Public Transportation Analysis

2023 Naan Mudhalvan - IBM Data Analytics with Cognos
Group 1 - Project 8
College : NM001 - College of Engineering Guindy
Proj_200340_Team_2

Members: Abinithi R, Abirami S V, Adithya R U, Akshaya G R,
Sai Rishi A N
Faculty Mentor : Dr. G Geetha

PHASE 5

PROJECT DOCUMENTATION AND SUBMISSION

PROBLEM DEFINITION :

Analyse public transportation data to assess **service efficiency**, **on time performance**, and **passenger feedback.**
Provide insights that **support transportation improvement initiatives** and enhance the overall public transportation experience.

Design Thinking :

1. **Analysis Objectives**: Define specific objectives for analysing public transportation data, such as assessing on-time performance, passenger satisfaction, and service efficiency.
2. **Data Collection**: Identify the sources and methods for collecting transportation data, including schedules, real-time updates, and passenger feedback.
3. **Visualisation Strategy**: Plan how to visualise the insights using IBM Cognos to create informative dashboards and reports.
4. **Code Integration**: Decide which aspects of the analysis can be enhanced using code, such as data cleaning, transformation, and statistical analysis.

Analysis Objectives :

1. **On-Time Performance :**
   - *Punctuality Rate*, *Average Delay* and *Peak Hour Delays*

2. **Reliability :**
   - *Service Reliability* and *Headway Adherence*

3. **Ridership :**
   - *Ridership Volume, Ridership Growth ,* and *Occupancy Rate*

4. **Service Coverage :**
   - *Service Area Coverage* and *Accessibility*

5. **Service Efficiency :**
   - *Average Speed, Dwell Time* and *Vehicle Utilisation*

6. **Customer Satisfaction :**
   - *Customer Surveys* and *Complaints and Feedback Analysis*

7. **Safety :**
   - *Accident Rate* and *Emergency Response Time*

8. **Environmental Impact :**
   - *Emissions Reduction* and *Vehicle Efficiency*

9. **Financial :**
   - *Cost per Rider* and *Revenue Generation*

10. **Infrastructure :**
    - *Maintenance and Repairs* and *Infrastructure Investment*

11. **Data Visualization :**
    - *Dashboard Usage*

12. **Operational Efficiency :**
    - *Operational Costs* and *Route Optimization*

With the provided dataset, metrics **3**, **4** and **12** can be applied to provide relevant and consistent results.

Dataset Summary :

Obtained from **https://www.kaggle.com/datasets/rednivrug/unisys?select=20140711.CSV** as per the project instructions.

TripID : an **integer** attribute which uniquely identifies trips
RouteID : a **string** attribute which uniquely specifies a transport route
StopID : an **integer** attribute which uniquely identifies stops in a transport network
StopName : a **string** attribute which gives a name to the corresponding StopID
WeekBeginnning : a **date/time** attribute which represents the first day of any given week
NumberOfBoardings : an **integer** attribute which keeps count of all the boardings at that specific stop for that specific trip over the week.

For example, a record in the dataset can be read as :

```
23631, 100, 14156, 181 Cross Rd, 2013-06-30 00:00:00, 1
```

> " For **StopName** *181 Cross Road* (corresponding **StopID** of *14156*) in the week which began at *2013-06-30*, had *1* boarding(s) during the trip identified by **TripID** *23631* which followed the route identified by **RouteID** *100*. "

DEVELOPMENT PHASES

1. DATA PREPROCESSING

❖ Cleaning and Preprocessing the Dataset:

Handling Missing Values

Missing data can significantly affect the performance of machine learning models. There are several methods to handle missing values, including:

- ***Removing Rows:*** Rows with missing values can be removed, but this might result in losing valuable data.
- ***Filling with Mean/Median/Mode*:** Filling missing values with the mean (average), median (middle value), or mode (most frequent value) of the respective column.
- ***Advanced Imputation Techniques**:* Using advanced techniques such as K-nearest neighbors imputation or regression imputation to predict missing values based on other features.

