Executive Summary

Banking Sector is an ever-blooming sector of finance, without bank's support people would have difficulty getting investments, loans, etc. a nation's economy blooms because of the banks it has. to understand the various modes of transaction done by banks for the customers and by the customers this paper deals with studying the various banking transactions with the help of scan statistics. along the course of this project, we will be performing various statistical analysis like descriptive statistics & statistical statistics, we will be developing regression models and single smoothing models and comparing to them for better efficiency of model. we will be using forecast errors for determining the accuracy of forecast. we will also perform multicollinearity and autocorrelation test on the data. we will check the goodness of fit with the help of chi-square test and also check the consistency of data by ranking them based on the coefficient of variation. finally, we will find hotspots of the ATMs & CRMs present in India at state level.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

RBI	Reserve Bank of India
RMSE	Root Mean Square Error
MAD	Mean Absolute Deviation
SLR	Simple Linear Regression
MLR	Multiple Linear Regression
SE	Standard Error
SES	Single Exponential Smoothing
KPI	Key Performance Indicator
MAD	Mean Absolute Deviation
MAE	Mean Absolute Error
MSE	Mean Square Error
RMSE	Root Mean Square Error
MAPE	Mean Absolute Percentage Error
VIF	Variance Inflation Factor
R^2	R-Square
R^2_{Adj}	Adjusted R Square
PCA	Principal Component Analysis
CV	Coefficient of Variation

1 INTRODUCTION

1.1 Objective

1) Descriptive characterization and summary statistics of Banking Transactions of different modes. (NEFT, RTGS, Credit Card, Debit Card, Mobile Transactions)
2) To develop a Regression Model on Bank Transaction dataset and assessing the model fit.
3) Time Series analysis of Simple Exponential Smoothing of Bank Transaction dataset.
4) Developing consistency ranking for various modes of Transaction.
5) Conducting Chi-Square Test of goodness of fit & independence for deviation between observed (filled survey) & expected (theorical) frequency.
6) Identification of Hotspots (MLC) using ATM's, CRM's, etc. for State (2012-2022) level.
7) Investigating the nature of Hotspots.

1.2 Motivation

The banking sector has always been the backbone of the economy for any country in the world. Without the development of this sector, the country cannot develop as the banks provide investments, credit & infrastructure.

Therefore, the efficiency of banking services always needs to be monitored & improved. This is the motivation for the project to explore the various banking transactions provided by the banks to their customers.

To understand the transactions with the perspective of scan statistics i.e., descriptive statistics & inferential statistics.

Being a time series temporal data, we have performed regression analysis and exponential smoothing analysis on the data.

1.3 Background

There is no 1 or 2 research paper that covers the entirety of this project, this project is the accumulation of different strategies and techniques on a entirely new set of data of the biggest economic sector of the world, the banking sector.

[1] **[A. Rajwani, T. Syed, B. Khan and S. Behlim]** research results for regression techniques, including employing an LSTM model for time series, are presented in order to address the "Cash Estimate" issue. As a result, banks would be able to adapt to the shifting demands for cash related to certain events, holidays, etc. Banks will be able to successfully cut the additional costs that they incur for maintaining their cash as a result of this research, which will also help them satisfy more customers. [2] **[Akrati Saxena, Yulong Pei, Jan Veldsink, Werner van Ipenburg, George Fletcher, Mykola Pechenizkiy]** They created a network of 1.6 million nodes using the banking activities of Rabobank customers. The overall amount moved and the total number of transactions between users from 2010 to 2020 are the two weights assigned to each edge. They have examined the network's meso-scale characteristics in further detail and evaluated them against a randomised reference system. [3] **[Ali Abolhassani. Marcos O. Prates]** This paper examines the evolution of scan statistics over the past three decades, the primary concerns of scan statistics researchers, and how scholars have addressed these problems. [10] **[Dave, Isha and Patel, Raaj]** In this study, the authors examine consumer preferences for various payment methods, the variables influencing debit card usage, the frequency, the costs, and the motivations for utilizing debit cards after demonetization. The writers also made other comparisons between credit cards, net banking, and debit cards. The authors concluded that although debit cards are widely accepted and incredibly popular with clients, they are underutilized. [11] **[Feng Chen and Daniel B. Neill]** This study introduces Non-Parametric Heterogeneous Graph Scan (NPHGS), a novel method that considers the complete heterogeneous network for event detection. They used the detection of uncommon disease outbreaks and civil disturbance events as two applications leveraging Twitter data for our case study and then give empirical assessments demonstrating their usefulness and efficiency. [12] **[Gorodetskaya O, Gobareva Y, Koroteev M. A Machine]** In order to address optimisation issues in the banking industry, this paper outlines a methodical approach to developing a machine learning prediction model. This article describes a machine learning approach to economics that can produce reliable, repeatable results and be used to other tasks that are similar. The article's methodology was evaluated in three scenarios and demonstrated

the capacity to produce models that are at minimum 3 percent points more accurate than comparable predictive models stated in the literature. .[13] **[Wang, Haiqi; Kong, Haoran; Yan, Bin; Li, Liuke; Xu, Jianbo; Wang, Zhihai; Wang, Qiong/** Researchers suggested a dynamic cylinder with a changeable radius, they also suggested an enhanced GSA based on mental search (MSGSA), which was used to optimise the dynamic scanning window to find spatiotemporally anomalous clusters. Based on the obtained accuracies and error rates, simulations of trials using the MSGSA and SaTScan revealed that the MSGSA-optimized dynamic window produced greater results. [15] **[Hasan, Morshadul & Le, Thi Ngoc Quynh & Hoque]** The study's conclusions touch on big data's advantages, difficulties, and crucial data-driven financial issues like banking security. Big data operations are essential for data-driven financial choices, and this study will have a huge impact on the banking sector. [16] **[Hasheminejad, Seyed Mohammad Hossein and Reisjafari, Zahra]** Researchers have covered topics like predicting cash demand, detecting fraud, ATM failure, replenishing technique, ATM location, customer behavior, etc. in this paper. They examine AI methods like support vector machines, regressions, and neural networks, along with their output in the form of graphs, in various sections. [19] **[Kasi, Urmika]** The method for extracting transactional data, pre-processing it using pattern matching, and then classifying it according to the Merchant Category Codes (MCCs) class of the transactions is provided in this work. The model's accuracy, recall, and F-Measure were all 0.927 with a majority class assumption. These excellent performance indicators show that using Naive Bayes as a classifier was the best option. [21] **[Lee, Junho & Sun, Ying & Chang, Howard]** They suggest a mixed-effects model that considers spatial correlation when detecting spatial clusters. The added random effects explain additional geographic response variability beyond the cluster effect, compared to a fixed effects model, which lowers the false positive rate. To assess the effectiveness of the suggested strategy in terms of the true and false positive rates of a known cluster and the identification of multiple known clusters, simulated studies are used. [22] **[Lee, Junho, Kamenetsky, Maria E., Gangnon, Ronald E., Zhu, Jun]** Authors suggest changing of the coefficient regression approach in this article. They specifically expand a spatial-only data-changing coefficient regression model to spatial-temporal data with variable temporal patterns, they also evaluate the accuracy and recognition of known clusters. [23] **[M. Asad, M. Shahzaib, Y. Abbasi and M. Rafi]** By utilising a data-driven technique for the estimation of the correct amount for each ATM or some groups of ATMs, the research proposed a machine learning-based approach to ATMs replenishment amount prediction. The Root Mean Squared Error (RMSE) produced by the Long Short-Term Memory (LSTM) based model is highly

encouraging for this situation. [26] **[Padalkar, Ishang, Mahesh Kulkarni, Pranav Kulkarni, Sukrut Pendharkar and Anita Shinde]** Here, authors have focused on the clients and businesses doing business. matching (checksum) the total of the transactions made and, if there are any, marking them as fraudulent. The fuzzy logic algorithm is used in this study for search purposes. [27] **[Patil, Priyanka S. and Dr. Nagaraj V. Dharwadkar]** Here, researchers are focusing on fraud detection and customer retention. The supervised ANN algorithm is used in this work to classify data. [29] **[Prasanth, S., and S. Sudhamathi]** This study discovered that students' perceptions of banking facilities are influenced by the banks they choose to use. The Multivariate Analysis Test, the NSQ, and the Kolmogorov-Smirnov analyses were used to process the data. With this study, it was discovered that income, ethnicity, and youth are significant factors in online banking. [33] **[Shaikh, Imlak & Anwar, Mohd]** In this study, we look at how the performance of Indian banks is affected by digital bank transactions. On the surface, it appears that the proportion of banks in the public sector that engage in digital transactions with their counterparts in the private sector banks has decreased. The research suggests two applications: Initially, RTGS-based transactions must be promoted more quickly; this will improve the bank's performance. In order to lower the cost of cash, banks should enable credit-based digital transactions. [34] **[Shukla, Savinay & Vishesh, Siddesh & Pa, Priyanka]** To construct an ML model of prediction for this task, researchers have combined classification and regression methods. Using the formula method, a linear regression model is created from scratch. The dataset is successfully fitted using classification techniques including Support Vector Machine (SVM), Random Forest Classifier, and KNN. [35] **[Simutis, Rimvydas & Dilijonas, Darius & Bastina, Lidija]** In this study, the daily cash demand for ATMs is forecasted using two distinct methodologies. Based on a flexible artificial neural network, the first technique ANNin which using a unique adaptive regularisation term, the generalisation properties of this ANN were enhanced. The support vector regression (SPR) algorithm is used in the second forecasting technique. Notwithstanding the current too-optimistic assumptions about the capabilities of SPR, the analysis revealed that, for this application, a forecasting method based on flexible ANN can yield a little superior outcome. [37] **[Vijayakrishnan, R. & Vetrivel, S.]** The writers have made an effort to explain how branches affect how frequently RTGS and NEFT are used in Indian public-sector banks. The branch impact on volume has been explained using descriptive statistics and simple linear regression. It is clear that the new strategy shifts the emphasis to reevaluating the traditional delivery mechanism in the Indian Banking model. [38] **[Vollset, Erlend and Olav Eirik Ek Folkestad]** In order to increase the precision of a genuine bank transaction classification system, this study aims to investigate the

extent to which external semantic resources about organisations can be utilised. The classification method is based on a Bag-of-Words model and employs the classification technique of Logistic Regression. According to this study, enhancing bank transactions with outside firm data greatly increases the categorization system's accuracy. [40] **[Wu, Q., Glaz, J]** In this article, researchers examine the effectiveness of moving median-based scan statistics as test statistics for identifying a regional change in population mean for one- and two-dimensional normal data, where outliers are present, and the population variances are unknown. The multiple window scan statistics, implemented using the bootstrap method, worked quite well when the window size where the regional change of the population mean has happened is unknown. [41] **[X. Liu and P. Zhang]** The strategy for identifying suspicious sequences the transactions for financial institutions is proposed in this paper and is based on scan statistics. By using actual financial information from commercial institutions, we ultimately assess our algorithm. Furthermore, the outcomes of the initial experiment show how effective our strategy is. [42] **[Yeliz Ekinci, Nimet Uray, Fusun Ulengin]** The objective of this study is to create a practical and comprehensive model for customer lifetime value (CLV). By using the outcomes of least square estimation (LSE) and artificial neural network (ANN). This study demonstrates that in addition to the indicators that are frequently used in the literature to measure CLV, two other groups—the monetary value and risk of certain bank services, as well as indicators related to product/service ownership—are also relevant determinants. [44] **[Yi Wu, Yuwen Pan]** In this paper, the test datasets is cleansed, the discrete data is one-HOT coded, and the data are standardised in accordance with the usage scenario of credit evaluation of personal credit data. The pdC-RF technique is used in this research to maximise the correlation of data characteristics and decrease the dimensional data due to the high dimension of personal credit data. Lastly, the user samples are rated and the final score card is output based on the logistic regression model with the best parameters.

2 PROJECT DESCRIPTION

This paper deal with the scan or window statistics on the banking transaction data and also using various models to perform forecasting and find hotspot.

At first, collection of data from an authentic data source, for this project the data was collected from RBI. They had the data of various banks in India and their different transactions provided by them in a spreadsheet on a monthly-basis format. As per the project's problem definition data had to convert to yearly-basis with respect to their transaction types.

After the data has been prepared perform various summary statistics on the data to understand the dataset, its values better.

Then, perform regression modelling and simple exponential smoothing on the data set with the help of Python Language and Jupyter Notebook. Choosing the appropriate model for the forecast based on the forecast accuracy calculation of the data set. Checking the multicollinearity and autocorrelation in the selected model and fixing them, if any.

Performing chi-square test of Goodness of fit and test of independence on the dataset and concluding the results for both the test.

Determining the consistency of the banking transactions by creating a different dataset of different transaction modes with respect to the banks and understanding the results.

Finally, performing spatio-temporal analysis on the Banks ATMs data set on a state level.

PROJECT GOALS

I) To perform descriptive & inferential statistics.
II) To forecast the different banking transactions using different models.
III) To check the consistency of the banking transactions.
IV) To analyze hot spots of Bank ATMs.

3 TECHNICAL SPECIFICATION

3.1 Tools

❖ Jupyter Notebook

Jupyter Notebook is an open-source platform, a web-based interactive computational environment an IDE for Python Programming, a quick code experiment and to record and share your analysis of code snippets with others.

Jupyter Notebooks can be used for a variety of data science tasks, such as data cleansing and transformation, numerical simulation, exploratory data analysis, data visualization, statistical modelling, machine learning, deep learning, and more.

Like any normal handwritten notebook, these web notebooks can also be shared with others. It contains a list of input & output cells that contain code, headings, text, plots, etc. This notebook has a file extension of ".ipynb".

While the front-end of the Jupyter consists of rectangular text boxes for the programmer to enter code or text, whereas the back-end has a kernel which takes the code and executes it, to provide results.

❖ Microsoft 365 Excel

A spreadsheet application developed by Microsoft for computing, calculating, manipulating data which is stored in the form of tables (rows & columns). Its file extension are ".xls", ".xlsx", ".csv", etc.

It has many in-built functions and formulas, which is used for performing various operations in the cells where the data is stored. Once a formula is entered in an Excel spreadsheet, it

automatically does the desired calculations and gives the results of the formula we entered or used.

❖ SaTScan

Along with Information Management Services Inc., Martin Kulldorff created the SaTScan programme. It is a free piece of software that uses scan statistics to evaluate spatial, temporal, and space-time data.

A good place to start when looking for circular clusters using scan statistics is the work of Kulldorff and Nagarwalla (1995). Finding non-circular clusters became more significant as a result.

Following are some examples of the statistical techniques used by SaTScan: Bernoulli Model, Discrete Poisson Model, Space-Time Permutation Model, Multinomial Model, Ordinal Model, Exponential Model, Normal Model, Continuous Poisson Model, Probability Model Comparison, and Likelihood Ratio Test.

Several files should contain the input data. All probability models save the continuous Poisson model require a case file and a coordinates file. While the Bernoulli model needs a control file, the Poisson model also needs a population file.

3.2 Language

❖ Python

Python is a high-level, object-oriented, general-purpose, interpreted programming language with dynamic semantics.

The most popular uses of Python are in the creation of websites, software, task automation, data analysis, and data visualisation.

Python is a crucial tool for data science jobs and is used to build machine learning algorithms, carry out intricate statistical computations, and visualise data.

The primary users of the Python-based SciPy ecosystem include scientists, engineers, financial analysts, mathematicians, and other specialists engaged in data-driven research. It includes potent data visualisation and analysis tools like pandas, NumPy, scikit-learn, and matplotlib.

➢ Libraries used:

- Numpy

Numerical Python is referred to as NumPy. The n-dimensional array is NumPy's most potent feature. Additionally, this library includes facilities for integrating with other low level languages including Fortran, C, and C++ as well as fundamental linear algebra algorithms, Fourier transforms, and advanced random number capabilities.

Several programmers and data scientists utilise NumPy for AI (artificial intelligence) and ML (machine learning) projects since it can also manipulate multidimensional arrays.

- Pandas

Pandas a library built on top of Numpy, it is widely used for data preprocessing and munging. Pandas, a relatively new addition to Python, have greatly increased the language's popularity among data scientists.

This library enables users to create high-level data structures that are seamless and understandable. Pandas is used in many different fields, including statistics, engineering, and even banking.

The versatility of Pandas and its compatibility with other Python scientific and mathematical libraries are two things that make it outstanding.

- Openpyxl

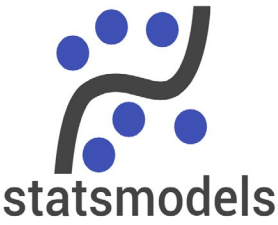
One of those things you likely encounter at some time is Excel spreadsheets, in which case understanding openpyxl will be helpful.

Without the need for additional Microsoft application software, Excel files can be worked with using the Python language's Openpyxl module. With this module, we can alter Excel without having to open the programme. It is used to carry out Excel

operations like accessing an Excel sheet, renaming the sheet, editing the sheet, formatting, and styling the sheet. It is also used to read data from an Excel file or write data to an Excel file, create charts, and perform other operations on the sheet.

Openpyxl is often used by data scientists to carry out numerous activities like data copying, data mining, and data analysis.

- statsmodels



statsmodels offers classes and functions for estimating a variety of statistical models, running statistical tests, and exploring statistical data. For each estimator, a comprehensive collection of result statistics is supplied. To verify that the findings are accurate, the results are compared to those of current statistical software.

Users can explore data, estimate statistical models, and run statistical tests using the Python library Statsmodels.

For various types of data and each estimator, a comprehensive array of descriptive statistics, statistical tests, charting functions, and outcome statistics are available.

The mathematical libraries NumPy and SciPy are the foundation of Statsmodels, which also combines Pandas for data handling and Patsy[3] for an R-like formula interface.

- Scikit-learn

The most effective and reliable Python machine learning library is called Sklearn (Skit-Learn), via a Python consistency interface, it offers a variety of effective tools for

statistical modelling and machine learning, including classification, regression, clustering, and dimensionality reduction.

This library is based on NumPy, SciPy, and Matplotlib and was written primarily in Python.

- SciPy

SciPy is a common abbreviation for Scientific Python. Based on NumPy so it is free and open source.

For many high-level science and engineering modules, such as discrete Fourier transform, linear algebra, optimisation, and sparse matrices, SciPy is one of the most helpful libraries.

