these findings being wired to the Django templates and served up in a web browser dashboard:



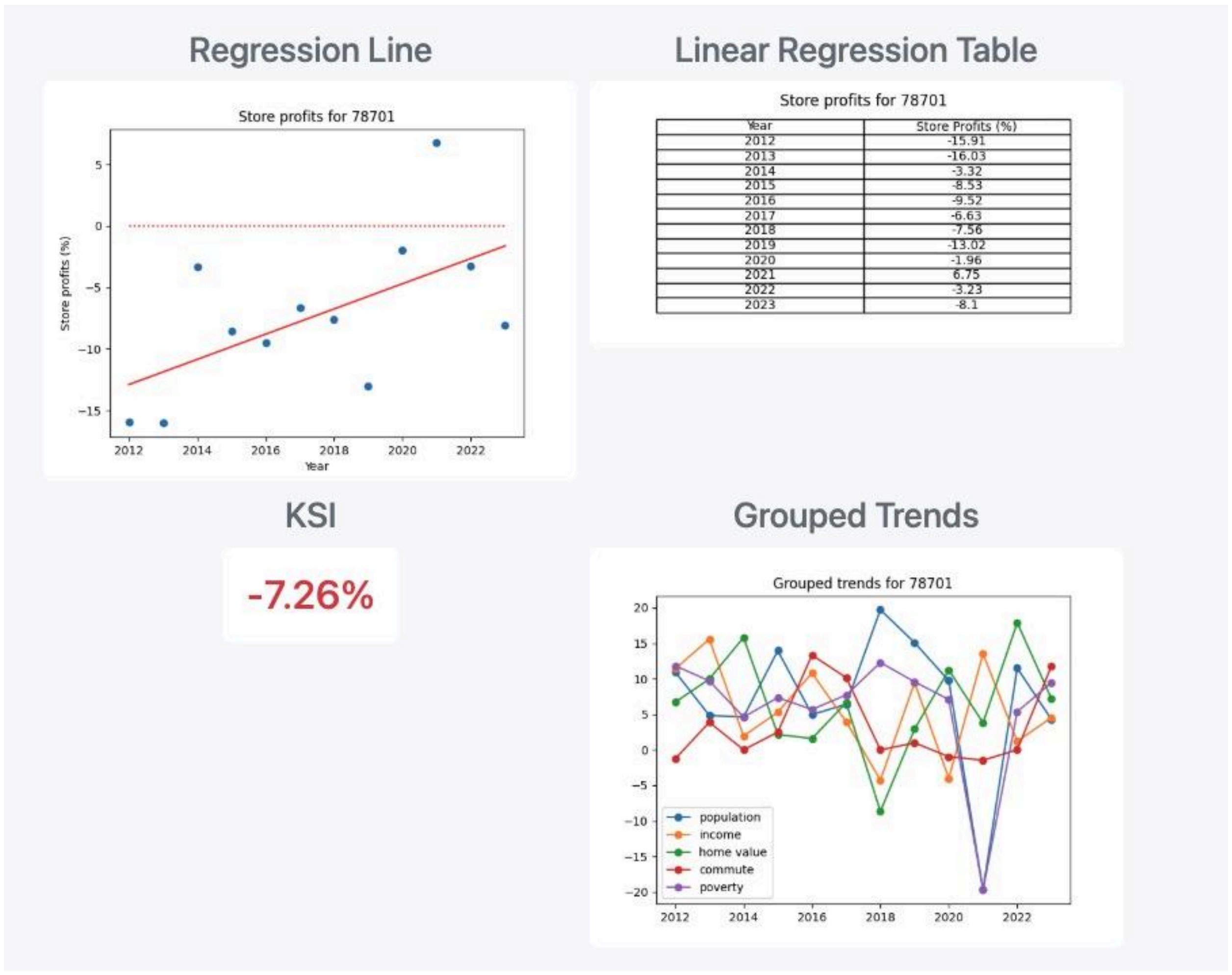
Store profits for 78735

Year	Store Profits (%)
2012	2.41
2013	-4.78
2014	6.69
2015	10.94
2016	3.88
2017	15.82
2018	7.1
2019	7.28
2020	8.63
2021	22.17
2022	23.29
2023	11.67

The dashboard contains four visualizations: The store profits plot with the regression line, the grouped trends line plots, a table with the individual store profits values from the graph, a KSI (key performance indicator), which is the mean of the datapoints from the regression table and plot. This is for giving investors an idea of 'average yearly performance' in store profits.

The results from product testing show consistent mirroring between the store regression line and the grouped trend lines, indicating that the forecasted financial plot is accurately following the trend movements of its derived datasets:

78701 visualizations fig.