Encoding Categorical Variables:

Machine learning algorithms generally work with numerical data. Categorical variables like RouteID and StopName need to be converted into numerical representations. Common methods include:

- ***Label Encoding:*** Assign a unique numerical label to each category. For example, for RouteID: {"RouteA": 1, "RouteB": 2, ...}.
- ***One-Hot Encoding:*** Creating binary columns for each category. For example, for RouteID, columns like RouteA, RouteB, etc., can be created with binary values indicating presence or absence.
- ***Hashing:*** Converting categories into hash values. This method is useful when dealing with a large number of categories.

❖ Creating Features:

The day of the week's information can be extracted from the 'WeekBeginning' attribute. This can be important for capturing patterns related to specific days (e.g., weekdays vs. weekends). This information can be also used to create a new feature with values like 1 for Monday, 2 for Tuesday, and so on.

❖ Splitting Data into Training and Testing Sets:

- **Training Set**: This portion of the dataset is used to train the machine learning model. It contains input features and their corresponding output labels. The model learns patterns from this data.
- **Testing Set:** This portion of the dataset is used to evaluate the performance of the trained model. It helps assess how well the model generalizes to new, unseen data. The testing set also contains input features and corresponding output labels, but the model has never seen these data points during training.

2. FEATURE ENGINEERING

Feature engineering is the process of selecting, transforming, or creating relevant features from raw data to enhance the performance of machine learning models. Well-engineered features can significantly improve a model's ability to learn patterns and make predictions. It can be performed in the following ways:

❖ Extracting Relevant Features:

Relevant features can be extracted in several ways:

- **Aggregation:** Calculating statistics like averages, sums, minimum, maximum, or standard deviations for numerical attributes. For example, the average number of boardings per day for each stop can be calculated. This aggregated information can provide valuable insights.

- **Temporal Features:** Extracting time-based features such as day of the week, month, year, or specific events like holidays can capture patterns related to specific time periods.

❖ Feature Scaling and Normalization:

It might be necessary to scale or normalize the features. Many machine learning algorithms are sensitive to the scale of features. Common techniques include Min-Max scaling and Z-score normalization.

❖ Iterative Process:

Feature engineering is often an iterative process. After creating initial features, it's essential to assess the model's performance. If the model is not performing well, revisiting and refining the features can lead to better results. Domain expertise and a good understanding of the problem domain play a crucial role in this iterative process.

3. LABEL GENERATION

Label Generation is a critical step in supervised machine learning, where the target variable (also known as the label or output) that the model will learn to predict is defined. For predicting service disruptions, a binary variable indicating whether a disruption has occurred (1) or not (0) can be created. The following steps are followed:

❖ Defining the Disruption Criteria:

- **Domain Knowledge:** Domain experts can be consulted to understand what constitutes a service disruption. Disruptions can vary based on the nature of the service, and experts can provide valuable insights into defining disruption criteria.
- **Identifying Patterns:** Analyzing historical data to identify patterns associated with disruptions and looking for features such as unusually low boardings, delays in schedules, incidents reported, or any other indicators that suggest service interruptions.
- **Thresholds:** Based on the analysis and expert consultation, thresholds can be set for the identified criteria. For example, a day with boardings significantly lower than the average may be considered as a disruption.

❖ Creating the Binary Label:

Once the disruption criteria and thresholds have been defined, create a binary label column in the dataset. Assign a value of 1 to instances (days, stops, routes, etc.) where the disruption criteria are met, and 0 to instances where there is no disruption. For instance, if a specific stop has boardings lower than a certain threshold on a given day, that day would be labelled with a 1, indicating a disruption. Days where boardings are above the threshold would be labeled as 0, indicating no disruption.

❖ Ensuring Consistency:

Consistency has to be ensured in applying the disruption criteria across the dataset. It is crucial that the criteria and thresholds are applied uniformly to all relevant instances. Consistency in

labeling is essential for the model to learn meaningful patterns.

❖ **Validation and Refinement:**

The labeled data should be validated by cross-referencing it with incident reports or other reliable sources of disruption information to ensure accuracy. If discrepancies are found or the model isn't performing as expected, the disruption criteria and labels should be refined iteratively based on feedback from the validation process.