SciPy performs well when used for image editing. It features advanced instructions that are frequently used for manipulating and displaying data.

- Matplotlib

Matplotlib is frequently promoted as a substitute for the more expensive MATLAB. To construct graphs and plots, the SciPy extension Matplotlib was developed for data visualisation. Both the intricate data models produced by Pandas and the NumPy data structures can be used with Matplotlib.

There is a limitation to Matplotlib; it can only perform 2D plotting. This library is still quite good at producing publish-ready data visualisations in the form of plots, diagrams, histograms, plots, scatter plots, error charts, and of course, bar charts.

- Seaborn

Seaborn is a Python package designed for charting and data visualisation. It was built on Matplotlib; however, it also incorporates elements of Pandas' intricate data structures. Users of Seaborn can build statistics graphs that are not only accurate but also educational because to the high-level interface's wealth of capabilities.

With Seaborn, visualisation will become a key component of data exploration and comprehension.

- D-Tale

D-Tale provides you with a simple way to examine and analyse Pandas data structures using the combination of a Flask back-end and a React front-end.

It works perfectly with Python/Ipython terminals and Ipython notebooks.

The Pandas objects DataFrame, Series, MultiIndex, DatetimeIndex, and RangeIndex are currently supported by this tool.

- Plotly

An interactive, open-source plotting toolkit for Python, plotly provides over 40 different chart types for a variety of statistical, financial, geographic, scientific, and three-dimensional use-cases.

4 DESIGN APPROACH AND DETAILS

4.1 Design Approach / Materials & Methods

The first step is to perform the Summary Statistics on the dataset in excel by using its various in-built formulas.

These would include descriptive measures like –

i. Count – To determine the no. observations in the dataset.

ii. Minimum – To determine the minimum value in the observation of the data.

iii. Maximum – To determine the maximum value in the observation of the data.

iv. Mean – To compute the arithmetic mean of the observations.

v. Median – To determine the mid-point of the data.

vi. Mode – To determine the frequency of repeated values.

vii. Quartile 1 – To compute the lower 25% of data points of the dataset.

viii. Quartile 2 – Similar as Median, 50% of data points selection.

ix. Quartile 3 – To compute the upper 75% of data points of the dataset.

x. Quartile Deviation – It is difference between the quartile 3 & quartile 1 divided by 2.

xi. Coefficient Of Quartile Deviation – It is the used estimate the variability of the dataset.

xii. Variance – To compute distance between the mean and data points.

xiii. Standard Deviation – To compute the distance between the data points.

The project is based on modes like Regression Model & Single Exponential Smoothing Model.

Regression Model:

Linear Regression are of two types:

i. Simple Linear Regression (SLR)
   SLR is a model with 1 dependent variable & 1 independent variable where the value of independent variable is used to predict the value of dependent variable.

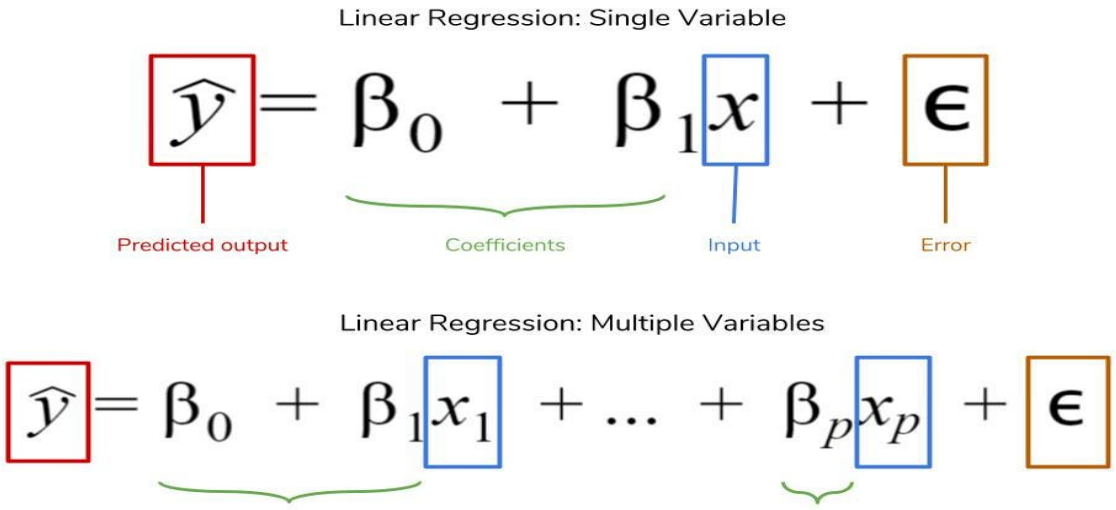
ii. Multiple Linear Regression (MLR)

MLR is a model with 1 dependent variable & more than 1 independent variables where the value of independent variables is used to predict the value of dependent variable.

Since we have, 1 dependent variable that is the Total calculated value of the various banking transactions for any given year of last decade and sorted in yearly fashion of ascending order.

Other variables are independent variables which are the transactions of the banks.

All the data has been stored in the excel file in rows and columns format for easy access to Jupyter Notebook and perform data analysis on the said dataset.



Equation 1: SLR & MLR

The dependent variable is also called as response variable or predict variable and is usually denoted with the letter 'y'.

The independent variable is also called as regressors or predictors and is usually denoted with the letter 'x_i'. {i.e. x1, x2, x3, …. , xn Depending upon the no. of predictors.}

'Ɛ' is the error term with mean as 0, which represents that the actual value of y may not be equal to the coefficients and input $\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$.

There is also, Hypothesis Testing done in MLR, while H_0 signifies that there is a significant effect of x_i on the y i.e., other independent variables does have a significant effect on the dependent variable in the regression equation, whereas H_1 signifies that there is no significant effect of x_i on the y i.e., other independent variables does not have a significant effect on the dependent variable in the regression equation.

Regression model is used for forecast, the best way to achieve that is selecting the equation with lowest value of SE, since that will yield the most accurate forecasts that is possible to make.

A measure of the accuracy of predictions derived from regression is given by SE.

Exponential Smoothing:

The most popular category of processes for smoothing discrete time series in order to predict the near future is exponential smoothing.

Using the smoothed series to predict future values of the target variable is the rationale behind exponential smoothing, which smoothes the original series in a similar manner to how the moving average works.

In contrast, we want to allow the more recent values of the series to have a bigger impact on the prediction of future values than the more distant observations in exponential smoothing.

i. Single/Simple Exponential Smoothing (SES)

   Among all forecasting methods, the SES forecasting approach is the most popular. Little computation is needed. When data patterns are roughly horizontal (i.e., there are neither cyclical variations nor clear trends in the historical data), this method is applied.

$$F_{t+1} = \alpha y_t + (1-\alpha) F_t$$

Where F_{t+1} → Forecasted value of time series at time t+1

Ft → Forecasted value of time series at time t

Equation 2: Single Exponential Smoothing

ii. Double Exponential Smoothing (DES)

   Double exponential smoothing's fundamental principle is to include a term to account for the likelihood that a series may show some sort of trend. With exponential smoothing, this slope component is also updated.

   It is also known as Holt's Trend method.

1. $\mu_t = \alpha y_t + (1 - \alpha)(\mu_{t-1} + T_{t-1})$
2. $T_t = \beta(\mu_t - \mu_{t-1}) + (1 - \beta)T_{t-1}$
3. $F_{t+m} = \mu_t + mT_t$

where

$\mu_t \rightarrow$ *Exponentially smoothed value of the series at time t*

$y_t \rightarrow$ Actual observation of time series at time t

$Tt \rightarrow$ Trend Estimate

$\alpha \rightarrow$ Exponential Smoothing Constant for the data

$\beta \rightarrow$ Smoothing constant for trend

$F_{t+m} \rightarrow$ m period ahead forecasted value

Equation 3: Double Exponential Smoothing

iii. Triple Exponential Smoothing (TES)

Three times of exponential smoothing used is the procedure of TES. When the data exhibits both a linear trend and a seasonal pattern, this method is used to forecast the time series.

It is also known as Holt-Winters method.

- Exponentially smoothed series equation

  $\mu_t=\alpha\ (y_t-S_{t-p}) + (1-\alpha)\ (\mu_{t-1}-+b_{t-1})$ $\quad 0\leq \alpha \leq 1$

- Trend estimating equation

  $b_t = \beta(\mu_t- \mu_{t-1}) + (1-\beta)\ b_{t-1}$

- Seasonality updating equation

  $S_t = \gamma\ (y_t-\mu_t)+ (1-\gamma)\ S_{t-p}$

- Forecast equation

  $F_{t+m} = \mu_t + m\ b_t+S_{t+m-p}$

where

μ_t →Exponentially smoothed value of the series at time t

y_t →Actual observation of time series at time t

T_t →Trend Estimate

α→Exponential Smoothing Constant for the data

β→Smoothing constant for trend

γ→ Smoothing constant for seasonality

F_{t+m}→m period ahead forecasted value

p →the period of seasonality (p=4 for quarterly data & p=12 for monthly data)

Equation 4: Triple Exponential Smoothing

Forecast Accuracy Metrics:

Despite the availability of numerous forecasting techniques, it is recommended that we perform a variety of forecast models and compare them to determine which model is providing us with a more accurate forecast.

It is also known as Forecast Error, Our model is more precise the lower the forecast error. Since there is no universal indicator, determining forecast accuracy (or error) is not a simple job. What a KPI is optimal for you can only be determined through experimentation.

These KPI's are:

i. MAD\MAE

The MAD calculation averages the prediction errors' absolute values (the discrepancy between real demand and the forecast) over the anticipated time frames. Absolute value denotes that even a negative difference between the real and anticipated demand turns into a positive value.

$$MAE = \frac{1}{n}\sum |e_t|$$

Equation 5: Mean Absolute Deviation/Error

ii. MSE

MSE is the model's parameter count minus the total of the squared discrepancies between the fitted values and the observed values, divided by the number of historical points. The number of model parameters is deducted from the total number of historical points in order to be consistent with an objective model variance estimate.

$$MSE = \frac{1}{n}\sum e_t^2$$

Equation 6: Mean Square Error

iii. RMSE

We may compute the squared root of the MSE to prevent the loss of its unit. The result is a new error measure known as RMSE.

It has the same benefits as its brothers and sisters, MAE and MSE. But it is also susceptible to outliers, much like MSE.

$$RMSE = \sqrt{\frac{1}{n}\sum e_t^2}$$

Equation 7: Root Mean Square Error

iv. MAPE

One of the most often employed metrics for error in forecasting is MAPE. It is determined by dividing the actual values by the absolute difference that exists between actual values and the projected values. This yields the average (mean).

$$MAPE = \frac{1}{n}\sum \frac{|e_t|}{d_t}$$

Equation 8: Mean Absolute Percentage Error

Evaluating these metrics for both the models i.e., Regression and Single Smoothing, we will compare the error values of both the models with each other and choose the model which has the least error value.

Once, we have decided which model to consider for further analysis, we need to identify multicollinearity severity.

Multicollinearity is a concept where independent variables in a regression model have a high correlation with one another. It complicates model interpretation and leads to an overfitting issue.

To check multicollinearity, we compute a Correlation Matrix and\or VIF.

Correlation Matrix is a matrix shows the relationship between each set of potential value pairs in a table.

VIF is a measurement of the amount that multicollinearity in the model increases the variance of a regression coefficients. In a multiple regression model, it is utilised to find independent variables that are correlated.

$$VIF = \frac{1}{1 - R^2}$$

Equation 9: Variance Inflation Factor

R^2 is stat measure where the variance of one variable is partially explained by the variance of the second variable.

R^2_{Adj} is the correctness in the accuracy of model, as it compensates the limitations of R^2 i.e., optimistic estimate of the fit of model by penalizing for adding new variables that do not enhance the model in any way.

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where
R^2 Sample R-Squared
N Total Sample Size
p Number of independent variable

Equation 10: Adjusted R-Square

Note: "$R^2_{Adj} < R^2$"

After computing this value, we evaluate them, if the VIF value is less than 10 then we accept the model and conclude that there is Less to No multicollinearity else if the value of VIF is above 10 then their multicollinearity.

VIF value 1 is the best case which indicates that there is no multicollinearity at all (Variable are not correlated). VIF value ranging in 1-5 is considered Moderately Correlated.

To fix multicollinearity, perform –

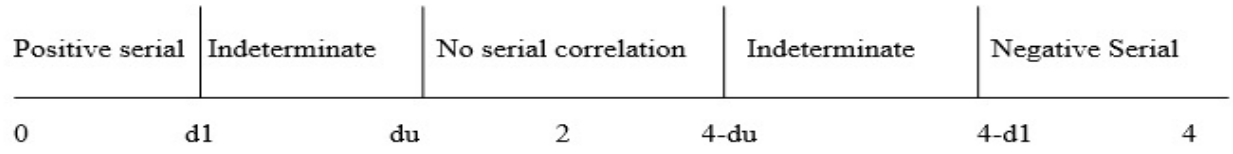
i. Variable Selection – remove highly correlated variable from the dataset.
ii. Variable Transformation – transform some variables to decrease their correlation, while maintaining the features
iii. PCA – reduce the dimension of the dataset (reducing predictors).

Once, the severity of multicollinearity has been fixed then we check for autocorrelation in the data.

Autocorrelation is the term for the relationship that develops over time between a time series and its lagging form. The OLS estimator is biased, if autocorrelation exist, which contradicts the presumption.

To check for autocorrelation, we perform the Durbin-Watson test.

Durbin-Watson Test is used to study autocorrelation in the difference of predicted and observed values, with acceptable range of 1.50 to 2.50.



To deal with autocorrelation, we can perform –

i. Include dummy variable in the data.
ii. Estimated Generalized Least Squares
iii. Include a linear (trend) term if the residuals show a consistent increasing or decreasing pattern.

Chi-Squared Test:

For categorical data, a statistical test called the Pearson's chi-square test is used. It is employed to assess whether your data significantly depart from your expectations.

Pearson's chi-square tests are divided into two categories:

i. Goodness of Fit

To determine whether the frequency distribution of a categorical variable deviates from your expectations, apply the chi-square goodness of fit test.

If your calculated value is less than tabulated value of chi-square test than that means there is no significance between the observed and expected frequencies and vice versa.

ii. Test of Independence

To determine if two category variables are connected to one another, the chi-square test of independence is utilised.

If your calculated value is less than tabulated value of chi-square test than that means the observed and expected frequencies are independent and vice versa.

$$X^2 = \sum \frac{(\text{observed - expected})^2}{\text{expected}}$$

Equation 11: Chi-Square

We want to know if the chi-squared value in our sample is so uncommon that we should reject the null hypothesis for the entire population. The background for that determination is provided by the chi-squared distribution.

Compare calculated value of the chi-square with the tabular value of chi-square from chi-distribution table with the help of degree of freedom = (r-1)(c-1) where r = number of rows and c = number of columns and alpha = 0.05.

Coefficient of Variation:

The standard deviation to mean ratio, or coefficient of variation (CV), illustrates the degree of variability in proportion to the population mean. The dispersion increases with increasing CV.

$$\text{Coefficient of Variation (CV)} = \frac{\text{Standard Deviation (s)}}{\text{Sample Mean } (\bar{x})}$$

Equation 12: Coefficient of Variation

The coefficient of variation is helpful because it may be compared between data sets with various units or significantly different means because it is dimensionless (i.e., regardless of the scale for which the measurement was taken).

We perform CV for various transactions of the banks like credit cards, debit cards, neft, rtgs, and mobile banking with respect to the no. banks for that particular year.

CV helps to measure the consistency & uniformity of the data, the higher its consistency as smaller is value.

Hotspot:

A spatial analysis and mapping technique called hotspot analysis looks for clustering in spatial occurrences.

Finally, we will perform a hotspot analysis on the no. of ATMs available in India state-wise and district-wise over the past decade.

We will perform the analysis with help of SaTScan application which is freely available.

We will require 3 files namely Location File, Case File & Population File for the analysis.

4.2 Codes and Standards

- In this project, we have taken data from the RBI's website for the research work, we assume that the data is verified and authenticated and to best knowledge of RBI.
- We have used Python Jupyter Notebook to carry out the analysis as it one of the best available tools to do Data Analysis.
- Specifically, we used NumPy, pandas, matplotlib, seaborn and plotly packages of python for data cleaning and basic visualizations.
- We, imported the Data and conducted the pre-processing i.e. cleaning the data, removing the outlier and made it ready for the analysis.
- We had cleaned the data month by month for every year, as the available data was monthly-basis on the RBI.
- After cleaning the data we merged them together, by taking sum of all the numeric values of variables and created a new data frame for that respective year and so on for the rest of the data.
- Columns like Bank Name, ATMs Onsite & Offsite are only one were not merged as Bank Name is an object type and there is no meaning merging them.
- The monthly data on variables ATMs Onsite & Offsite were the sum of ATMs till that particular month so merging them would be give me wrong data, that's why we considered the data for every year's month end (i.e., December) and concatenated that with the previously merged data frame of that year.

