GODADDY MICROBUSINESS DENSITY FORECASTING

Problem Statement:
Microbusiness density refers to the concentration or number of microbusinesses in a particular geographic area. A microbusiness is typically defined as a small business with fewer than 10 employees, and it may include sole proprietorships, partnerships, or small corporations. Microbusinesses are a critical component of the economy, but they often lack the resources and expertise needed to make informed decisions about expansion and investment. By providing accurate density forecasts, GoDaddy could empower small business owners to make data-driven decisions about growth and expansion, ultimately supporting the health of the microbusiness sector as a whole. In addition to its potential business benefits, the development of a microbusiness density forecasting tool can also have broader social and economic implications. Microbusinesses are an important driver of job creation and economic growth, particularly in underserved communities.

Impact:
The concentration of micro-businesses in a region is an important metric to assess its economic condition. It provides valuable insights into the number of small businesses operating in the area, which can aid policymakers and entrepreneurs in making informed decisions. Micro-businesses are essential for local economies as they create employment and stimulate economic growth. Therefore, knowing the density of these businesses in a region is vital to support and expand small businesses. Additionally, the density of micro-businesses can also indicate other significant trends such as population growth, urbanization, and migration patterns. For instance, a high concentration of micro-businesses may signify a thriving local economy and a growing population. This information can be useful in attracting new investments and businesses to the region and supporting existing businesses' growth and success.

Dataset Description:
The Godaddy microbusiness density estimation dataset is a collection of data related to the location and density of small businesses in the United States. The dataset includes information on the geographic location of businesses, their industry classification, and various attributes such as their website presence and social media activity collected from a variety of sources. Apart from this, there is another dataset which includes data from the Census Bureau’s American Community(ACS). The percentage fields in the dataset were derived from the raw counts provided by the ACS. All fields have a two year lag to match what information was available at the time a given microbusiness data update was published. The GoDaddy microbusiness data contains 122k rows and 20+ columns.

Data Preprocessing:

As a part of our exploratory data analysis, we initially plotted the correlation between dates and microbusiness density per state. Below (Fig. (1)) is the plot which shows the same. Also, we have plotted number of counties per state(this is shown in Fig. (2))

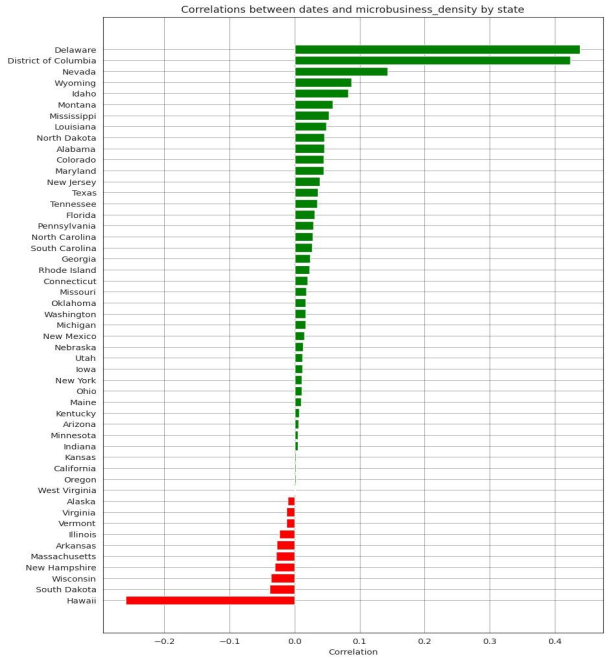

Fig. (1)

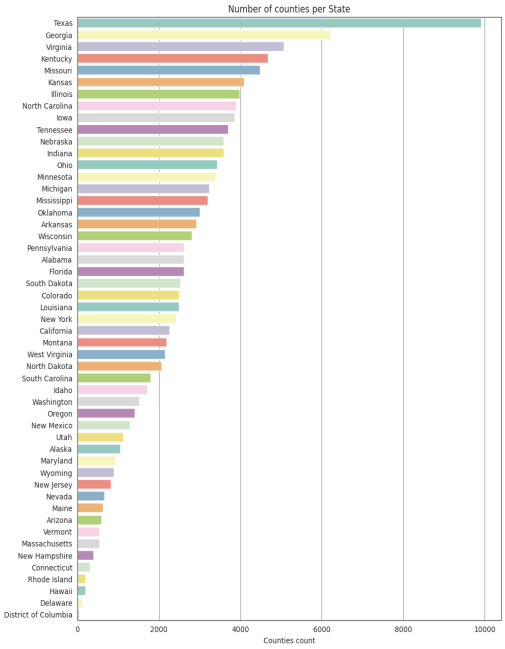

Fig. (2)

From Fig(1) we can observe that Delaware, District of Columbia are the states with the highest and Hawaii and South Dakota have the lowest microbusiness density. We then plotted microbusinesses over time across different states to see the trend. Fig(3) shows the microbusiness density over time.