Regression Line

Linear Regression Table

Store profits for 78701

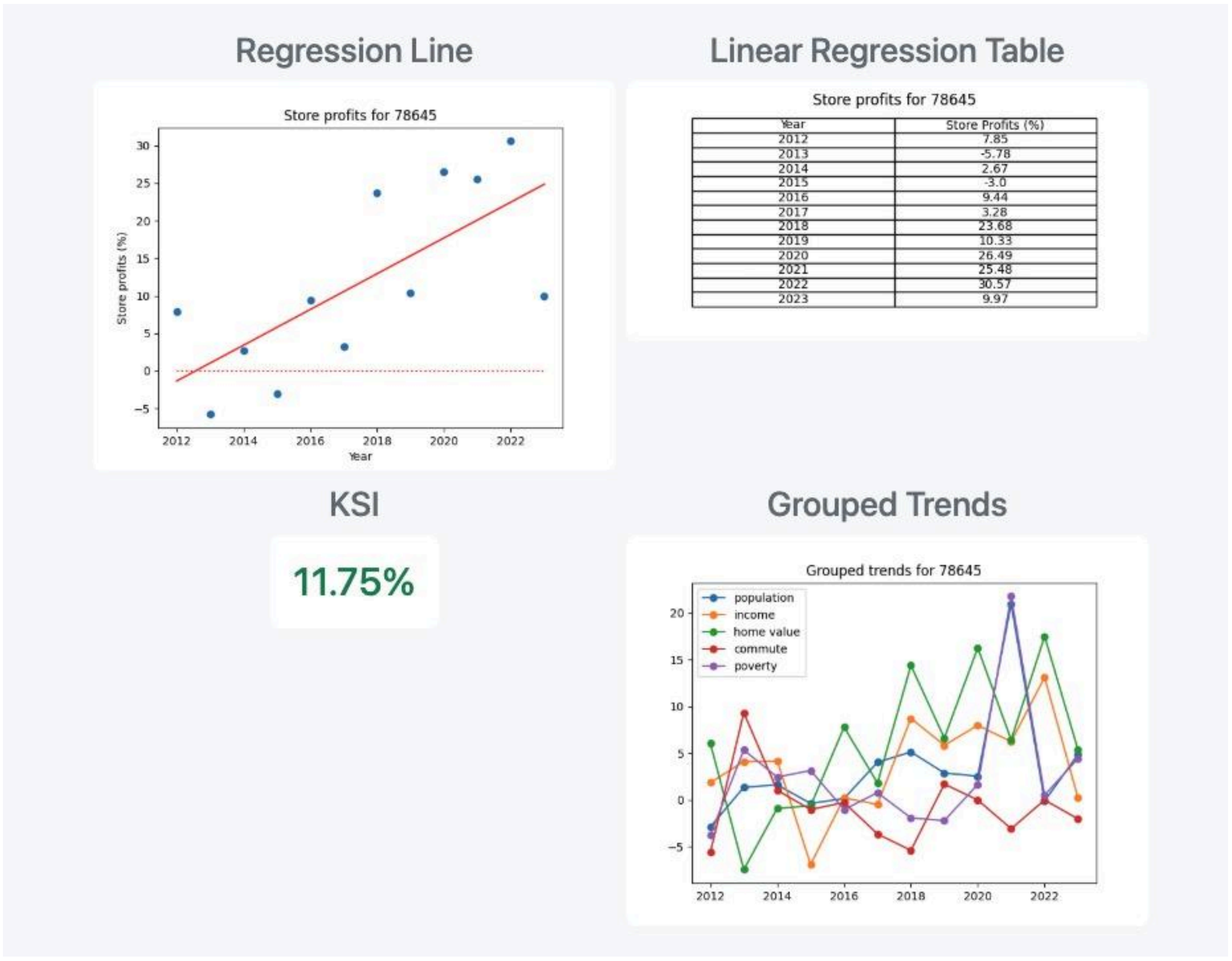
Year	Store Profits (%)
2012	-15.91
2013	-16.03
2014	-3.32
2015	-8.53
2016	-9.52
2017	-6.63
2018	-7.56
2019	-13.02
2020	-1.96
2021	6.75
2022	-3.23
2023	-8.1

KSI

-7.26%

Grouped Trends

78645 visualizations fig.



Regression Line

Linear Regression Table

Store profits for 78645

Year	Store Profits (%)
2012	7.85
2013	-5.78
2014	2.67
2015	-3.0
2016	9.44
2017	3.28
2018	23.68
2019	10.33
2020	26.49
2021	25.48
2022	30.57
2023	9.97

KSI

11.75%

Grouped Trends

The visualizations demonstrate that dips and peaks are accurately projected.