- Before the data was finally set to be use for analyses, we also created new variable inside the yearly data frame naming them ATMs Total, Debit Card Transaction Total & No. of Cards Total and similarly of Credit Cards.
- Finally, the data was ready to be showcased for various plots like Line Plot, Scatter Plot, correlation matrix, etc. to get insights and concluded.

```
In [2]: import pandas as pd

In [3]: df1104 = pd.read_excel('D:/Study/Sem 2/Project/ATM/2011/APR2011.xls')
        df1104 #Preview of the raw dataset
```

Out[3]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12
0	NaN	Bank wise ATM and Card Statistics - April 2011	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	Sr. No.	Bank Name	Number of ATMs	NaN	Number of POS terminals	NaN	Credit Cards	NaN	NaN	NaN	NaN	Debit Cards
2	NaN	NaN	NaN	On-site	Off-site	On-line	Off-line	No .of outstanding cards	No. of Transactions	NaN	Amount of transactions	NaN	No .of outstanding cards
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	(Actuals)	NaN	(Rs Million)	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	at ATM	at POS	at ATM	at POS	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
56	NaN	52	DBS Ltd.	4	26	0	0	0	0	0	0	0	1167
57	NaN	53	HSBC	72	79	9472	5171	910559	3288	1102183	23.166191	2948.921332	584020
58	NaN	54	Oman International Bank SAO	1	0	0	0	0	0	0	0	0.0269	0
59	NaN	55	Standard Chartered Bank	95	224	16	0	1095738	2574	1698682	12	5190	762176
60	NaN	NaN	Total	41268	34377	585714	10244	17777686	171978	23227778	963.720952	70553.980764	230256833

61 rows × 17 columns

- Here, the column names are improperly stated, the useful data starts from the 5th row.
- From the 55th row onwards, the column descriptions have been given and they need to be removed as well.
- The column "Unnamed: 0" stores garbage data as in the excel file, the 1st row has been left blank.

```
In [4]: df1104 = df1104.drop(df1104.index[0:5])
        df1104 = df1104.drop(df1104.index[55:])

        df1104 = df1104.drop(df1104.columns[[0,1]],axis=1)
        df1104
```

15	Indian Overseas Bank	726	330	651	0	34454	2912	30135	6	66.3	2710632	3342723	229061
16	Oriental Bank of Commerce	883	318	857	0	0	0	0	0	0	2634081	4056753	69918
17	Punjab and Sind Bank	83	2	0	0	0	0	0	0	0	48669	17281	0
18	Punjab National Bank	3033	2195	176	0	80999	2277	84560	10	183.6	13567190	36192527	647495
19	Syndicate Bank	1021	201	376	0	63431	2007	41911	7.7	103.3	5324961	4485495	227842
20	UCO Bank	426	198	0	0	0	0	0	0	0	1277651	1442942	65292
21	Union Bank of India	1832	804	2709	0	37971	818	35948	3.4	103.4	6530298	12066286	277161

- The data frame looks much better. But the columns aren't named, and the index seems to starts from 4 rather than 0.

Now that the dataframe is in a proper format, the datatypes and missing values, if any, in the dataframe need to be checked.

```
In [6]: df1104.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 15 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Bank Name                             55 non-null     object
 1   ATMs Onsite                           55 non-null     object
 2   ATMs Offsite                          55 non-null     object
 3   POS Online                            55 non-null     object
 4   POS Offline                           55 non-null     object
 5   CC outstanding at the end of the month  55 non-null     object
 6   CC No. of transactions ATM            55 non-null     object
 7   CC No. of transactions POS            55 non-null     object
 8   CC Amount of transactions ATM         55 non-null     object
 9   CC Amount of transactions POS         55 non-null     object
 10  DC outstanding at the end of the month  55 non-null     object
 11  DC No. of transactions ATM            55 non-null     object
 12  DC No. of transactions POS            55 non-null     object
 13  DC Amount of transactions ATM         55 non-null     object
 14  DC Amount of transactions POS         55 non-null     object
dtypes: object(15)
memory usage: 6.6+ KB
```

- There are no missing values, but all the integer columns are stored as objects.

```
In [7]: #Converting all the columns stroing integer data into integer format.

df1104[["ATMs Onsite", "ATMs Offsite", "POS Online", "POS Offline", "CC outstanding at the end of the month", "CC No. of transact
        "CC No. of transactions POS", "CC Amount of transactions ATM", "CC Amount of transactions POS", "DC outstanding at the
        "DC Amount of transactions ATM", "DC Amount of transactions POS"]] = df1104[["ATMs Onsite", "ATMs Offsite", "POS Onlin
        "CC No. of transactions POS", "CC Amount of transactions ATM", "CC Amount of transactions POS", "DC outstanding at the
        "DC Amount of transactions ATM", "DC Amount of transactions POS"]].apply(pd.to_numeric)
df1104.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 15 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Bank Name                             55 non-null     object
 1   ATMs Onsite                           55 non-null     int64
 2   ATMs Offsite                          55 non-null     int64
 3   POS Online                            55 non-null     int64
 4   POS Offline                           55 non-null     int64
 5   CC outstanding at the end of the month  55 non-null     int64
 6   CC No. of transactions ATM            55 non-null     int64
 7   CC No. of transactions POS            55 non-null     int64
 8   CC Amount of transactions ATM         55 non-null     float64
 9   CC Amount of transactions POS         55 non-null     float64
 10  DC outstanding at the end of the month  55 non-null     int64
 11  DC No. of transactions ATM            55 non-null     int64
 12  DC No. of transactions POS            55 non-null     int64
 13  DC Amount of transactions ATM         55 non-null     float64
 14  DC Amount of transactions POS         55 non-null     float64
dtypes: float64(4), int64(10), object(1)
memory usage: 6.6+ KB
```

- Converting all the columns expect Bank Names from Object type to Numeric Type.
- Renaming Columns.

```
In [8]: df1104.describe()
```

Out[8]:

	ATMs Onsite	ATMs Offsite	POS Online	POS Offline	CC outstanding at the end of the month	CC No. of transactions ATM	CC No. of transactions POS	CC Amount of transactions ATM	CC Amount of transactions POS	DC outstanding at the end of the month	DC No. transactio A
count	55.000000	55.000000	55.000000	55.000000	5.500000e+01	55.000000	5.500000e+01	55.000000	55.000000	5.500000e+01	5.500000e+
mean	750.327273	625.036364	10649.345455	186.254545	3.232307e+05	3126.872727	4.223232e+05	17.522199	1282.799650	4.186488e+06	7.264606e+
std	1567.033303	1468.234591	37872.327215	864.469549	9.019618e+05	8349.677168	1.158165e+06	52.424823	3548.168997	1.012538e+07	2.051577e+
min	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+
25%	110.000000	77.500000	0.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000	0.000000	4.710885e+05	4.807750e+
50%	372.000000	200.000000	0.000000	0.000000	1.611000e+03	0.000000	3.221000e+03	0.000000	7.800000	1.559268e+06	2.029143e+
75%	707.000000	456.000000	1695.000000	0.000000	9.961200e+04	2351.000000	8.149350e+04	9.350000	203.250000	4.421444e+06	5.051332e+
max	10962.000000	9318.000000	186708.000000	5171.000000	5.089800e+06	50701.000000	5.344558e+06	298.968633	17822.830808	7.313400e+07	1.448090e+

The data seems to be proper.

All the columns have the minimum value 0 and the standard deviation is large almost everywhere.

There is a lot of variation in the data, which might indicate the presence of outliers that affect the mean value.

The column "POS Offline" has the value 0 everywhere. Point of Sale (POS) offline transactions are the ones which take place in standalone mode without network (Internet) or cloud support. Such method of payment isn't popular in India and hence, the numbers can be considered genuine.

This column is dropped as it doesn't add much to the analysis.

Now, other transaction modes.

```
import pandas as pd
```

```
df01= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0112.xls', sheet_name='MB')
df02= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0212.xls', sheet_name='MB')
df03= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0312.xls', sheet_name='MB')
df04= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0412.xls', sheet_name='MB')
df05= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0512.xls', sheet_name='MB')
df06= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0612.xls', sheet_name='MB')
df07= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0712.xls', sheet_name='MB')
df08= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0812.xls', sheet_name='MB')
df09= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\0912.xls', sheet_name='MB')
df10= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\1012.xls', sheet_name='MB')
df11= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\1112.xls', sheet_name='MB')
df12= pd.read_excel(r'D:\Study\Project\Bank Transaction Analysis\NEFT\2012\1212.xls', sheet_name='MB')
```

```
dff = pd.concat([df01,df02,df03,df04,df05,df06,df07,df08,df09,df10,df11,df12])
dff
```

- Import pandas library and store excel files of all the previous years from 2012 to 2022 in separate data frames each year wise and then concating them into a single data frame.

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Mobile Banking Transactions data for the month of June, 2012	Bank-wise Mobile Banking Transactions data for the month of August, 2012
0	NaN	Mobile Banking Transaction of volume and valu...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	Volume in Actual and Value in Thousand Rupees	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	Bank Name	Total Volume and Value of Mobile Banking trans...	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	Volume	Value	NaN	NaN	NaN	NaN	NaN
4	NaN	BARCLAYS BANK PLC	3553	9296	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
48	NaN	47	UCO BANK	5732	7006.74	NaN	NaN	NaN	NaN
49	NaN	48	UNION BANK OF INDIA	48218	66847.82	NaN	NaN	NaN	NaN
50	NaN	49	UNITED BANK OF INDIA	3910	5628.05106	NaN	NaN	NaN	NaN
51	NaN	50	VIJAYA BANK	6419	10487.7	NaN	NaN	NaN	NaN
52	NaN	NaN	Total	5221007	5981382.782	NaN	NaN	NaN	NaN

627 rows × 9 columns

```
1 dff.to_csv('MB.csv')
```

- After verifying successful concatenation, save the data frame toa csv or xlsx file format.

Similarly, concatenate the files of NEFT Transactions, RTGS Transactions, Debit Card Transactions and Credit Card Transactions.

4.3 Constraints

1) Unavailability of yearly data of transaction modes.
2) Unavailability of customer data due data privacy.
3) Requirement of a better system for running the models.

5 SCHEDULE, TASK AND MILESTONE

S.NO	MONTH WEEK	PLAN
1	DECEMBER WEEK 3	PROBLEM FORMULATION
2	JANUVARY WEEK 1	BACKGROUND CHECK
3	JANUVARY WEEK 2	OBJECTIVE & PROBLEM DEFINITION
4	JANUVARY WEEK 3,4	DATA COLLECTION, CLEANING
5	FEBRUARY WEEK 1	DATA VISUALIZATION
6	FEBRUARY WEEK 2	DATA ANALYSIS
7	FEBRUARY WEEK 3	THIRD REVIEW
8	MARCH WEEK 1,2	MODEL BUILDING & TESTING
9	MARCH WEEK 3,4	DOCUMENTATION AND REPORT WRITING
10	APRIL WEEK 1	REPORT REVIEW
11	APRIL WEEK 2	FINAL REVIEW

Table 1: Schedule

6 PROJECT DEMONSTRATION

In [1]:
```
import pandas as pd
from statsmodels.tsa.api import SimpleExpSmoothing
import plotly.express as px
```

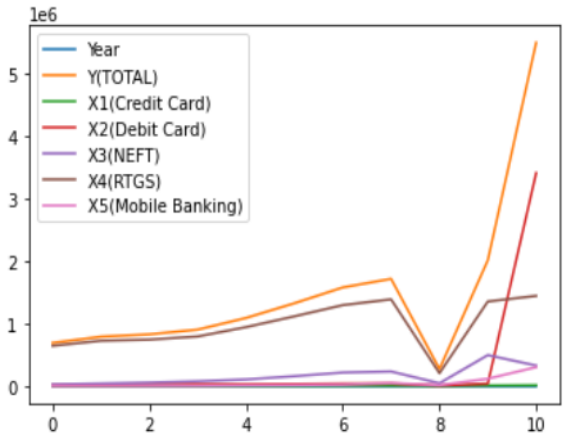
In [2]:
```
df=pd.read_excel(r"C:\Users\jigne\OneDrive\Desktop\New folder\2.xlsx")
df
```

Out[2]:

	Year	Y(TOTAL)	X1(Credit Card)	X2(Debit Card)	X3(NEFT)	X4(RTGS)	X5(Mobile Banking)
0	2012	6.912864e+05	1176.915295	1.666356e+04	25887.73695	6.447141e+05	41.853287
1	2013	7.910072e+05	1468.181288	1.991353e+04	39919.76326	7.259831e+05	161.934560
2	2014	8.284560e+05	1828.128037	2.287817e+04	55339.51744	7.439936e+05	666.247640
3	2015	9.038585e+05	2297.109326	2.585332e+04	75985.79897	7.941606e+05	2862.592590
4	2016	1.090648e+06	3011.659023	2.697110e+04	106103.79310	9.437279e+05	10288.908640
5	2017	1.326151e+06	4335.728709	3.160950e+04	157997.26160	1.116611e+06	15466.695400
6	2018	1.579515e+06	5701.087934	3.829477e+04	216347.84720	1.296190e+06	22840.465970
7	2019	1.717419e+06	7179.802832	4.090007e+04	232966.47130	1.388670e+06	47618.096630
8	2020	2.727253e+05	1312.495010	6.928071e+03	44877.67696	2.053994e+05	14207.686240
9	2021	2.015006e+06	8909.459800	3.844364e+04	495839.09900	1.354685e+06	117128.147700
10	2022	5.500892e+06	13335.165300	3.412431e+06	327951.42390	1.444560e+06	302614.025100

In [3]:
```
df.plot()
```

Out[3]: <AxesSubplot:>



- Import excel data into Jupyter and the plot data for a basic reference.

```
dff=df[['Y(TOTAL)']]
ses = SimpleExpSmoothing(dff)
alpha=0.5
model = ses.fit(smoothing_level = alpha, optimized = False)
foracast=model.forecast(11)
```

```
import math
import numpy as np
actual = dff['Y(TOTAL)'].to_numpy()
predicted = foracast.to_numpy()

MSE = np.square(np.subtract(actual,predicted)).mean()

rsme = math.sqrt(MSE)
print("Root Mean Square Error:\n")
print(rsme)
```

```
Root Mean Square Error:

2379721.0751185706
```

- Performing Single Exponential Smoothing on the data and calculating RMSE foe the same.

```
y = pd.DataFrame(df2.iloc[:,-6])
y
```

	Y(TOTAL)
0	6.912864e+05
1	7.910072e+05
2	8.284560e+05
3	9.038585e+05
4	1.090648e+06
5	1.326151e+06
6	1.579515e+06
7	1.717419e+06
8	2.727253e+05
9	2.015006e+06
10	5.500892e+06

```
x = pd.DataFrame(df2.iloc[:,2:7])
x
```

	X1(Credit Card)	X2(Debit Card)	X3(NEFT)	X4(RTGS)	X5(Mobile Banking)
0	1176.915295	1.666356e+04	25887.73695	6.447141e+05	41.853287
1	1468.181288	1.991353e+04	39919.76326	7.259831e+05	161.934560
2	1828.128037	2.287817e+04	55339.51744	7.439936e+05	666.247640
3	2297.109326	2.585332e+04	75985.79897	7.941606e+05	2862.592590
4	3011.659023	2.697110e+04	106103.79310	9.437279e+05	10288.908640
5	4335.728709	3.160950e+04	157997.26160	1.116611e+06	15466.695400
6	5701.087934	3.829477e+04	216347.84720	1.296190e+06	22840.465970
7	7179.802832	4.090007e+04	232966.47130	1.388670e+06	47618.096630
8	1312.495010	6.928071e+03	44877.67696	2.053994e+05	14207.686240
9	8909.459800	3.844364e+04	495839.09900	1.354685e+06	117128.147700
10	13335.165300	3.412431e+06	327951.42390	1.444560e+06	302614.025100

- Dividing data frame into dependent and independent data frame, x & y respectively.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=5)
```

```
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

```
    X1(Credit Card)  X2(Debit Card)      X3(NEFT)      X4(RTGS)  \
10     13335.165300    3.412431e+06  327951.42390  1.444560e+06
4       3011.659023    2.697110e+04  106103.79310  9.437279e+05
7       7179.802832    4.090007e+04  232966.47130  1.388670e+06
1       1468.181288    1.991353e+04   39919.76326  7.259831e+05
0       1176.915295    1.666356e+04   25887.73695  6.447141e+05
9       8909.459800    3.844364e+04  495839.09900  1.354685e+06
6       5701.087934    3.829477e+04  216347.84720  1.296190e+06
3       2297.109326    2.585332e+04   75985.79897  7.941606e+05

    X5(Mobile Banking)
10       302614.025100
4         10288.908640
7         47618.096630
1           161.934560
0            41.853287
9        117128.147700
6         22840.465970
3          2862.592590
   X1(Credit Card)  X2(Debit Card)      X3(NEFT)      X4(RTGS)  \
5      4335.728709     31609.50258  157997.26160  1.116611e+06
8      1312.495010      6928.07116   44877.67696  2.053994e+05
2      1828.128037     22878.17437   55339.51744  7.439936e+05

   X5(Mobile Banking)
5         15466.69540
8         14207.68624
2           666.24764
       Y(TOTAL)
10  5.500892e+06
4   1.090648e+06
7   1.717419e+06
1   7.910072e+05
0   6.912864e+05
9   2.015006e+06
6   1.579515e+06
3   9.038585e+05
      Y(TOTAL)
5  1.326151e+06
8  2.727253e+05
2  8.284560e+05
```

- Import train_test_split from sklearn.model_selection to divide the x & y data frame into testing set and training set with random state 5 and test data frame size 20 %.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

```
LinearRegression()
```

```
v = pd.DataFrame(regressor.coef_,index=['Co-Efficient']).transpose()
w = pd.DataFrame(x.columns, columns=['Attribute'])
```

```
coeff_df = pd.concat([w,v], axis=1, join='inner')
coeff_df
```

	Attribute	Co-Efficient
0	X1(Credit Card)	2.661491
1	X2(Debit Card)	1.000306
2	X3(NEFT)	0.997612
3	X4(RTGS)	0.985509
4	X5(Mobile Banking)	0.959758

```
y_pred = regressor.predict(x_test)
y_pred = pd.DataFrame(y_pred, columns=['Predicted'])
y_pred
```

	Predicted
0	1.326749e+06
1	2.819486e+05
2	8.275055e+05

```
y_test
```

	Y(TOTAL)
5	1.326151e+06
8	2.727253e+05
2	8.284560e+05

- Performing Linear Regression, calculating the coefficient of Regressors, forecasting and comparing the predicted values with actual values.