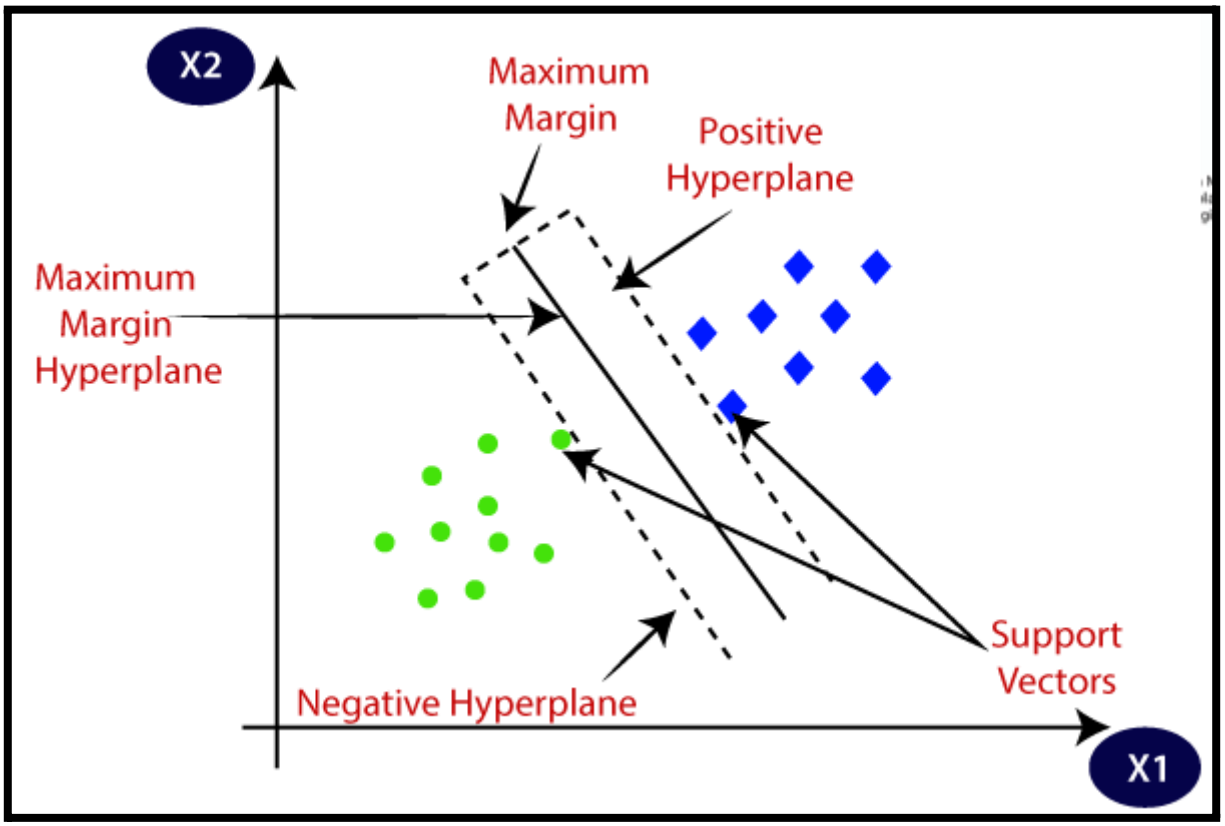
4. MODEL ARCHITECTURE

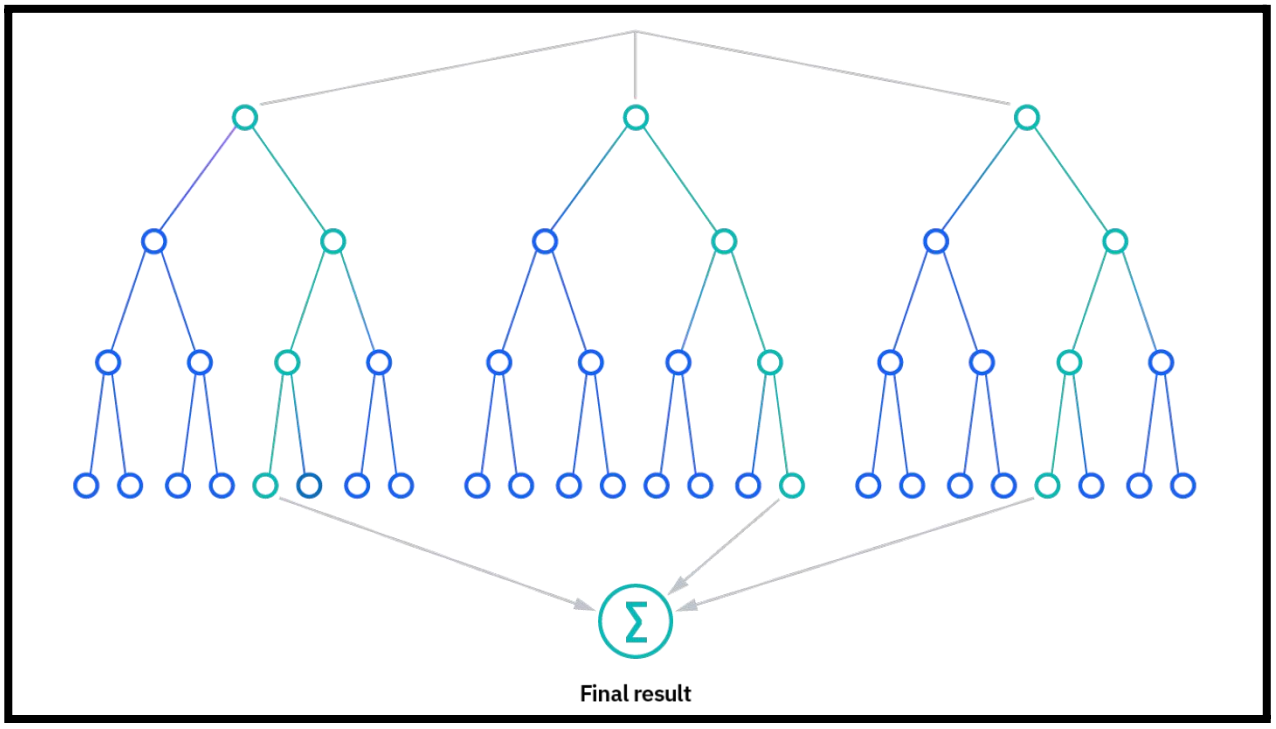
Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct categoryin the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:





.

Types of SVM:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

5. MODEL EVALUATION

❖ Performance Metrics for Classification

In a classification problem, the category or classes of data is identified based on training data. The model learns from the given dataset and then classifies the new data into classes or groups based on the training. It predicts class labels as the output, such as *Yes or No, 0 or 1, Spam or Not Spam,* etc. To evaluate the performance of a classification model, different metrics are used, and some of them are as follows:

- **Accuracy**
- **Confusion Matrix**
- **Precision**
- **Recall**
- **F-Score**
- **AUC(Area Under the Curve)-ROC**

I. Accuracy

The accuracy metric is one of the simplest Classification metrics to implement, and it can be determined as the number of correct predictions to the total number of predictions.

It can be formulated as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

To implement an accuracy metric, we can compare ground truth and predicted values in a loop, or we can also use the scikit-learn module for this. Although it is simple to use and implement, it is suitable only for cases where an equal number of samples belong to each class.

When to Use Accuracy?
It is good to use the Accuracy metric when the target variable classes in data are approximately balanced. For example, if 60% of classes in a fruit image dataset are of Apple, 40% are Mango. In this case, if the model is asked to predict whether the image is of Apple or Mango, it will give a prediction with 97% of accuracy.

When not to use Accuracy?
It is recommended not to use the Accuracy measure when the target variable majorly belongs to one class. For example, Suppose there is a model for a disease prediction in which, out of 100 people, only five people have a disease, and 95 people don't have one. In this case, if our model predicts every person with no disease (which means a bad prediction), the Accuracy measure will be 95%, which is not correct.

II. Confusion Matrix

A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known. A typical confusion matrix for a binary classifier looks like the below image(However, it can be extended to use for classifiers with more than two classes).