Feature Engineering:

Some of the feature engineering steps we have performed are listed below:

1) Performed operations such as lag features, rolling statistics, exponential smoothing.
2) Converted and formatted the date (month, day, year) and performed scaling of features which had large values.
3) We then categorized the string features - Performed One hot encoding on categorical features.
4) Grouped the data using county, State code to better understand the business density in specific areas.

Fig(4) below is the plot for visually inspecting the trend in microbusiness density and as we can see there is a linear trend which had to be removed before training time-series models.

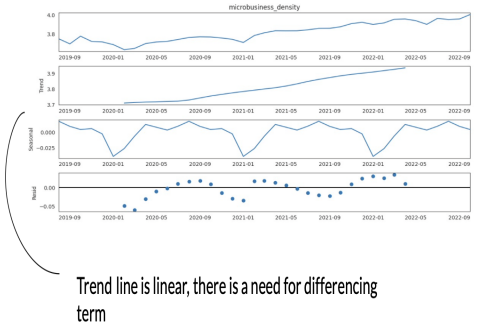

Fig. (4)

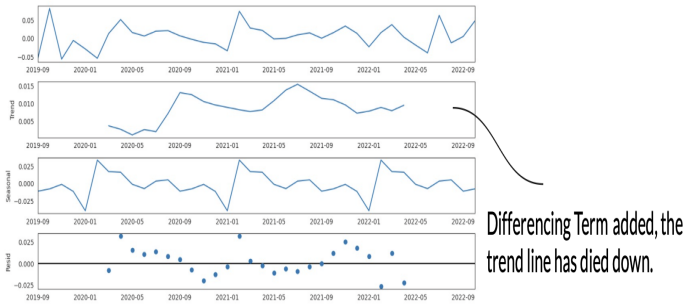

Fig. (5)

Fig(5) shows how adding a differencing term has helped in removing the trend. The next step in our feature engineering pipeline was performing Principal Component Analysis to find the principal components which capture almost 95% of variance in the data. Doing this allowed us to reduce the number of dimensions in the dataset before we started modeling.

We performed all of the EDA using both **pyspark** and **sklearn** and compared the time taken for different tasks in both of these libraries.

The table below represents the time taken for the functions:

Functions	PySpark (In Seconds)	Scikit-learn (In Seconds)
ReadData	2.4*e-5	8.7*e-6
DataPreparation	8.15*e-5	0.00395
BasicStatistics	4.29*e-5	0.01
FeatureImportances (RF)	8.21	10.6
PCA	9.2	29.2
Modeling	8.3	49.4

We can observe from the table that Pyspark performs better than sklearn in all of the tasks except reading the data. A brief description of what each function does is given below:

ReadData: ReadData function reads in the data into spark dataframes and pandas dataframes respectively.

Data Preparation: Data preparation function involves removing the null values from the data, changing the datatypes of a few columns(from string to date), performing one hot encoding on the features and merging two different dataframes.

BasicStatistics: BasicStatistics function is used for computing the county level and state level microbusiness densities to get a better understanding regarding the correlation.

FeatureImportances: FeatureImportances function involves fitting a Random Forest model on the dataset for finding the important features that could be used while training Machine Learning models to predict microbusiness density.

PCA: This function is used for applying PCA on the dataset to identify the principal components that capture the highest amount of variance in the data. This is done to reduce the number of features in the data before training machine learning models.

Modeling: This function involves training multiple Machine learning models but the time comparison shown above is for the Gradient Boosting Regressor.

Performance analysis:

Spark dataframes are generally faster than Pandas dataframes due to several reasons:

Distributed computing: Spark is designed to work with large datasets distributed across multiple machines, allowing it to process data in parallel and speed up computations.

Lazy evaluation: Spark uses lazy evaluation, which optimizes the execution plan and minimizes the number of operations needed to process the data.

Memory management: Spark is designed to handle data that is too large to fit into memory and uses a combination of memory and disk-based storage to manage data.

Optimization: Spark has a built-in optimizer that can automatically optimize queries and operations to make them faster.

Parallelism: Spark can process data in parallel across multiple nodes, whereas Pandas is limited to a single thread on a single machine.