Single Smoothing

```
print(f'MSE is : {metrics.mean_squared_error(actual, predicted)}')
print(f'MAD is : {metrics.mean_absolute_error(actual, predicted)}')
print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(actual, predicted))}')
print(f'MAPE is : {metrics.mean_absolute_percentage_error(actual, predicted)}')
print(f'R2 is : {metrics.r2_score(actual, predicted)}')
```

```
MSE is : 5663072395363.485
MAD is : 2328385.637778924
RMSE is : 2379721.0751185706
MAPE is : 2.9448190850023748
R2 is : -2.116447253922005
```

```
vif = 1/(1-(-2.11))
vif
```

```
0.3215434083601286
```

Regression

```
MSE = print(f'MSE is : {metrics.mean_squared_error(y_test, y_pred)}')
MAD = print(f'MAD is : {metrics.mean_absolute_error(y_test, y_pred)}')
RMSE = print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_test, y_pred))}')
MAPE = print(f'MAPE is : {metrics.mean_absolute_percentage_error(y_test, y_pred)}')
R2 = print(f'R2 is : {metrics.r2_score(y_test, y_pred)}')
```

```
MSE is : 28776388.02028868
MAD is : 3590.332900236865
RMSE is : 5364.36277858691
MAPE is : 0.011805531733231076
R2 is : 0.9998445680799171
```

```
vif = 1/(1-(0.99))
vif
```

```
99.99999999999991
```

- Calculating MAD, MSE, RMSE & MAPE to determine the forecast accuracy of the models.
- Calculating R^2 & VIF to check the multicollinearity in the models.

```
model = ols('Y ~ X1 + X2 + X3 + X4 + X5', data = df3).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	Y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	3.252e+06
Date:	Fri, 31 Mar 2023	Prob (F-statistic):	2.85e-16
Time:	20:19:54	Log-Likelihood:	-88.393
No. Observations:	11	AIC:	188.8
Df Residuals:	5	BIC:	191.2
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	251.6883	1346.282	0.187	0.859	-3209.039	3712.415
X1	-1.0355	0.768	-1.349	0.235	-3.008	0.937
X2	0.9965	0.004	228.148	0.000	0.985	1.008
X3	0.9887	0.019	51.868	0.000	0.940	1.038
X4	1.0088	0.004	266.388	0.000	0.999	1.019
X5	1.0987	0.066	16.765	0.000	0.930	1.267

Omnibus:	1.832	Durbin-Watson:	1.622
Prob(Omnibus):	0.400	Jarque-Bera (JB):	0.501
Skew:	-0.518	Prob(JB):	0.778
Kurtosis:	3.148	Cond. No.	5.06e+06

- Durbin-Watson = 1.622, which is positively correlated.

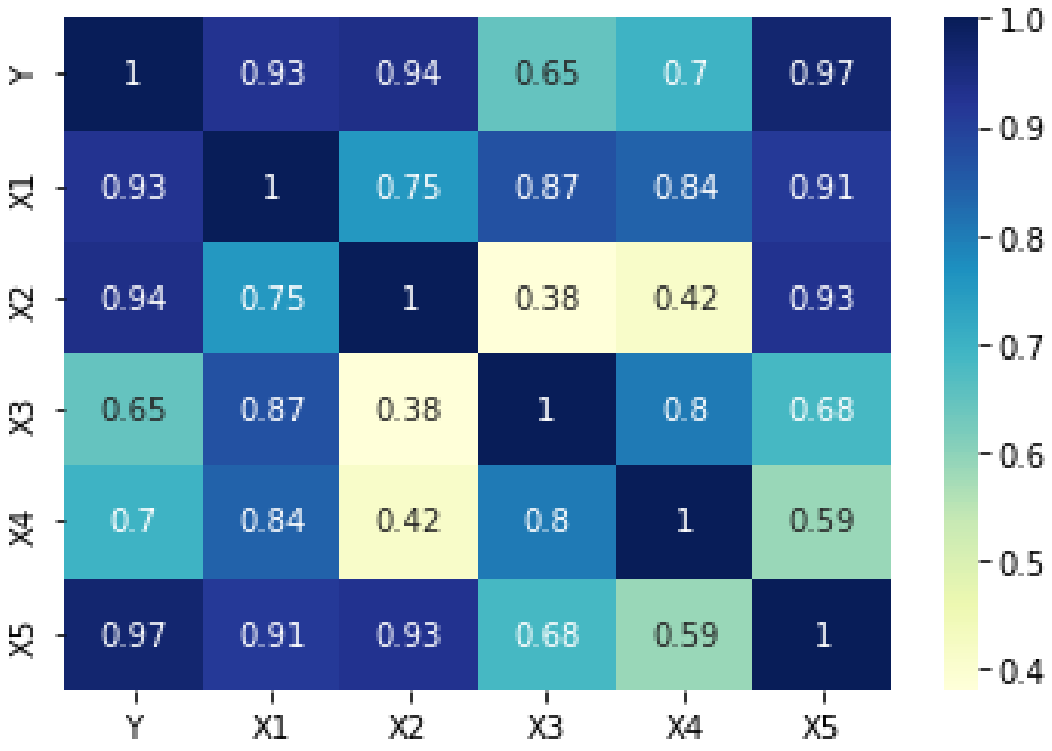


Figure 1: Correlation Matrix

- As seen in the above figure, Y (dependent variable) is high correlated with (independent variables) X1(Credit Card), X2(Debit Card) and X5(Mobile Banking) with correlation value 0.93, 0.94 and 0.97 respectively.

- Therefore, we will drop 1 independent variable at a time & create a regression model for each case and compare them with each other to check multicollinearity and autocorrelation of the cases.

Case 1: Removing X1(Credit Card)

```
df1 = df.drop("X1", axis="columns")
df1
```

	Y	X2	X3	X4	X5
0	6.912864e+05	1.666356e+04	25887.73695	6.447141e+05	41.853287
1	7.910072e+05	1.991353e+04	39919.76326	7.259831e+05	161.934560
2	8.284560e+05	2.287817e+04	55339.51744	7.439936e+05	666.247640
3	9.038585e+05	2.585332e+04	75985.79897	7.941606e+05	2862.592590
4	1.090648e+06	2.697110e+04	106103.79310	9.437279e+05	10288.908640
5	1.326151e+06	3.160950e+04	157997.26160	1.116611e+06	15466.695400
6	1.579515e+06	3.829477e+04	216347.84720	1.296190e+06	22840.465970
7	1.717419e+06	4.090007e+04	232966.47130	1.388670e+06	47618.096630
8	2.727253e+05	6.928071e+03	44877.67696	2.053994e+05	14207.686240
9	2.015006e+06	3.844364e+04	495839.09900	1.354685e+06	117128.147700
10	5.500892e+06	3.412431e+06	327951.42390	1.444560e+06	302614.025100

```
y1 = pd.DataFrame(df1.iloc[:,-5])
y1
```

	Y
0	6.912864e+05
1	7.910072e+05
2	8.284560e+05
3	9.038585e+05
4	1.090648e+06
5	1.326151e+06
6	1.579515e+06
7	1.717419e+06
8	2.727253e+05
9	2.015006e+06
10	5.500892e+06

```
x1 = pd.DataFrame(df1.iloc[:,1:5])
x1
```

	X2	X3	X4	X5
0	1.666356e+04	25887.73695	6.447141e+05	41.853287
1	1.991353e+04	39919.76326	7.259831e+05	161.934560
2	2.287817e+04	55339.51744	7.439936e+05	666.247640
3	2.585332e+04	75985.79897	7.941606e+05	2862.592590
4	2.697110e+04	106103.79310	9.437279e+05	10288.908640
5	3.160950e+04	157997.26160	1.116611e+06	15466.695400
6	3.829477e+04	216347.84720	1.296190e+06	22840.465970
7	4.090007e+04	232966.47130	1.388670e+06	47618.096630
8	6.928071e+03	44877.67696	2.053994e+05	14207.686240
9	3.844364e+04	495839.09900	1.354685e+06	117128.147700
10	3.412431e+06	327951.42390	1.444560e+06	302614.025100

- Dividing data frame into dependent and independent data frame, x1 & y1 respectively.

```
from sklearn.model_selection import train_test_split
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2, random_state=5)
```

```
print(x1_train)
print(x1_test)
print(y1_train)
print(y1_test)
```

```
             X2            X3            X4             X5
10  3.412431e+06  327951.42390  1.444560e+06  302614.025100
4   2.697110e+04  106103.79310  9.437279e+05   10288.908640
7   4.090007e+04  232966.47130  1.388670e+06   47618.096630
1   1.991353e+04   39919.76326  7.259831e+05     161.934560
0   1.666356e+04   25887.73695  6.447141e+05      41.853287
9   3.844364e+04  495839.09900  1.354685e+06  117128.147700
6   3.829477e+04  216347.84720  1.296190e+06   22840.465970
3   2.585332e+04   75985.79897  7.941606e+05    2862.592590
          X2            X3           X4           X5
5  31609.50258  157997.26160  1.116611e+06  15466.69540
8   6928.07116   44877.67696  2.053994e+05  14207.68624
2  22878.17437   55339.51744  7.439936e+05    666.24764
              Y
10  5.500892e+06
4   1.090648e+06
7   1.717419e+06
1   7.910072e+05
0   6.912864e+05
9   2.015006e+06
6   1.579515e+06
3   9.038585e+05
            Y
5  1.326151e+06
8  2.727253e+05
2  8.284560e+05
```

- Import train_test_split from sklearn.model_selection to divide the x1 & y1 data frame into testing set and training set with random state 5 and test data frame size 20 %.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x1_train, y1_train)
```

```
LinearRegression()
```

```
v1 = pd.DataFrame(regressor.coef_,index=['Co-Efficient']).transpose()
w1 = pd.DataFrame(x.columns, columns=['Attribute'])
```

```
coeff_df1 = pd.concat([w1,v1], axis=1, join='inner')
coeff_df1
```

	Attribute	Co-Efficient
0	Y(TOTAL)	0.998103
1	X1(Credit Card)	0.993807
2	X2(Debit Card)	1.002233
3	X3(NEFT)	1.051581

```
y1_pred = regressor.predict(x1_test)
y1_pred = pd.DataFrame(y1_pred, columns=['Predicted'])
y1_pred
```

	Predicted
0	1.326942e+06
1	2.753173e+05
2	8.271915e+05

```
y1_test
```

	Y
5	1.326151e+06
8	2.727253e+05
2	8.284560e+05

- Performing Linear Regression, calculating the coefficient of Regressors, forecasting and comparing the predicted values with actual values.

```
MSE is : 2980728.9972739215
MAD is : 1548.9666378492063
RMSE is : 1726.4787856425926
MAPE is : 0.003875474716086526
R2 is : 0.9999838999796998
```

```
1 vif1 = 1/(1-0.99)
2 vif1
```

```
99.99999999999991
```

- Calculating MAD, MSE, RMSE & MAPE to determine the forecast accuracy of the models.
- Calculating R^2 & VIF to check the multicollinearity in the models.

```
1 model = ols('Y ~ X2 + X3 + X4 + X5', data = df1).fit()
2 print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                3.576e+06
Date:                Sat, 08 Apr 2023   Prob (F-statistic):           2.95e-19
Time:                        03:00:54   Log-Likelihood:                -90.100
No. Observations:                  11   AIC:                             190.2
Df Residuals:                       6   BIC:                             192.2
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1110.9250   1264.613      0.878      0.413   -1983.472    4205.322
X2             0.9970      0.005    214.961      0.000       0.986       1.008
X3             0.9855      0.020     48.860      0.000       0.936       1.035
X4             1.0050      0.003    373.890      0.000       0.998       1.012
X5             1.0659      0.065     16.427      0.000       0.907       1.225
==============================================================================
Omnibus:                        0.413   Durbin-Watson:                   1.330
Prob(Omnibus):                  0.813   Jarque-Bera (JB):                0.478
Skew:                           0.070   Prob(JB):                        0.788
Kurtosis:                       1.989   Cond. No.                     4.46e+06
==============================================================================
```

- Durbin-Watson = 1.33, which is positively correlated.

- Case 2: Removing X2(Debit Card)

```
df2 = df.drop("X2", axis="columns")
df2
```

	Y	X1	X3	X4	X5
0	6.912864e+05	1176.915295	25887.73695	6.447141e+05	41.853287
1	7.910072e+05	1468.181288	39919.76326	7.259831e+05	161.934560
2	8.284560e+05	1828.128037	55339.51744	7.439936e+05	666.247640
3	9.038585e+05	2297.109326	75985.79897	7.941606e+05	2862.592590
4	1.090648e+06	3011.659023	106103.79310	9.437279e+05	10288.908640
5	1.326151e+06	4335.728709	157997.26160	1.116611e+06	15466.695400
6	1.579515e+06	5701.087934	216347.84720	1.296190e+06	22840.465970
7	1.717419e+06	7179.802832	232966.47130	1.388670e+06	47618.096630
8	2.727253e+05	1312.495010	44877.67696	2.053994e+05	14207.686240
9	2.015006e+06	8909.459800	495839.09900	1.354685e+06	117128.147700
10	5.500892e+06	13335.165300	327951.42390	1.444560e+06	302614.025100

```
y2 = pd.DataFrame(df2.iloc[:,-5])
y2
```

	Y
0	6.912864e+05
1	7.910072e+05
2	8.284560e+05
3	9.038585e+05
4	1.090648e+06
5	1.326151e+06
6	1.579515e+06
7	1.717419e+06
8	2.727253e+05
9	2.015006e+06
10	5.500892e+06

```
x2 = pd.DataFrame(df2.iloc[:,1:5])
x2
```

	X1	X3	X4	X5
0	1176.915295	25887.73695	6.447141e+05	41.853287
1	1468.181288	39919.76326	7.259831e+05	161.934560
2	1828.128037	55339.51744	7.439936e+05	666.247640
3	2297.109326	75985.79897	7.941606e+05	2862.592590
4	3011.659023	106103.79310	9.437279e+05	10288.908640
5	4335.728709	157997.26160	1.116611e+06	15466.695400
6	5701.087934	216347.84720	1.296190e+06	22840.465970
7	7179.802832	232966.47130	1.388670e+06	47618.096630
8	1312.495010	44877.67696	2.053994e+05	14207.686240
9	8909.459800	495839.09900	1.354685e+06	117128.147700
10	13335.165300	327951.42390	1.444560e+06	302614.025100

- Dividing data frame into dependent and independent data frame, x2 & y2 respectively.

```
from sklearn.model_selection import train_test_split
x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size=0.2, random_state=5)

print(x2_train)
print(x2_test)
print(y2_train)
print(y2_test)
```

```
             X1            X3            X4            X5
10  13335.165300  327951.42390  1.444560e+06  302614.025100
4    3011.659023  106103.79310  9.437279e+05   10288.908640
7    7179.802832  232966.47130  1.388670e+06   47618.096630
1    1468.181288   39919.76326  7.259831e+05     161.934560
0    1176.915295   25887.73695  6.447141e+05      41.853287
9    8909.459800  495839.09900  1.354685e+06  117128.147700
6    5701.087934  216347.84720  1.296190e+06   22840.465970
3    2297.109326   75985.79897  7.941606e+05    2862.592590
            X1            X3            X4           X5
5  4335.728709  157997.26160  1.116611e+06  15466.69540
8  1312.495010   44877.67696  2.053994e+05  14207.68624
2  1828.128037   55339.51744  7.439936e+05    666.24764
              Y
10  5.500892e+06
4   1.090648e+06
7   1.717419e+06
1   7.910072e+05
0   6.912864e+05
9   2.015006e+06
6   1.579515e+06
3   9.038585e+05
              Y
5  1.326151e+06
8  2.727253e+05
2  8.284560e+05
```

- Import train_test_split from sklearn.model_selection to divide the x2 & y2 data frame into testing set and training set with random state 5 and test data frame size 20 %.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x2_train, y2_train)

v2 = pd.DataFrame(regressor.coef_,index=['Co-Efficient']).transpose()
w2 = pd.DataFrame(x2.columns, columns=['Attribute'])

coeff_df2 = pd.concat([w2,v2], axis=1, join='inner')
coeff_df2
```

	Attribute	Co-Efficient
0	X1	-291.595229
1	X3	-2.609258
2	X4	3.178305
3	X5	21.705794

```
y2_pred = regressor.predict(x2_test)
y2_pred = pd.DataFrame(y2_pred, columns=['Predicted'])
y2_pred
```

	Predicted
0	1.280587e+06
1	-4.661308e+05
2	7.741045e+05

```
y2_test
```

	Y
5	1.326151e+06
8	2.727253e+05
2	8.284560e+05

- Performing Linear Regression, calculating the coefficient of Regressors, forecasting, and comparing the predicted values with actual values.

```
MSE is : 183646188733.879
MAD is : 279590.55718247994
RMSE is : 428539.5999600025
MAPE is : 0.9363740612180219
R2 is : 0.008058978399435479
```

```
1 vif1 = 1/(1-0.00805)
2 vif1
```

```
1.0081153283935682
```

- Calculating MAD, MSE, RMSE & MAPE to determine the forecast accuracy of the models.
- Calculating R^2 & VIF to check the multicollinearity in the models.

```
1 model = ols('Y ~ X1 + X3 + X4 + X5', data = df2).fit()
2 print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.997
Model:                            OLS   Adj. R-squared:                  0.995
Method:                 Least Squares   F-statistic:                     467.0
Date:                Fri, 31 Mar 2023   Prob (F-statistic):           1.31e-07
Time:                        20:20:10   Log-Likelihood:                -139.27
No. Observations:                  11   AIC:                             288.5
Df Residuals:                       6   BIC:                             290.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -1.179e+05   1.16e+05     -1.019      0.348   -4.01e+05    1.65e+05
X1           -16.6848     71.204     -0.234      0.823    -190.915     157.545
X3            -3.1417      0.556     -5.655      0.001      -4.501      -1.782
X4             1.5237      0.283      5.379      0.002       0.831       2.217
X5            15.3714      1.820      8.447      0.000      10.919      19.824
==============================================================================
Omnibus:                        1.222   Durbin-Watson:                   2.356
Prob(Omnibus):                  0.543   Jarque-Bera (JB):                0.105
Skew:                          -0.216   Prob(JB):                        0.949
Kurtosis:                       3.207   Cond. No.                     3.93e+06
==============================================================================
```

- Durbin-Watson = 2.35.

Case 3: Removing X5(Mobile Banking)

```
df5 = df.drop("X5", axis="columns")
df5
```

	Y	X1	X2	X3	X4
0	6.912864e+05	1176.915295	1.666356e+04	25887.73695	6.447141e+05
1	7.910072e+05	1468.181288	1.991353e+04	39919.76326	7.259831e+05
2	8.284560e+05	1828.128037	2.287817e+04	55339.51744	7.439936e+05
3	9.038585e+05	2297.109326	2.585332e+04	75985.79897	7.941606e+05
4	1.090648e+06	3011.659023	2.697110e+04	106103.79310	9.437279e+05
5	1.326151e+06	4335.728709	3.160950e+04	157997.26160	1.116611e+06
6	1.579515e+06	5701.087934	3.829477e+04	216347.84720	1.296190e+06
7	1.717419e+06	7179.802832	4.090007e+04	232966.47130	1.388670e+06
8	2.727253e+05	1312.495010	6.928071e+03	44877.67696	2.053994e+05
9	2.015006e+06	8909.459800	3.844364e+04	495839.09900	1.354685e+06
10	5.500892e+06	13335.165300	3.412431e+06	327951.42390	1.444560e+06

```
y5 = pd.DataFrame(df5.iloc[:,-5])
y5
```

	Y
0	6.912864e+05
1	7.910072e+05
2	8.284560e+05
3	9.038585e+05
4	1.090648e+06
5	1.326151e+06
6	1.579515e+06
7	1.717419e+06
8	2.727253e+05
9	2.015006e+06
10	5.500892e+06

```
x5 = pd.DataFrame(df5.iloc[:,1:5])
x5
```

	X1	X2	X3	X4
0	1176.915295	1.666356e+04	25887.73695	6.447141e+05
1	1468.181288	1.991353e+04	39919.76326	7.259831e+05
2	1828.128037	2.287817e+04	55339.51744	7.439936e+05
3	2297.109326	2.585332e+04	75985.79897	7.941606e+05
4	3011.659023	2.697110e+04	106103.79310	9.437279e+05
5	4335.728709	3.160950e+04	157997.26160	1.116611e+06
6	5701.087934	3.829477e+04	216347.84720	1.296190e+06
7	7179.802832	4.090007e+04	232966.47130	1.388670e+06
8	1312.495010	6.928071e+03	44877.67696	2.053994e+05
9	8909.459800	3.844364e+04	495839.09900	1.354685e+06
10	13335.165300	3.412431e+06	327951.42390	1.444560e+06

- Dividing data frame into dependent and independent data frame, x5 & y5 respectively.