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

We can determine the following from the above matrix:

- In the matrix, columns are for the prediction values, and rows specify the Actual values. Here Actual and prediction give two possible classes, Yes or No. So, if we are predicting the presence of a disease in a patient, the Prediction column with Yes means, Patient has the disease, and for NO, the Patient doesn't have the disease.
- In this example, the total number of predictions are 165, out of which 110 time predicted yes, whereas 55 times predicted No.
- However, in reality, 60 cases in which patients don't have the disease, whereas 105 cases in which patients have the disease.

In general, the table is divided into four terminologies, which are as follows:

1. **True Positive(TP):** In this case, the prediction outcome is true, and it is true in reality, also.
2. **True Negative(TN):** in this case, the prediction outcome is false, and it is false in reality, also.
3. **False Positive(FP):** In this case, prediction outcomes are true, but they are false in actuality.
4. **False Negative(FN):** In this case, predictions are false, and they are true in actuality.

III. Precision

The precision metric is used to overcome the limitation of Accuracy. The precision determines the proportion of positive prediction that was actually correct. It can be calculated as the True Positive or predictions that are actually true to the total positive predictions (True Positive and False Positive).

$$Precision = \frac{TP}{(TP + FP)}$$

IV. Recall or Sensitivity

It is also similar to the Precision metric; however, it aims to calculate the proportion of actual positive that was identified incorrectly. It can be calculated as True Positive or predictions that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative).

The formula for calculating Recall is given below:

$$\text{Recall} = \frac{TP}{TP+FN}$$

When to use Precision and Recall?

From the above definitions of Precision and Recall, we can say that recall determines the performance of a classifier with respect to a false negative, whereas precision gives information about the performance of a classifier with respect to a false positive.

So, if we want to minimize the false negative, then, Recall should be as near to 100%, and if we want to minimize the false positive, then precision should be close to 100% as possible.

In simple words, *if we maximize precision, it will minimize the FP errors, and if we maximize recall, it will minimize the FN error.*

V. F-Scores

F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, ***the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.***

The formula for calculating the F1 score is given below:

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$

When to use F-Score?

As F-score make use of both precision and recall, so it should be used if both of them are important for evaluation, but one (precision or recall) is slightly more important to consider than the other. For example, when False negatives are comparatively more important than false positives, or vice versa.

VI. AUC-ROC

Sometimes we need to visualize the performance of the classification model on charts; then, we can use the AUC-ROC curve. It is one of the popular and important metrics for evaluating the performance of the classification model.

ROC represents a graph to show the performance of a classification model at different threshold levels. The curve is plotted between two parameters, which are:

- **True Positive Rate**
- **False Positive Rate**

TPR or true Positive rate is a synonym for Recall, hence can be calculated as:

$$TPR = \frac{TP}{TP + FN}$$

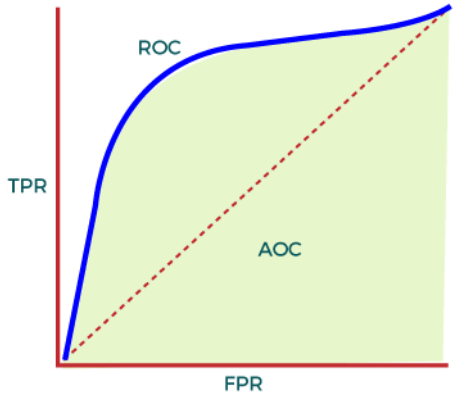
FPR or False Positive Rate can be calculated as:

$$TPR = \frac{FP}{FP + TN}$$

To calculate value at any point in a ROC curve, we can evaluate a logistic regression model multiple times with different classification thresholds, but this would not be much efficient. So, for this, one efficient method is used, which is known as AUC.

AUC: Area Under the ROC curve

AUC is known for **Area Under the ROC curve**. As its name suggests, AUC calculates the two-dimensional area under the entire ROC curve, as shown below image:



AUC calculates the performance across all the thresholds and provides an aggregate measure. The value of AUC ranges from 0 to 1. It means a model with 100% wrong prediction will have an AUC of 0.0, whereas models with 100% correct predictions will have an AUC of 1.0.