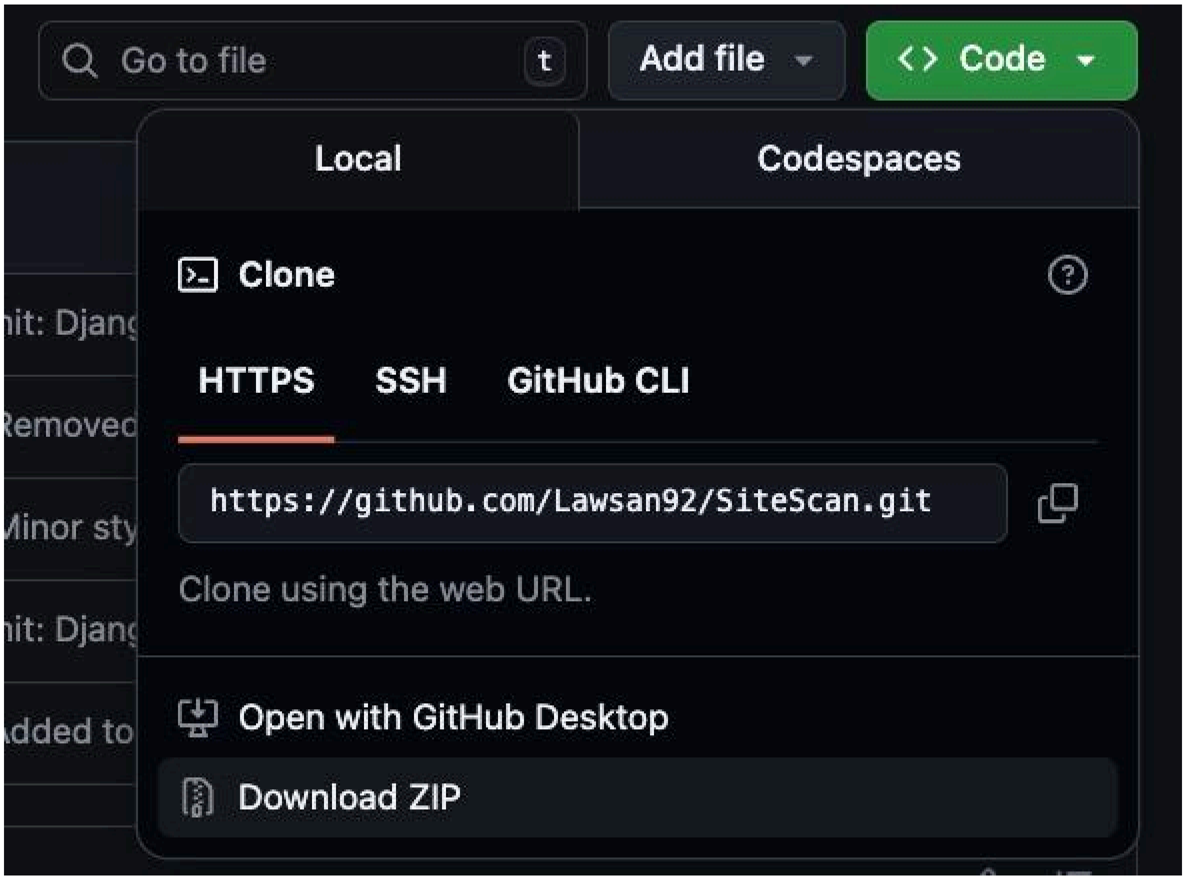
These validation methods are prototypical. Future enhancements, additions, and extensions are anticipated, given a successful product launch.

User Guide

The final is hosted in a public GitHub repository and includes a README file:
https://github.com/Lawsan92/SiteScan/

If the instructions are unclear, then please follow these steps (for Windows-based OS users):

1. Make sure that you have Python locally installed: https://www.python.org/downloads/
2. Download the zip file to your local computer, unzip it and drag it to your Desktop folder.



```
https://github.com/Lawsan92/SiteScan.git
```

Clone using the web URL.

3. Open the Windows *Command Prompt* app by hitting the *Windows Key + R* and entering *cmd* in the 'Run' dialog
4. Navigate to the project's root directory:

```
cd Desktop/SiteScan-master/
```

5. Create a Python virtual environment file & filepath (command syntax may vary based on your computer's OS and Python version):

```
py -m venv venv
```

6. Activate the virtual environment script

```
.\venv\Scripts\activate
```

7. Install the following libraries

```
py -m pip install Django matplotlib datacommons_client scipy pandas mlxtend
```

8. Configure matplotlib to run in a headless state to prevent app crashes

```
set MPLBACKEND=Agg
```

9. Run the Django server

```
py manage.py runserver

server logs should appear in your shell:

November 17, 2025 - 21:48:54
Django version 6.0b1, using settings 'SiteScan.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

10. Copy/Paste your localhost's url address into your browser: ***127.0.0.1:800***