```
from sklearn.model_selection import train_test_split
x5_train, x5_test, y5_train, y5_test = train_test_split(x5, y5, test_size=0.2, random_state=5)

print(x5_train)
print(x5_test)
print(y5_train)
print(y5_test)
```

```
             X1            X2            X3            X4
10  13335.165300  3.412431e+06  327951.42390  1.444560e+06
4    3011.659023  2.697110e+04  106103.79310  9.437279e+05
7    7179.802832  4.090007e+04  232966.47130  1.388670e+06
1    1468.181288  1.991353e+04   39919.76326  7.259831e+05
0    1176.915295  1.666356e+04   25887.73695  6.447141e+05
9    8909.459800  3.844364e+04  495839.09900  1.354685e+06
6    5701.087934  3.829477e+04  216347.84720  1.296190e+06
3    2297.109326  2.585332e+04   75985.79897  7.941606e+05
           X1           X2            X3            X4
5  4335.728709  31609.50258  157997.26160  1.116611e+06
8  1312.495010   6928.07116   44877.67696  2.053994e+05
2  1828.128037  22878.17437   55339.51744  7.439936e+05
              Y
10  5.500892e+06
4   1.090648e+06
7   1.717419e+06
1   7.910072e+05
0   6.912864e+05
9   2.015006e+06
6   1.579515e+06
3   9.038585e+05
              Y
5  1.326151e+06
8  2.727253e+05
2  8.284560e+05
```

- Import train_test_split from sklearn.model_selection to divide the x5 & y5 data frame into testing set and training set with random state 5 and test data frame size 20 %.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x5_train, y5_train)

v5 = pd.DataFrame(regressor.coef_,index=['Co-Efficient']).transpose()
w5 = pd.DataFrame(x5.columns, columns=['Attribute'])

coeff_df5 = pd.concat([w5,v5], axis=1, join='inner')
coeff_df5
```

	Attribute	Co-Efficient
0	X1	23.222816
1	X2	1.035074
2	X3	1.102146
3	X4	0.848467

```
y5_pred = regressor.predict(x5_test)
y5_pred = pd.DataFrame(y5_pred, columns=['Predicted'])
y5_pred
```

	Predicted
0	1.326926e+06
1	3.333629e+05
2	8.303570e+05

```
y5_test
```

	Y
5	1.326151e+06
8	2.727253e+05
2	8.284560e+05

- Performing Linear Regression, calculating the coefficient of Regressors, forecasting and comparing the predicted values with actual values.

```
MSE is : 1227041519.5689313
MAD is : 21104.274172055495
RMSE is : 35029.15242435836
MAPE is : 0.075072565544043
R2 is : 0.9933722946996124
```

```
1 vif1 = 1/(1-0.99337)
2 vif1
```

```
150.8295625942679
```

- Calculating MAD, MSE, RMSE & MAPE to determine the forecast accuracy of the models.
- Calculating R^2 & VIF to check the multicollinearity in the models.

```
1 model = ols('Y ~ X1 + X2 + X3 + X4', data = df5).fit()
2 print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 8.526e+04
Date:                Sat, 08 Apr 2023   Prob (F-statistic):           2.18e-14
Time:                        03:02:13   Log-Likelihood:                -110.65
No. Observations:                  11   AIC:                             231.3
Df Residuals:                       6   BIC:                             233.3
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1.114e+04   8142.377      1.368      0.220   -8783.471    3.11e+04
X1             3.7388      4.921      0.760      0.476      -8.303      15.780
X2             1.0664      0.009    118.624      0.000       1.044       1.088
X3             1.2620      0.068     18.510      0.000       1.095       1.429
X4             0.9602      0.017     57.086      0.000       0.919       1.001
==============================================================================
Omnibus:                        0.603   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.740   Jarque-Bera (JB):                0.494
Skew:                          -0.424   Prob(JB):                        0.781
Kurtosis:                       2.401   Cond. No.                     4.42e+06
==============================================================================
```

- Durbin-Watson = 2.03.

7 RESULTS & DISCUSSION

7.1 Summary Statistics

- We start with summary statistics of our data, calculating mean, median, mode, standard deviation & quartiles of each parameter (dependent & independent variables).

Year	Y(TOTAL)	X1(Credit Card)	X2(Debit Card)	X3(NEFT)	X4(RTGS)	X5(Mobile Banking)
2012	691286.4044	1176.915295	16663.55518	25887.73695	644714.0835	41.853287
2013	791007.1656	1468.181288	19913.53303	39919.76326	725983.1333	161.93456
2014	828455.9973	1828.128037	22878.17437	55339.51744	743993.6193	666.24764
2015	903858.5047	2297.109326	25853.32103	75985.79897	794160.6312	2862.59259
2016	1090648.495	3011.659023	26971.10334	106103.7931	943727.941	10288.90864
2017	1326151.464	4335.728709	31609.50258	157997.2616	1116611.165	15466.6954
2018	1579514.564	5701.087934	38294.76715	216347.8472	1296189.968	22840.46597
2019	1717419.19	7179.802832	40900.06691	232966.4713	1388669.596	47618.09663
2020	272725.3454	1312.49501	6928.07116	44877.67696	205399.416	14207.68624
2021	2015005.69	8909.4598	38443.63987	495839.099	1354685.344	117128.1477
2022	5500892.006	13335.1653	3412431.187	327951.4239	1444560.204	302614.0251
COUNT	11	11	11	11	11	11
MINIMUM	272725.3454	1176.915295	6928.07116	25887.73695	205399.416	41.853287
MAXIMUM	5500892.006	13335.1653	3412431.187	495839.099	1444560.204	302614.0251
MEAN	1519724.075	4595.975687	334626.0838	161746.9445	968972.2819	48536.05943
MEDIAN	1090648.495	3011.659023	26971.10334	106103.7931	943727.941	14207.68624
MODE	0	0	0	0	0	0
QUARTILE 1	809731.5815	1648.154663	21395.8537	50108.5972	734988.3763	1764.420115
QUARTILE 2	1090648.495	3011.659023	26971.10334	106103.7931	943727.941	14207.68624
QUARTILE 3	1648466.877	6440.445383	38369.20351	224657.1593	1325437.656	35229.2813
QUARTILE DEVIATION	419367.6478	2396.14536	8486.674905	87274.28103	295224.6399	16732.43059
COEFFICIENT OF QUARTILE DEVIATION	0.341199179	0.59247468	0.284001231	0.635263158	0.286566599	0.904609701
VARIANCE	1.81716E+12	13695898.92	9.47384E+11	19640065758	1.37183E+11	7520526038
STANDARD DEVIATION	1348019.474	3700.797066	973336.5145	140143.0189	370382.5225	86720.96654

Table 2: Summary Statistics

- We observe that mode value of each variable is 0, meaning that there is no repeated frequency value in the data.

7.2 Single Exponential Smoothing vs Regression Model

FA	SES	RE
MAD	2328385.638	3590.3329
MSE	5.66307E+12	28776388.02
RMSE	2379721.075	5364.362779
MAPE	2.944819085	0.011805532
R2	-2.116447254	0.999844568
VIF	0.321543408	99.99

Table 3: Smoothing Model vs Regression Model

- By performing Forecast Accuracy on both the models, we observe that MAD, MSE, RMSE & MAPE of Single Exponential Smoothing Model has higher error values comparing to Regression Model.
- But R^2 & VIF is better in Single Exponential Smoothing Model.
- Since, Regression model has less errors present i.e., forecast accuracy of regression model is better than the Single Exponential Smoothing model, that's why we select regression model for further analysis.
- VIF = 99.99 which far exceeds the benchmark value of 10, which suggest that multicollinearity is a serious issue in our dataset. As such we try to reduce the multicollinearity with variable selection technique of reducing some of the independent variable from the model.

FA	RE	Case 1 (X1)	Case 2 (X2)	Case 3 (X5)
MAD	3590.3329	1548.966638	279590.5572	21104.2742
MSE	28776388.02	2980728.997	1.83646E+11	1227041520
RMSE	5364.362779	1726.478786	428539.6	35029.1524
MAPE	0.011805532	0.003875475	0.936374061	0.07507257
R2	0.999844568	0.9999839	0.008058978	0.99337229
VIF	99.99	99.99	1.008064516	99.99
D-W	1.622028946	1.329576976	2.355920513	2.03755551
AIC	188.8	190.2	288.5	231.3
BIC	191.2	192.2	290.5	233.3
SE	1346.282	1264.613	1.16E+05	8142.377
P-Value	0.859	0.413	0.348	0.22

Table 4: Regression Model Comparison

- As per the above table, Case 1 & Case 3 have VIF = 99.99 and Case 2 has VIF = 1.00, which is best case scenario where we can state that in Case 2 – Regression Model multicollinearity has been fixed.
- Further comparing the cases, Case 2 & Case 3 have Durbin-Watson test value = 2.35 & 2.03 respectively which falls under acceptable range of the Durbin-Watson Test i.e., 1.50 – 2.50. Therefore, we can say that Case 2 & 3 regression model do not have autocorrelation problem, whereas Case 1 with D-W value = 1.32 is slightly positively correlated and has autocorrelation issue.
- We can conclude that Case 2 – regression model is the best option to select for forecasting i.e., dropping the independent variable X2 (Debit Card Transactions).

7.3 Consistency Ranking

- Calculating CV for various banking transactions and determining their consistency ranking.

For Inward (Credit) Transactions of NEFT & RTGS –

YEAR	NEFTI	NEFTI RANK	RTGSI	RTGSI RANK
2012	3.036684174	**1**	2.702773864	**1**
2013	3.20784494	**4**	2.990800678	**2**
2014	3.173545635	**3**	3.13758987	**3**
2015	3.153328082	**2**	3.376328391	**4**
2016	3.436874864	**5**	3.621761476	**5**
2017	3.681728547	**6**	4.061753232	**6**
2018	3.73985212	**7**	4.228571346	**11**
2019	3.900404322	**8**	4.212231707	**10**
2020	4.003576769	**9**	4.091060886	**7**
2021	4.28570722	**11**	4.165210061	**8**
2022	4.254734877	**10**	4.18398925	**9**

Table 5: NEFT & RTGS Inward - CV

- We observe that, for NEFT Inward Transactions Year 2012 has the least CV score and Year 2021 has the highest CV score. Therefore, Inward NEFT transaction were less volatile in the Year 2012 and its volatility was high during the Year 2021.
- Similarly, for RTGS Inward Transactions Year 2012 has the least CV score and Year 2018 has the highest CV score. Therefore, Inward RTGS transaction were less volatile in the Year 2012 and its volatility was high during the Year 2018.

For Outward (Debit) Transactions of NEFT & RTGS –

YEAR	NEFTO	NEFTO RANK	RTGSO	RTGSO RANK
2012	2.941684486	**1**	2.742948037	**1**
2013	3.203808049	**2**	3.015161115	**2**
2014	3.289273582	**3**	3.172433939	**3**
2015	3.372349057	**4**	3.416407227	**4**
2016	3.388620842	**5**	3.670305462	**5**
2017	3.534470236	**6**	4.178007056	**6**
2018	3.872338707	**7**	4.324517709	**8**
2019	4.111232539	**9**	4.377560208	**9**
2020	4.179984138	**10**	4.180526478	**7**
2021	4.198618738	**11**	4.396339669	**10**
2022	4.096842494	**8**	4.419770077	**11**

Table 6: NEFT & RTGS Outward - CV

- We observe that, for NEFT Outward Transactions Year 2012 has the least CV score and Year 2021 has the highest CV score. Therefore, Outward NEFT transaction were less volatile in the Year 2012 and its volatility was high during the Year 2021.
- Similarly, for RTGS Outward Transaction Year 2012 has the least CV score and Year 2022 has the highest CV score. Therefore, Outward RTGS transaction were less volatile in the Year 2012 and its volatility was high during the Year 2022.

- So, with these observations we can conclude that the volatility of both Credit & Debit Transactions of NEFT was least in Year 2012 & highest in Year 2021.
- But, in respect to the RTGS transactions both Credit & Debit transactions had their least volatility in the Year 2012, whereas the Credit RTGS transactions had the highest volatility in the Year 2018 and Debit RTGS transactions were highest in the Year 2022.

For Credit Card & Debit Card Transactions –

YEAR	CC	CC RANK	DC	DC RANK
2012	2.742647298	**1**	2.605943758	**5**
2013	2.754597162	**2**	2.57594279	**4**
2014	2.959391967	**7**	2.11547537	**1**
2015	2.902967641	**5**	2.468206027	**3**
2016	2.800244258	**4**	2.387459493	**2**
2017	2.780515958	**3**	2.909102442	**9**
2018	3.019644	**9**	3.178152907	**11**
2019	3.020364765	**10**	3.028953203	**10**
2020	3.10617408	**11**	2.908266758	**8**
2021	2.904455932	**6**	2.704433508	**6**
2022	2.9648493	**8**	2.751686644	**7**

Table 7: Credit Card & Debit Card - C

- We observe that, for Credit Card Transactions Year 2012 has the least CV score and 2020 has the highest CV score. Therefore, Credit Card transactions were least volatile during the Year 2012 and highly volatile during the Year 2020.
- Similarly, for Debit Card Transactions Year 2014 has the least CV score and 2018 has the highest CV score. Therefore, Debit Card transactions were least volatile during the Year 2014 and highly during the Year 2018.

- Now, if we observe Inward RTGS Transactions & Debit Card Transactions, we see that both have their highest volatility period in the Year 2018.

For Mobile Banking Transactions –

YEAR	MB	MB RANK
2012	3.809090648	**4**
2013	3.203510315	**1**
2014	3.377466784	**2**
2015	3.525665917	**3**
2016	4.369097464	**5**
2017	5.042074361	**7**
2018	4.509155112	**6**
2019	6.201520555	**8**
2020	6.290697849	**9**
2021	6.690129456	**10**
2022	7.023819745	**11**

Table 8: Mobile Banking – CV

- We observe that, for Mobile Banking Transactions Year 2013 has the least CV score and 2022 has the highest CV score. Therefore, Mobile Banking transactions were least volatile during the Year 2013 and highly volatile during the Year 2022.

- As we were able to find the least & most volatile banking transactions during the years 2012-2022.
- We know try to see what the overall volatile transaction mode and consistency would be over the last decade for all the banking transactions.