When to Use AUC

AUC should be used to measure how well the predictions are ranked rather than their absolute values. Moreover, it measures the quality of predictions of the model without considering the classification threshold.

When not to use AUC

As AUC is scale-invariant, which is not always desirable, and we need calibrating probability outputs, then AUC is not preferable.

Further, AUC is not a useful metric when there are wide disparities in the cost of false negatives vs. false positives, and it is difficult to minimize one type of classification error.

❖ Performance Metrics for Regression

Regression is a supervised learning technique that aims to find the relationships between the dependent and independent variables. A predictive regression model predicts a numeric or discrete value. The metrics used for regression are different from the classification metrics. It means we cannot use the Accuracy metric (explained above) to evaluate a regression model; instead, the performance of a Regression model is reported as errors in the prediction. Following are the popular metrics that are used to evaluate the performance of Regression models.

- **Mean Absolute Error**
- **Mean Squared Error**
- **R2 Score**
- **Adjusted R2**

I. Mean Absolute Error (MAE)

Mean Absolute Error or MAE is one of the simplest metrics, which measures the absolute difference between actual and predicted values, where absolute means taking a number as Positive.
To understand MAE, let's take an example of Linear Regression, where the model draws a best fit line between dependent and independent variables. To measure the MAE or error in prediction, we need to calculate the difference between actual values and predicted values. But in order to find the absolute error for the complete dataset, we need to find the mean absolute of the complete dataset.
The below formula is used to calculate MAE:

$$MAE = 1/N \sum |Y - Y'|$$

Here,
Y is the Actual outcome, Y' is the predicted outcome, and N is the total number of data points.
MAE is much more robust for the outliers. One of the limitations of MAE is that it is not differentiable, so for this, we need to apply different optimizers such as Gradient Descent. However, to overcome this limitation, another metric can be used, which is Mean Squared Error or MSE.

II. Mean Squared Error

Mean Squared error or MSE is one of the most suitable metrics for Regression evaluation. It measures the average of the Squared difference between predicted values and the actual value given by the model.
Since in MSE, errors are squared, therefore it only assumes non-negative values, and it is usually positive and non-zero.
Moreover, due to squared differences, it penalizes small errors also, and hence it leads to over-estimation of how bad the model is.
MSE is a much-preferred metric compared to other regression metrics as it is differentiable and hence optimized better.
The formula for calculating MSE is given below:

$$MSE = 1/N \sum (Y - Y')^2$$

Here,
Y is the Actual outcome, Y' is the predicted outcome, and N is the total number of data points.

III. R Squared Score

R squared error is also known as Coefficient of Determination, which is another popular metric used for Regression model evaluation. The R-squared metric enables us to compare our model with a

constant baseline to determine the performance of the model. To select the constant baseline, we need to take the mean of the data and draw the line at the mean.
The R squared score will always be less than or equal to 1 without concerning if the values are too large or small.

$$R^2 = 1 - \frac{MSE(Model)}{MSE(Baseline)}$$

IV. Adjusted R Squared

Adjusted R squared, as the name suggests, is the improved version of R squared error. R square has a limitation of improvement of a score on increasing the terms, even though the model is not improving, and it may mislead the data scientists.
To overcome the issue of R square, adjusted R squared is used, which will always show a lower value than R^2. It is because it adjusts the values of increasing predictors and only shows improvement if there is a real improvement.
We can calculate the adjusted R squared as follows:

$$R_a^2 = 1-\left[\left(\frac{n-1}{n-k-1}\right) \times \left(1 - R^2\right)\right]$$

Here,
n is the number of observations
k denotes the number of independent variables
and R_a^2 denotes the adjusted R^2

6. TUNING AND OPTIMIZATION

The configuration and hyperparameter tuning can profoundly influence a model's performance. A slight tweak can be the difference between a mediocre outcome and stellar results. For instance, the Adam optimizer, a popular optimization method in deep learning, has specific hyperparameters that, when fine-tuned, can lead to faster and more stable convergence during training.