Some of the reasons why we believe reading data was slower in spark:

1) The initial overhead of setting up parallelism in Spark can be a factor that makes reading data into Spark dataframes slower than Pandas. When you start a Spark application,

there is some overhead involved in setting up the distributed computing environment and initializing resources. This initialization process can take some time, and this overhead can make reading data into Spark dataframes slower than Pandas.

2) Depending on the format of the data, Pandas may be faster at reading it than Spark. For example, if the data is in a CSV or Excel format, Pandas has built-in functions to read these formats efficiently. Spark, on the other hand, may require additional steps to read these formats.

Modeling:

Random Forest Regressor:

Random Forest Regressor is an ensemble learning method for regression that works by combining multiple decision trees together.The algorithm works as follows, Data is randomly sampled with replacement from the original dataset to create multiple subsets of data. For each subset of data, a decision tree is trained using a random subset of features. The final prediction is made by aggregating the predictions of all the decision trees in the forest. During the training process, each decision tree is built using a subset of features and data samples. The subset of features is selected randomly at each split, and the data samples are selected using a technique called bootstrapping. This random selection process helps to reduce overfitting and increase the robustness of the model. When making a prediction, each decision tree in the forest generates its own prediction. The final prediction is obtained by aggregating the predictions of all the decision trees. In the case of regression, the predictions are averaged, and the result is the final prediction.Random Forest Regressor is a powerful algorithm that can handle high-dimensional data and nonlinear relationships between features and the target variable.

To Achieve Parallelism in Random Forest, We have Loaded the data into an RDD (Resilient Distributed Dataset). Splitted the data into multiple partitions for parallel processing. Used the Spark MLlib library to build a Random Forest Regressor model. Trained the model on each partition of the data in parallel using the parallelism features of Spark. Combined the results of each partition to obtain the final model. Used the trained model to make predictions on new data . By leveraging the parallel processing capabilities of Apache Spark, you can train a Random Forest Regressor model more quickly and efficiently, making it easier to handle large datasets and perform complex calculations.

Gradient Boosting Regressor:

Gradient boosting regressor is a type of machine learning algorithm that aims to predict continuous numerical values in supervised learning problems. The algorithm constructs a sequence of decision trees in an ensemble, where each tree attempts to rectify the errors made by the previous one. Initially, the algorithm constructs a single decision tree using training data, and the errors of this tree are computed by comparing its predicted values with the actual values of the training data. A second decision tree is then created to predict the errors made by the first tree, and the results of both trees are combined to update the model. This process is repeated for a specific number of trees, and each new tree endeavors to reduce the errors of the previous trees. Finally, the algorithm combines all the decision trees in the ensemble, giving each tree a

weight proportional to its contribution to the overall prediction.

To achieve parallelism in Gradient Boosted Regressor using Apache Spark, we can split the data into multiple partitions and perform training of trees independently on each partition. The training can be done in parallel by leveraging the distributed computing power of Apache Spark. Additionally, we can use techniques such as early stopping to reduce the number of iterations required to converge, thus improving the overall efficiency of the process.

LSTM:
LSTM is particularly useful for time series forecasting because it is able to capture the long-term dependencies in the time series data. In time series forecasting, the input data consists of a sequence of values over time, and the goal is to predict the future values of the series. LSTM is able to learn the patterns and trends in the input sequence and use them to make accurate predictions. By maintaining a memory of past inputs, the network is able to take into account the context of each input value and make predictions based on the entire history of the series. During training, the network is fed a sequence of input values and corresponding output values. The weights of the network are updated using backpropagation through time, which allows the network to learn to make accurate predictions based on the past inputs.

PySpark can be used to parallelize LSTM for time series forecasting by partitioning the data into smaller chunks and distributing the partitions across multiple worker nodes in a PySpark cluster. Each worker node trains its own LSTM model on its partition using the same LSTM model architecture and training algorithm as in a single-node setting. Once each worker node has trained its own LSTM model, the models are combined to make predictions on the entire time series. This can be done using various techniques, such as averaging the predictions or using an ensemble of models. Finally, the predictions from each partition are aggregated to generate a final prediction for the entire time series.

Talk about the different ML models used, their architecture and performance analysis.

Results and Conclusion(Validation Analysis):

All of the below models were tested on the unseen test set. Overall, we can see that the LSTM(Long Short-Term Memory) model performs the best across all the metrics.

Model	SMAPE	MSE	MAE
Random Forest	1.919	5.287	0.079
LSTM	0.684	0.006	0.050
Gradient Boosting	1.145	5.506	0.058