For All Transactions –

TRANSACTIONS	CV	RANK
X1(Credit Card)	0.8052256	2
X2(Debit Card)	2.9087288	5
X3(NEFT)	0.8664338	3
X4(RTGS)	0.3822426	1
X5(Mobile Banking)	1.7867327	4

Table 9: All Transactions - CV

- From the above table we can conclude that RTGS Transactions has been the least volatile mode of transaction and Debit Card Transaction has been the most volatile mode of transaction over the last decade

7.4 Chi-Square Test

O1	O2		E1	E2	((O1-E1)^2)/E1	((O2-E2)^2)/E2	((O1-E1)^2)/E1	((O2-E2)^2)/E2
484	3	487	270.825	216.175	167.7957529	210.2161351	1590.894529	1960.677576
5791	9851	15642	8698.67	6943.33	971.9355154	1217.650173	4493.092573	31608.00613
407	5	412	229.117	182.883	138.105133	173.0194406	1717.048699	1956.685712
8784	2672	11456	6370.79	5085.21	914.1027836	1145.19677	16212.5442	252.9582584
5311	2855	8166	4541.19	3624.81	130.494945	163.4853238	3290.203785	401.2486361
1817	435	2252	1252.36	999.641	254.5747055	318.9336426	169.8060804	1192.888792
8108	4039	12147	6755.07	5391.93	270.9713419	339.4755066	12929.86693	2183.656965
168	2	170	94.5387	75.4613	57.0831599	71.51433244	2139.271401	1962.675034
1248	324	1572	874.205	697.795	159.8286279	200.2348442	599.9671022	1372.050455
47	432	479	266.377	212.623	180.669389	226.3443504	2370.712577	1197.566254
1115	564	1679	933.708	745.292	35.20009854	44.09902243	738.4092813	1000.416208
459	53	512	284.728	227.272	106.6653747	133.6314086	1631.326038	1862.1013
427	499	926	514.958	411.042	15.02368192	18.82181339	1683.818685	1095.28327
378	7	385	214.102	170.898	125.4655332	157.1844282	1765.808967	1952.697915
13	18	31	17.2394	13.7606	1.042527734	1.306088785	2437.884593	1930.837745
223	42	265	147.369	117.631	38.81431714	48.62695029	2037.999731	1883.569946
344	3	347	192.97	154.03	118.2049595	148.0883116	1823.845561	1960.677576
486	2	488	271.382	216.618	169.727988	212.6368579	1587.681925	1962.675034
1544	350	1894	1053.27	840.728	228.6341018	286.4350045	343.3947478	1328.960935
136	1	137	76.187	60.813	46.95796769	58.82939414	2199.323054	1964.673508
9588	9419	19007	10570	8437.02	91.22861551	114.292088	20599.75163	28239.15197
46	28	74	41.1521	32.8479	0.571096886	0.715475678	2372.67483	1911.071643
8805	7740	16545	9200.84	7344.16	17.02968105	21.33494842	16320.46164	16948.06643
2224	1119	3343	1859.08	1483.92	71.63243385	89.74180296	23.34253607	365.3629835
628	239	867	482.147	384.853	44.12143274	55.27575583	1367.886395	1517.717483
4358	611	4969	2763.31	2205.69	920.2864654	1152.943746	1456.250396	934.496626
2708	746	3454	1920.8	1533.2	322.6141895	404.1741634	24.20060015	757.646326
1649	1247	2896	1610.49	1285.51	0.920650301	1.153399563	269.4702502	263.3529961
858	627	1485	825.823	659.177	1.253719963	1.570672443	1046.606169	912.568458
59	2	61	33.9227	27.0773	18.53835067	23.22502425	2347.228849	1962.675034
869	594	1463	813.589	649.411	3.773910105	4.727990922	1032.316567	958.0805578
1382	851	2233	1241.79	991.207	15.83026647	19.83231028	475.005381	632.9096108
1485	1329	2814	1564.89	1249.11	4.078816061	5.109980041	388.8605163	206.7587519
9	1	10	5.5611	4.4389	2.126566063	2.664182443	2445.848876	1964.673508
1	62	63	35.0349	27.9651	33.06346363	41.42222564	2461.816406	1844.62757
787	22	809	449.893	359.107	252.5961721	316.4549169	1141.202061	1922.919101
8150	4807	12957	7205.52	5751.48	123.8011724	155.099301	13123.01263	4102.083807
373	40	413	229.673	183.327	89.44233841	112.0542229	1774.284908	1887.486557
31	1	32	17.7955	14.2045	9.797885282	12.27488504	2402.206045	1964.673508
857	420	1277	710.152	566.848	30.36567386	38.04240864	1047.910095	1216.367626
99	73	172	95.6509	76.3491	0.117265054	0.146910789	2269.793976	1823.382652
25686	39964	65650	36508.6	29141.4	3208.254715	4019.332402	218875.8534	734131.632
482	666	1148	638.414	509.586	38.32210702	48.01030472	1594.110379	860.2092228
2194	216	2410	1340.22	1069.78	543.888025	681.3881554	29.54793453	1558.396313
508	3	511	284.172	226.828	176.2977694	220.8675433	1552.557588	1960.677576
8042	3190	11232	6246.23	4985.77	516.2805255	646.8012142	12629.24534	760.9444727
231	1	232	129.017	102.983	80.61258653	100.9922247	2023.473867	1964.673508
255	8	263	146.257	116.743	80.85132706	101.2913211	1980.207987	1950.705542
1073	184	1257	699.03	557.97	200.0679245	250.6470225	785.1110415	1615.88786
120727	96367	217094	120728	96365.9	11029.06305	13817.3164	375623.1388	878169.507
					Test of Independence	**24846.37945**	**Goodness of Fit**	**1253792.646**

Table 10: Chi-Square Test

- We take new dataset of Total ATMs for all the banks available in India.
- This dataset has 2 frequencies of ATMs namely, On-site ATMs(O1) & Off-site ATMs(O2).

- For the Goodness of Fit Test –

  H0: There is no significant difference between observed frequencies and expected frequencies.
  H1: There is a significant difference between observed frequencies and expected frequencies.

  We observe that calculated chi-square value = 1253792.65, which is way more than the tabulated value of chi-square = 65.171 with degree of freedom = 48.

  Therefore, we reject our Null Hypothesis i.e., Reject H0.
  There is a significant difference between observed frequencies and expected frequencies.

- For the Test of Independence –

  H0: Observed frequencies and Expected frequencies are Independent.
  H1: Observed frequencies and Expected frequencies are not Independent.

  We observe that calculated chi-square value = 24856.37, which is way more than the tabulated value of chi-square = 65.171 with degree of freedom = 48.

  Therefore, we reject our Null Hypothesis i.e., Reject H0.
  Observed frequencies and Expected frequencies are not Independent.

- As per our findings, we observe that there is significant difference or gap between the actual & expected level of transactions & its mechanisms.
- This further indicate that there is a urgent need of improving such vital banking services by investing more in banking infrastructure across the country.

7.5 Hotspot

- We perform hotspot analysis on No. of ATMs in India on a State & Union Territory Level.

For Year 2012:

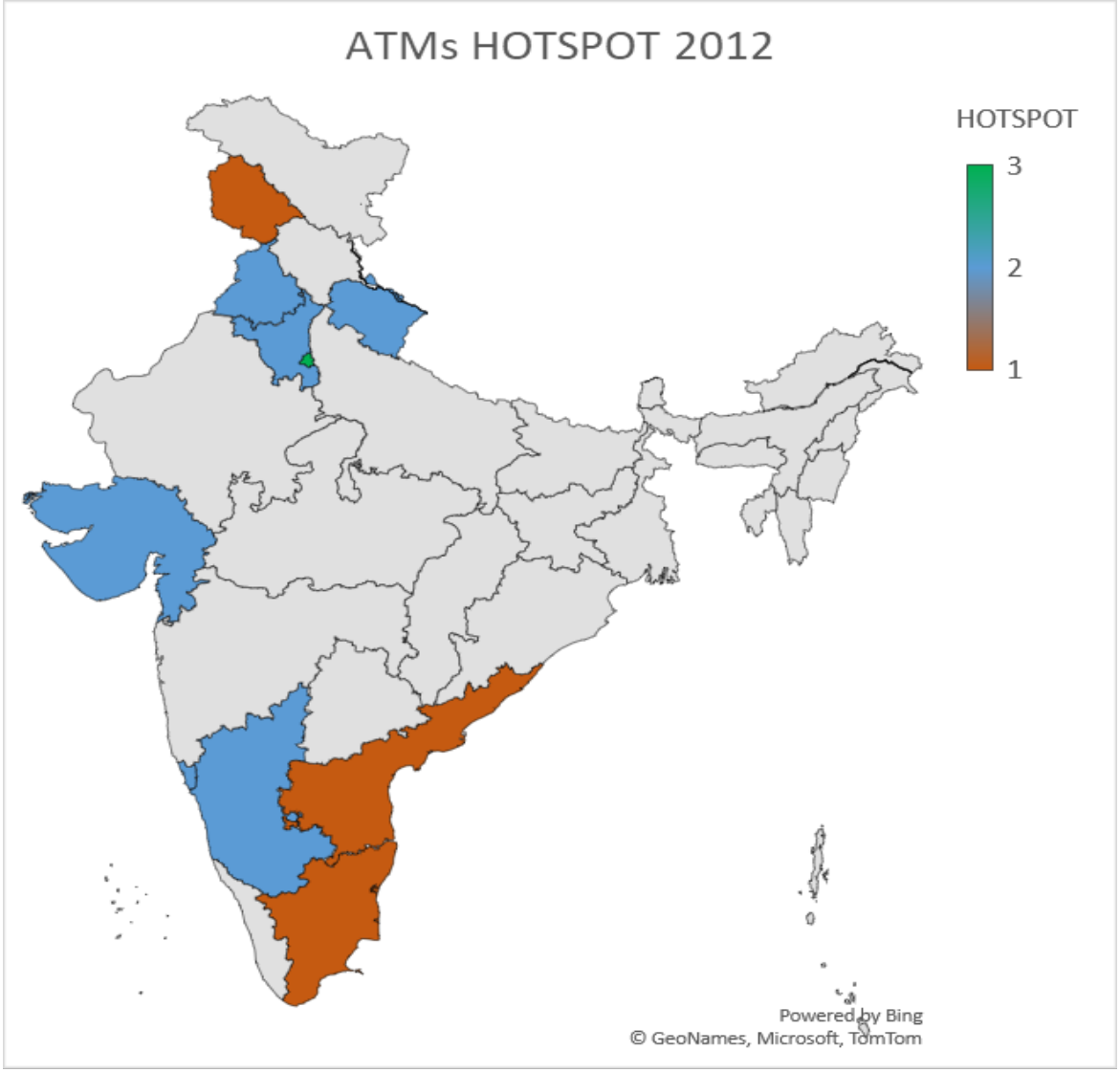


Figure 2: ATMs Hotspot 2012

- Primary Hotspots (Green): Delhi.
- Secondary Hotspots (Blue): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand
- Tertiary Hotspots (Red): Jammu and Kashmir, Puducherry, Tamil Nadu, Andhra Pradesh

Statistics	Primary	Secondary	Tertiary
Number of Cases	6469	25421	22622
Expected Cases	1428.22	15819.12	14210.73
Annual Cases / 100000	33.4	11.9	11.8
Relative Risk	4.76	1.8	1.75
Log Likelihood Ratio	4855.013945	2988.400826	2504.981071
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 4855.013945, 12988.400826 & 2504.981071 respectively.
- Also, the Relative Risk value is also in decreasing order from with values 4.76, 1.8 & 1.75 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2013:

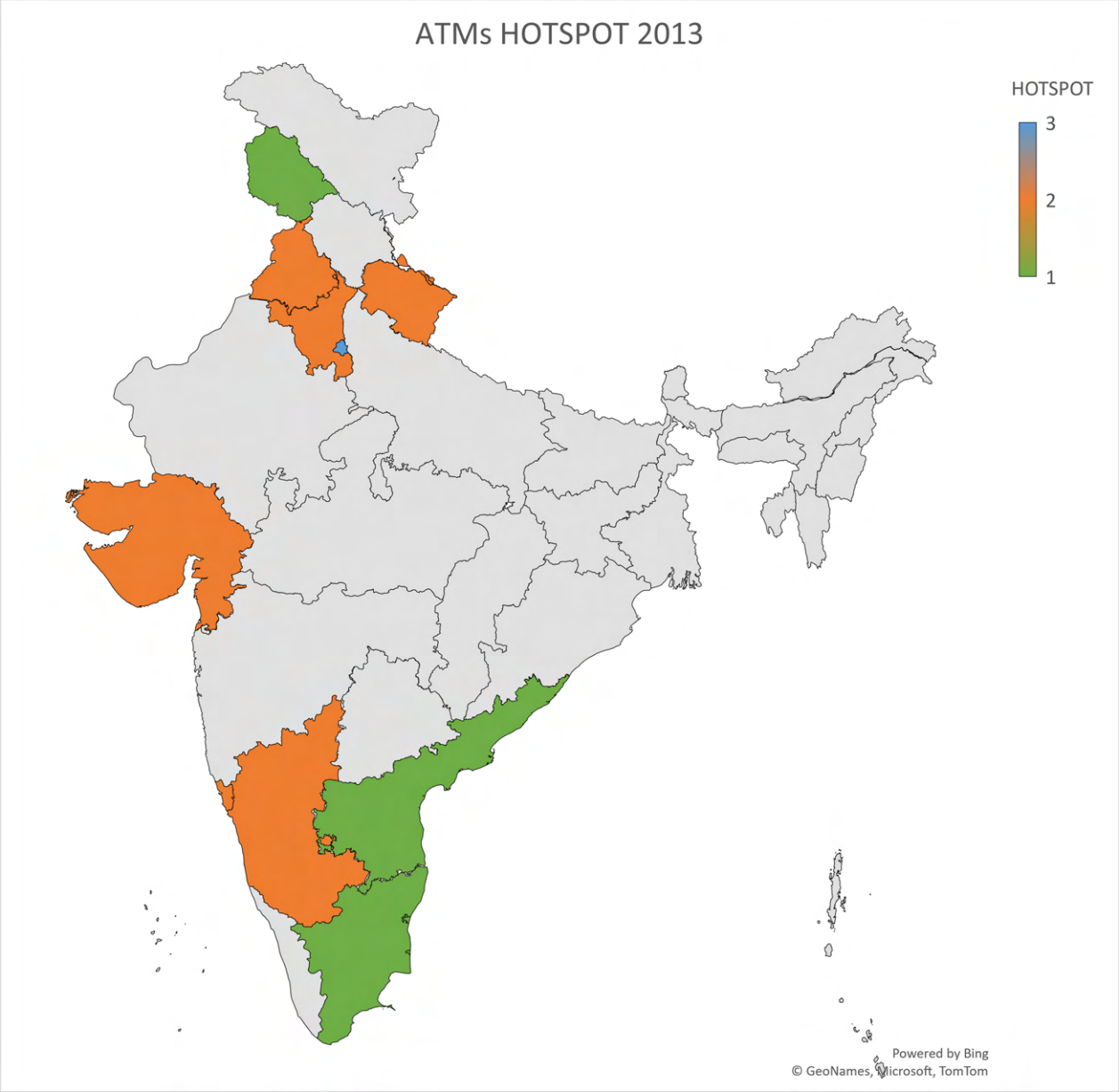


Figure 3: ATMs Hotspot 2013

- Primary Hotspots (Blue): Delhi.
- Secondary Hotspots (Orange): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand
- Tertiary Hotspots (Green): Jammu and Kashmir, Puducherry, Tamil Nadu, Andhra Pradesh

Statistics	Primary	Secondary	Tertiary
Number of Cases	7719	33137	29465
Expected Cases	1910.64	21162.54	19010.86
Annual Cases / 100000	39.9	15.5	15.3
Relative Risk	4.22	1.74	1.69
Log Likelihood Ratio	5091.803692	3501.255488	2916.524569
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 5091.803692, 3501.255488 & 2916.524569 respectively.
- Also, the Relative Risk value is also in decreasing order from with values 4.22, 1.74 & 1.69 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2014:

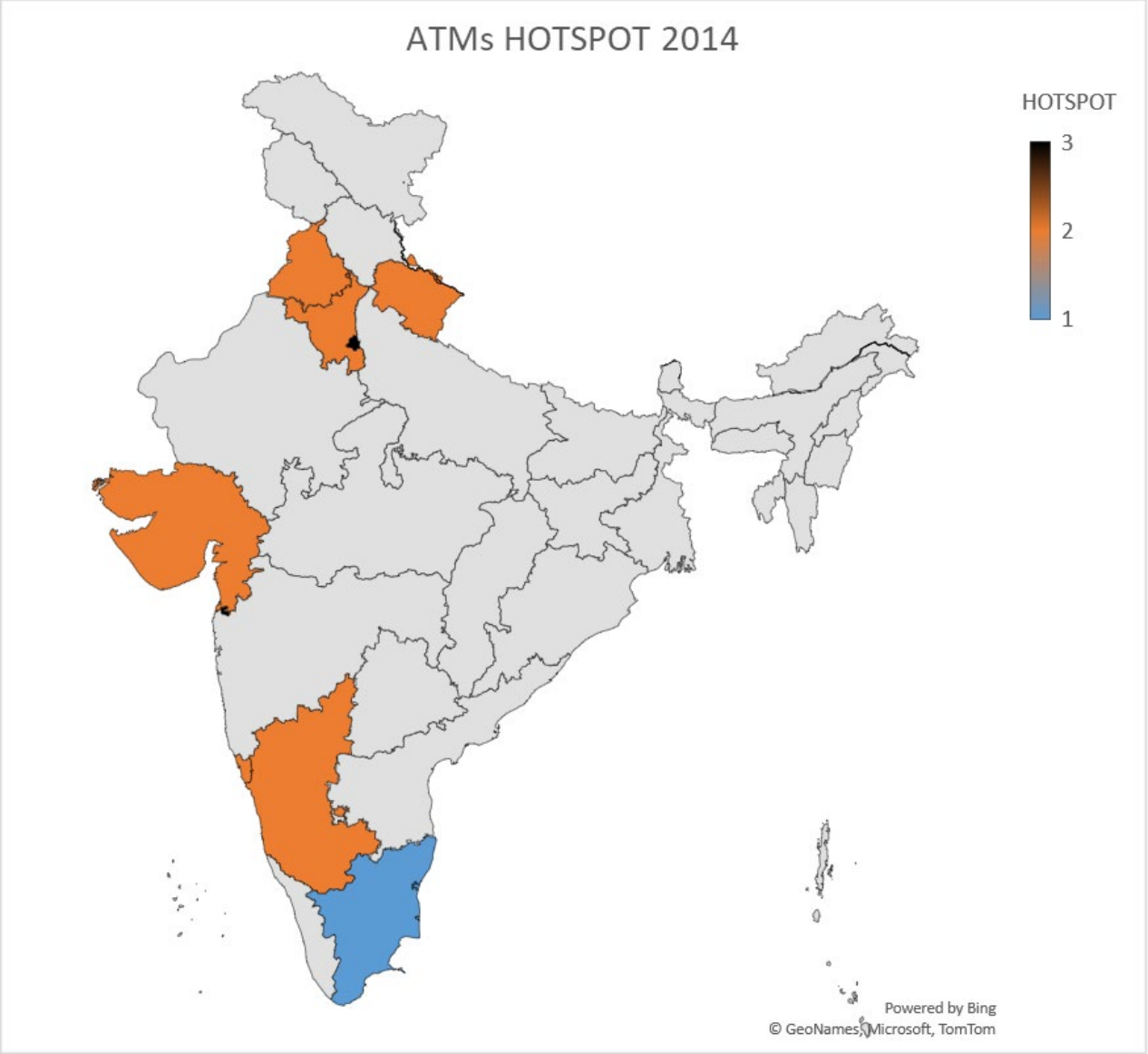


Figure 4:ATMs Hotspot 2014

- Primary Hotspots (Green): Delhi.
- Secondary Hotspots (Orange): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.
- Tertiary Hotspots (Blue): Puducherry, Tamil Nadu.

Statistics	Primary	Secondary	Tertiary
Number of Cases	8671	41417	19384
Expected Cases	2565.41	27319.33	10906.15
Annual Cases / 100000	43.1	19.3	22.7
Relative Risk	3.5	1.67	1.87
Log Likelihood Ratio	4559.192567	3795.685917	2883.030263
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 4559.192567, 3795.685917 & 2883.030263 respectively.
- Also, the Relative Risk value is also in decreasing order for primary & secondary cluster with values 3.5, 1.67 respectively, whereas tertiary cluster value is more than secondary cluster but less than primary cluster.
- P-Value for all 3 clusters is less than 0.05.

For Year 2015:

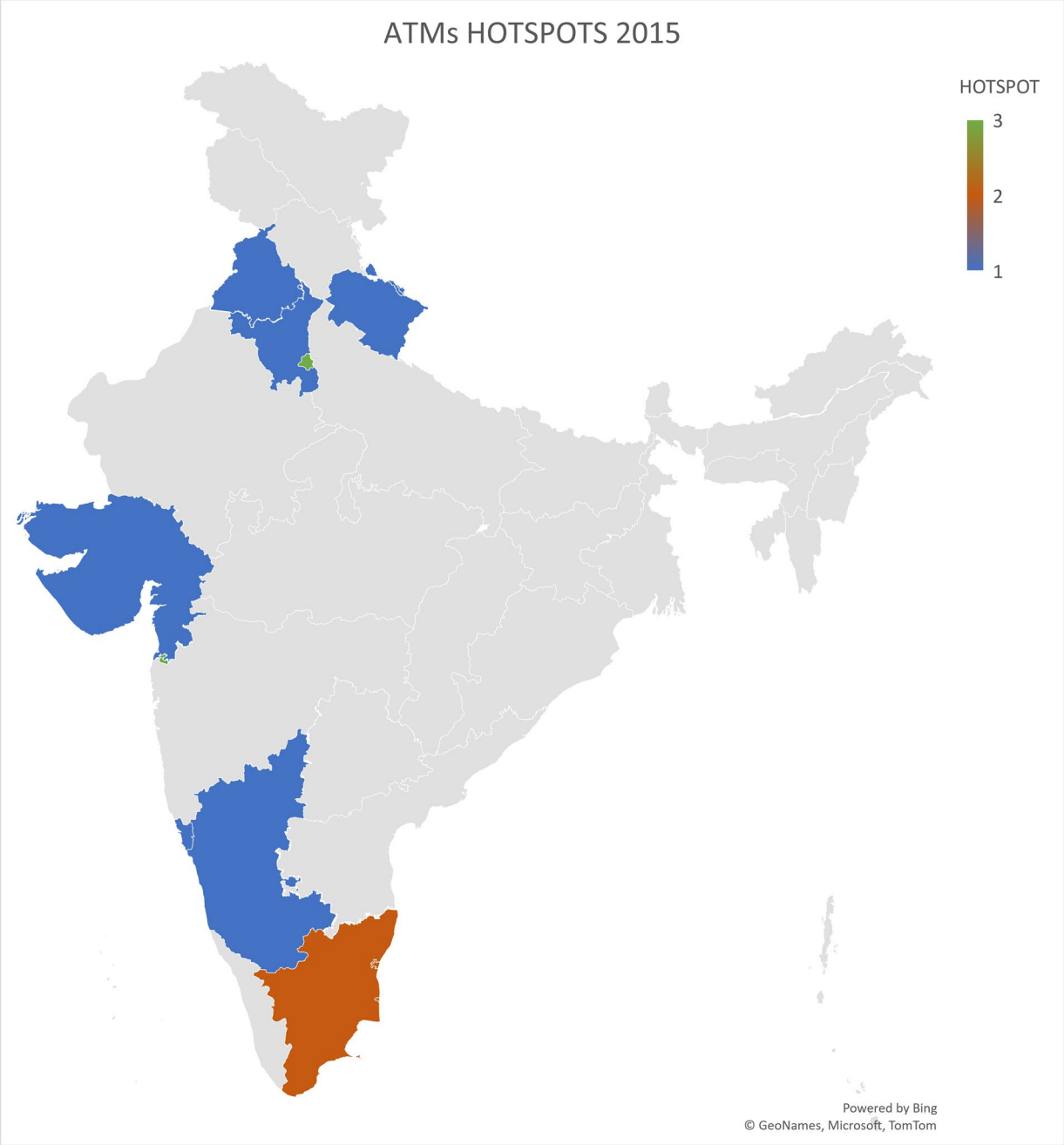


Figure 5: ATMs Hotspot 2015

- Primary Hotspots (Green): Delhi.
- Secondary Hotspots (Red): Puducherry, Tamil Nadu.
- Tertiary Hotspots (Blue): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.

Statistics	Primary	Secondary	Tertiary
Number of Cases	9086	23128	45104
Expected Cases	2883.31	12257.6	30704.65
Annual Cases / 100000	45.2	27	21.1
Relative Risk	3.25	2	1.6
Log Likelihood Ratio	4322.195027	4125.44521	3556.419426
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 4322.195027, 4125.44521 & 3556.419426 respectively.
- Also, the Relative Risk value is also in decreasing order from with values 3.25, 2 & 1.6 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2016:

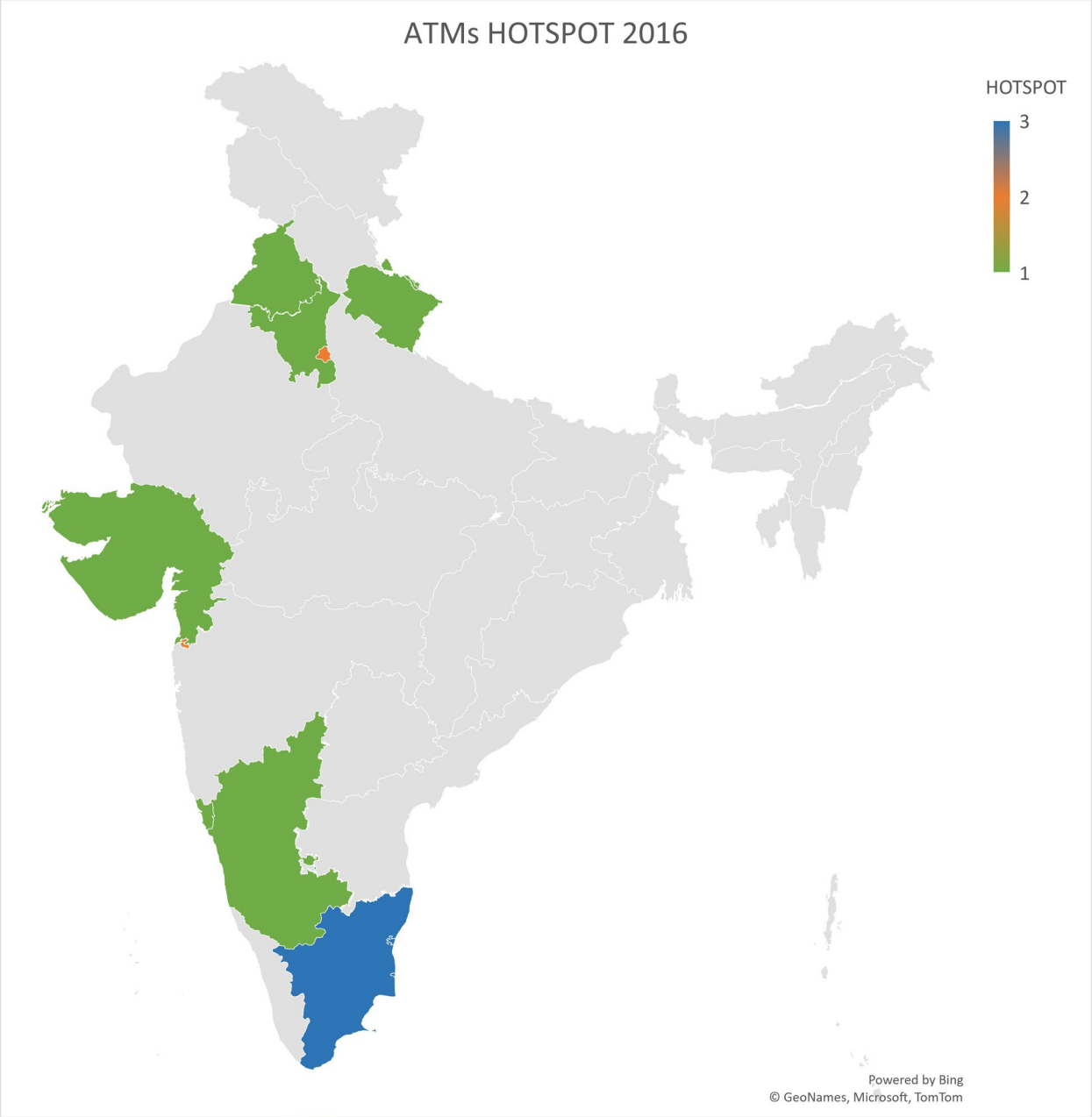


Figure 6: ATMs Hotspot 2016

- Primary Hotspots (Blue): Puducherry, Tamil Nadu.
- Secondary Hotspots (Red): Delhi.
- Tertiary Hotspots (Green): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.

Statistics	Primary	Secondary	Tertiary
Number of Cases	25038	9560	47804
Expected Cases	13112.01	3084.29	32844.89
Annual Cases / 100000	29.3	47.5	22.3
Relative Risk	2.03	3.2	1.58
Log Likelihood Ratio	4621.407375	4437.004969	3598.016332
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

`

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 4621.407375, 4437.004969 & 3598.016332 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.03, 3.2 & 1.58 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2017:

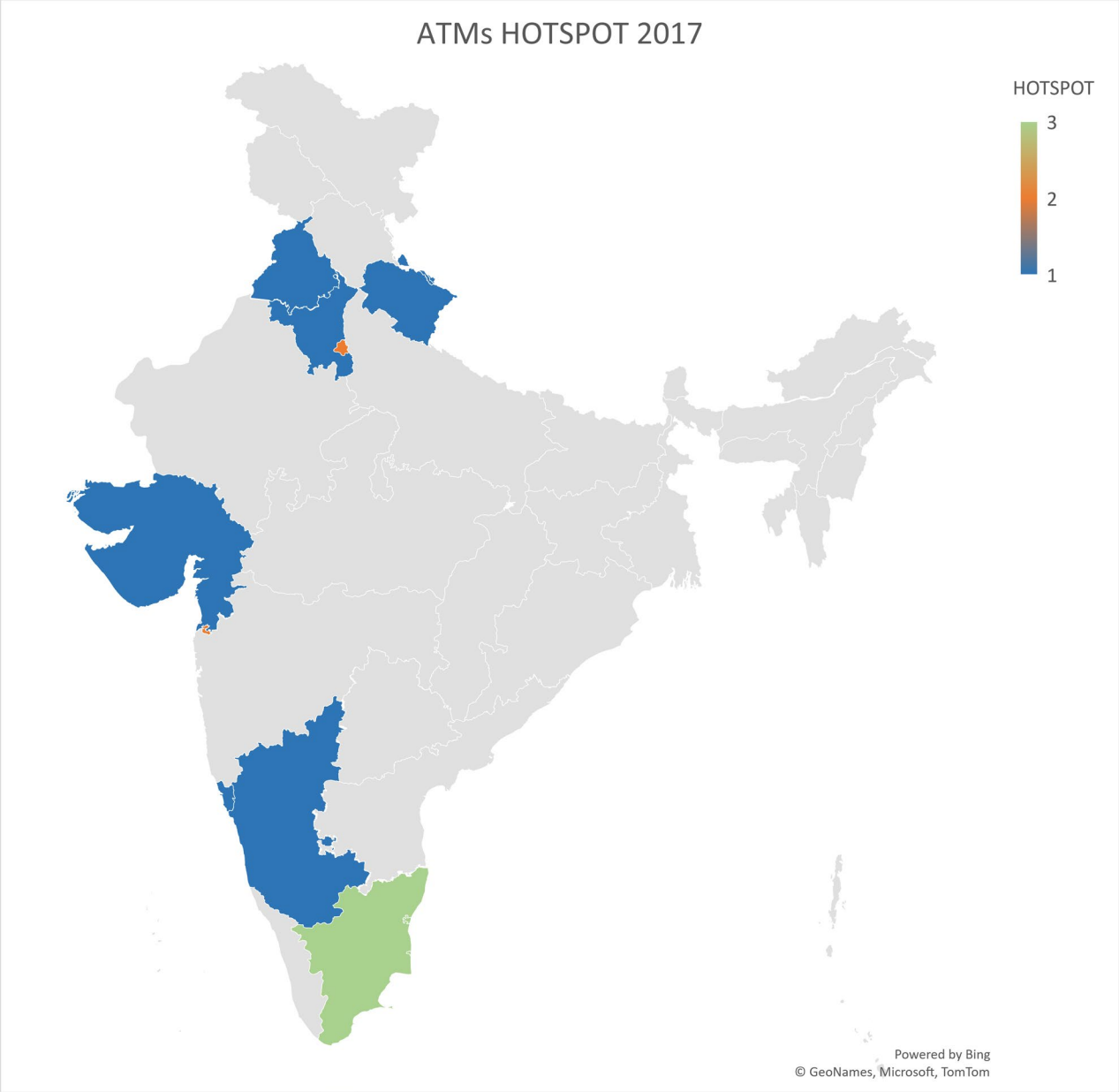


Figure 7: ATMs Hotspot 2017

- Primary Hotspots (Green): Puducherry, Tamil Nadu.
- Secondary Hotspots (Red): Delhi.
- Tertiary Hotspots (Blue): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.

Statistics	Primary	Secondary	Tertiary
Number of Cases	25720	9069	47921
Expected Cases	13232.66	3112.67	33147.11
Annual Cases / 100000	30.1	45.1	22.4
Relative Risk	2.07	3	1.57
Log Likelihood Ratio	4987.457322	3823.808579	3484.378182
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 4987.457322, 3823.808579 & 3484.378182 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.07, 3 & 1.57 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2018:

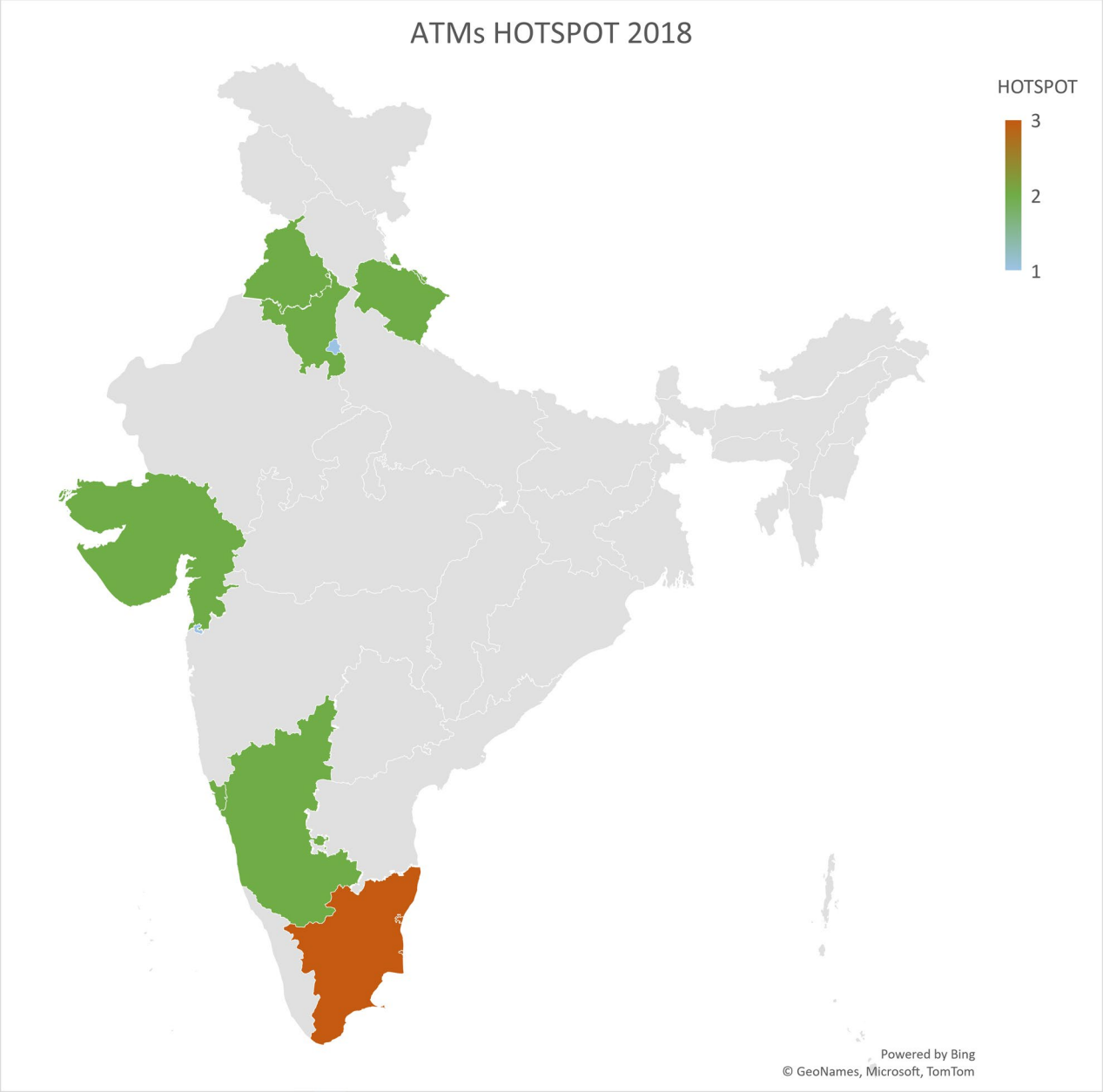


Figure 8: ATMs Hotspot 2018

- Primary Hotspots (Red): Puducherry, Tamil Nadu.
- Secondary Hotspots (Green): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.
- Tertiary Hotspots (Blue): Delhi.

Statistics	Primary	Secondary	Tertiary
Number of Cases	26131	47452	8631
Expected Cases	13136.73	32906.8	3090.1
Annual Cases / 100000	30.6	22.2	42.9
Relative Risk	2.12	1.56	2.87
Log Likelihood Ratio	5393.132114	3404.57967	3395.853925
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 5393.132114, 3404.57967 & 3395.853925 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.12, 1.56 & 2.87 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2019:

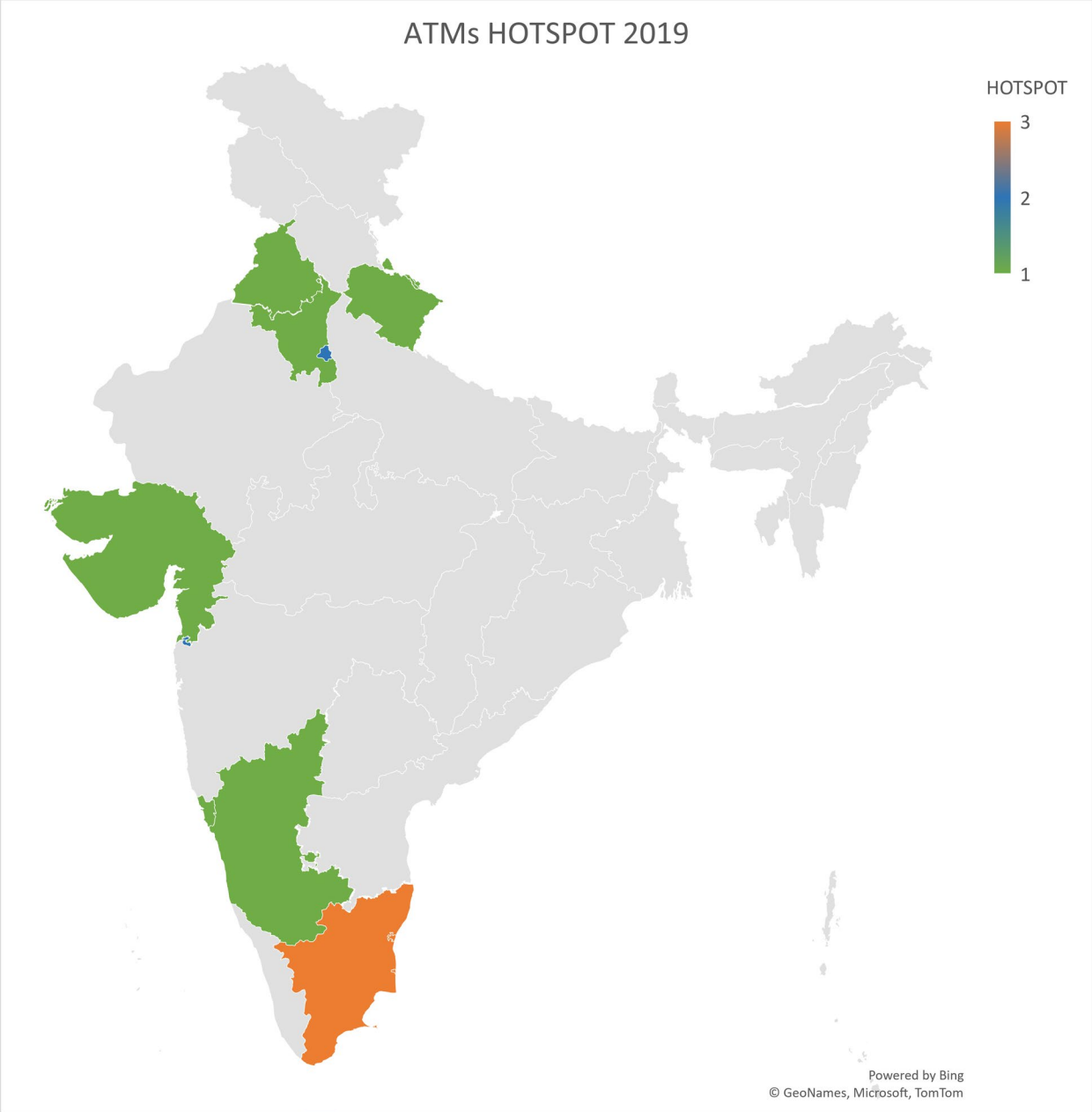


Figure 9: ATMs Hotspot 2019

- Primary Hotspots (Orange): Puducherry, Tamil Nadu.
- Secondary Hotspots (Blue): Delhi.
- Tertiary Hotspots (Green): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.

Statistics	Primary	Secondary	Tertiary
Number of Cases	27369	8861	48953
Expected Cases	13876.69	3264.16	34760.37
Annual Cases / 100000	32	44	22.9
Relative Risk	2.1	2.78	1.52
Log Likelihood Ratio	5522.014686	3321.165454	3090.152646
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 5522.014686, 3321.165454 & 3090.152646 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.1, 2.78 & 1.52 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2020:

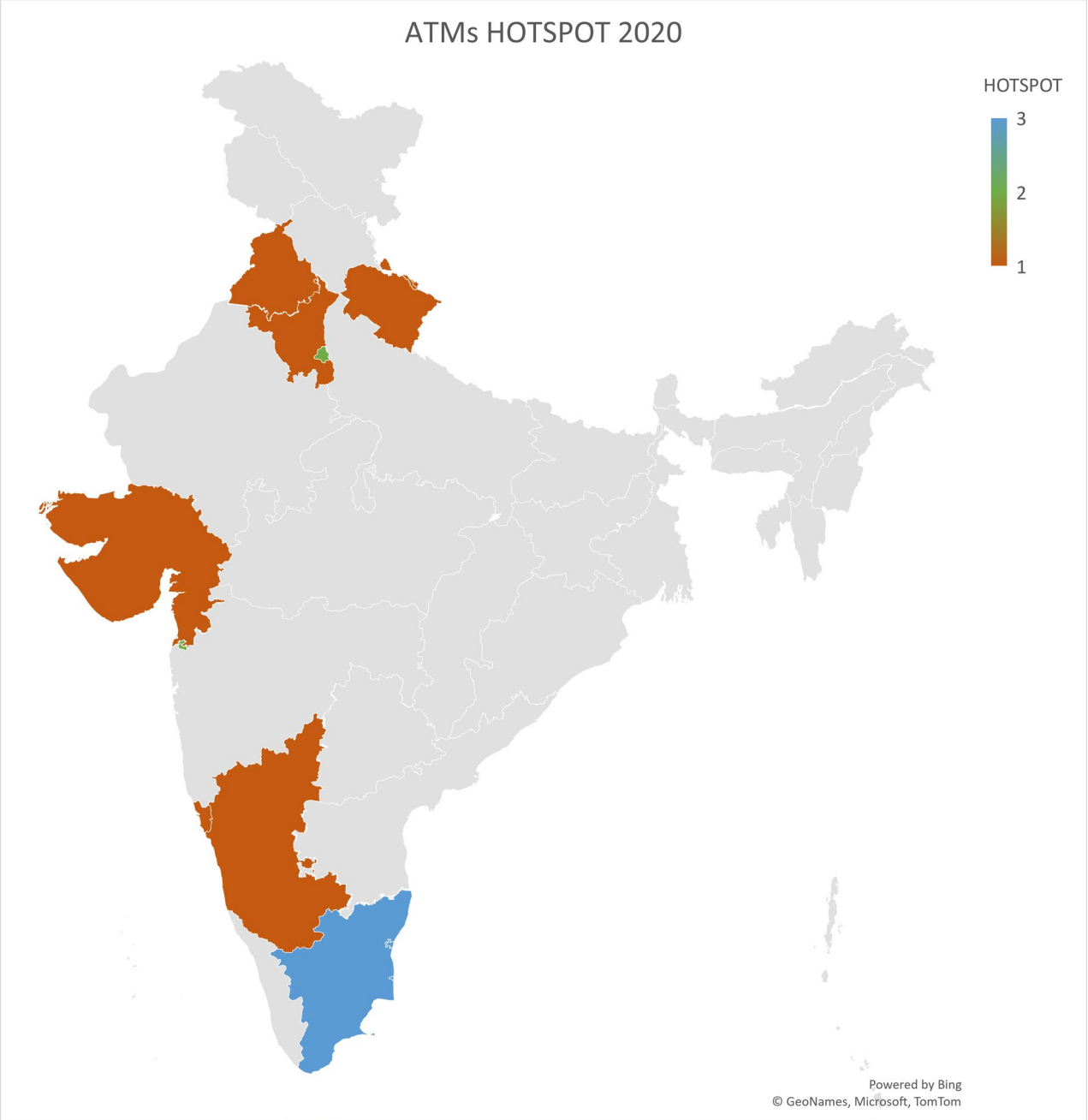


Figure 10: ATMs Hotspot 2020

- Primary Hotspots (Blue): Puducherry, Tamil Nadu.
- Secondary Hotspots (Green): Delhi.
- Tertiary Hotspots (Red): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.

Statistics	Primary	Secondary	Tertiary
Number of Cases	27484	8605	48741
Expected Cases	13908.63	3271.67	34840.38
Annual Cases / 100000	32.1	42.8	22.8
Relative Risk	2.11	2.69	1.5
Log Likelihood Ratio	5573.417185	3050.468898	2963.277849
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 5573.417185, 3050.468898 & 2963.277849 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.11, 2.69 & 1.5 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2021:

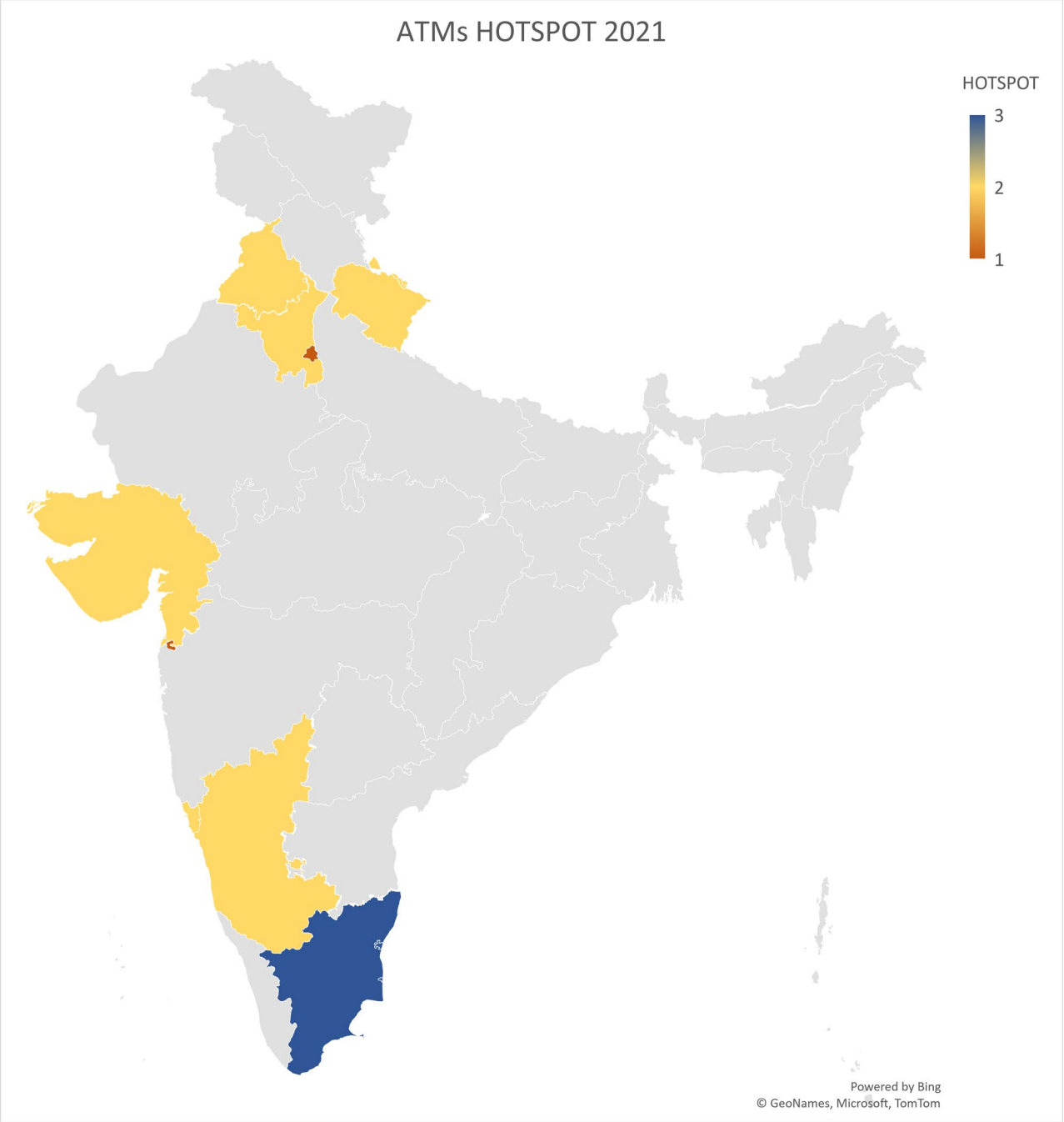


Figure 11: ATMs Hotspot 2021

- Primary Hotspots (Blue): Puducherry, Tamil Nadu.
- Secondary Hotspots (Yellow): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.
- Tertiary Hotspots (Red): Delhi.

Statistics	Primary	Secondary	Tertiary
Number of Cases	28913	50934	8270
Expected Cases	14604.35	36583.13	3435.33
Annual Cases / 100000	33.8	23.8	41.1
Relative Risk	2.11	1.5	2.46
Log Likelihood Ratio	5892.658677	3012.13476	2479.490267
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 5892.658677, 3012.13476 & 2479.490267 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.11, 1.5 & 2.46 respectively.
- P-Value for all 3 clusters is less than 0.05.

For Year 2022:

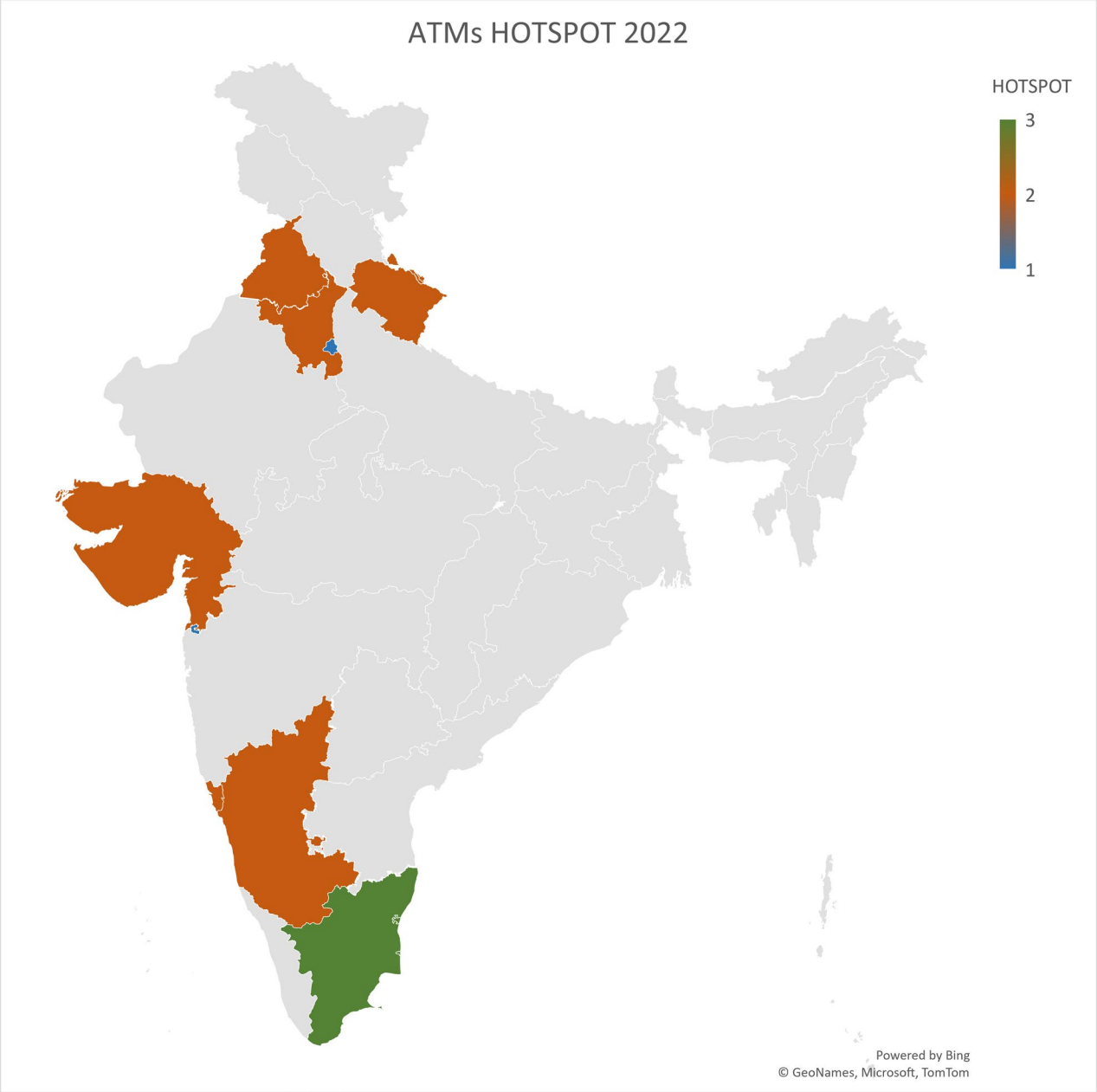


Figure 12: ATMs Hotspot 2021

- Primary Hotspots (Green): Puducherry, Tamil Nadu.
- Secondary Hotspots (Red): Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand.
- Tertiary Hotspots (Blue): Delhi.

Statistics	Primary	Secondary	Tertiary
Number of Cases	30534	53016	8220
Expected Cases	15304.26	38336.36	3599.96
Annual Cases / 100000	35.7	24.7	40.9
Relative Risk	2.13	1.48	2.33
Log Likelihood Ratio	6351.955064	3013.556158	2209.263486
P-Value	< 0.00000000000000001	< 0.00000000000000001	< 0.00000000000000001

- From the table we observe that the Log Likelihood Value is highest for primary cluster to secondary cluster & tertiary cluster is in the decreasing order 6351.955064, 3013.556158 & 2209.263486 respectively.
- Also, the Relative Risk value is in order of primary, secondary & tertiary values 2.13, 1.48 & 2.33 respectively.
- P-Value for all 3 clusters is less than 0.05.

- Based on the hotspots analysis we can observe that – Puducherry, Tamil Nadu, Delhi, Haryana, Karnataka, Gujarat, Chandigarh, Goa, Punjab, Uttarakhand has been frequent hotspots for No. of ATMs, CRMs in India.