In real-world applications, hyperparameter search and fine-tuning become even more evident. Consider a scenario where a pre-trained neural network, initially designed for generic image recognition, is repurposed for a specialized task like medical image analysis. Its accuracy and reliability can be significantly enhanced by searching for optimal hyperparameters and fine-tuning them for this dataset. This could mean distinguishing between accurately detecting a medical anomaly and missing it altogether.

Furthermore, as machine learning evolves, our datasets and challenges become more complex. In such a landscape, the ability to fine-tune models and optimize hyperparameters using various optimization methods is not just beneficial; it's essential. It ensures that our models are accurate, efficient, adaptable, and ready to tackle the challenges of tomorrow.

HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization focuses on finding the optimal set of hyperparameters for a given model. Unlike model parameters, these hyperparameters are not learned during training but are set before the training begins. Their correct setting can significantly influence the model's performance.

Grid Search

Grid Search involves exhaustively trying out every possible combination of hyperparameters in a predefined search space. For instance, if you're fine-tuning a model and considering two hyperparameters, learning rate and batch size, a grid search would test all combinations of the values you specify for these hyperparameters.

For an SVM applied to this problem, two critical hyperparameters are:

- The type and parameters of the kernel: For instance, if using the Radial Basis Function (RBF) kernel, we need to determine the **gamma value**.
- The **regularization parameter (C)** determines the trade-off between maximizing the margin and minimizing classification error.

By training the SVM with each combination and validating its performance on a separate dataset, grid search allows us to pinpoint the combination that yields the best classification accuracy.

Random Search

Random Search, as the name suggests, involves randomly selecting and evaluating combinations of hyperparameters. Unlike Grid Search, which exhaustively tries every possible combination, Random Search samples a predefined number of combinations from a specified distribution for each hyperparameter.

Consider a scenario where a financial institution develops a machine learning model to predict loan defaults. The dataset is vast, with numerous features ranging from a person's credit history to current financial status.

The model has several hyperparameters like learning rate, batch size, and the number of layers. Given the high dimensionality of the hyperparameter space, using Grid Search might be computationally expensive and time-consuming. By randomly sampling hyperparameter combinations, the institution can efficiently narrow down the best settings with the highest prediction accuracy, saving time and computational resources.

Bayesian Optimization

Bayesian Optimization is a probabilistic model-based optimization technique particularly suited for optimizing expensive-to-evaluate and noisy functions. Unlike random or grid search, Bayesian Optimization builds a probabilistic model of the objective function. It uses it to select the most promising hyperparameters to evaluate the true objective function.

Bayesian Optimization shines in scenarios where the objective function is expensive to evaluate. For instance, training a model with a particular set of hyperparameters in deep learning can be time-consuming. Using grid search or random search in such scenarios can be computationally prohibitive. By building a model of the objective function, Bayesian Optimization can more intelligently sample the hyperparameter space to find the optimal set-in fewer evaluations.

Bayesian Optimization is more directed than grid search, which exhaustively tries every combination of hyperparameters, or random search, which samples them randomly. It uses past evaluation results to choose the next set of hyperparameters to evaluate. This makes it particularly useful when evaluating the objective function is time-consuming or expensive.

7. DEPLOYMENT

Once an ML model is ready, the next step is to make it available for users to make predictions. Placing an ML model in an environment for users to interact with and use for decision making refers to model deployment.

The Process of Deploying Machine Learning Models

ML model deployment involves the following steps:

1. **Develop, create, and test the model in a training environment:** This step requires rigorous training, testing, and optimization of the model to ensure high performance in production. The

model training step determines how models perform in production. ML teams must collaborate to optimize, clean, test, and retest model code.

2. **Movement of models to deployment environment:** After rigorous testing and optimizing model code, the top-performing models undergo preparation for deployment. Models need a deployment environment that contains all the hardware resources and data required to make the model perform optimally. Different deployment environments include:
    - **Containers:** Most teams use a container deploying environment because containers are reproducible, predictable, and easy to modify and update, making collaboration among engineers easy. Containers encompass all the hardware, configurations and dependencies necessary to deploy the model, improving consistency among ML teams.
    - **Notebooks:** Jupyter and AWS Sagemaker are common notebooks used by data scientists for experimentation in the ML lifecycle. However, notebooks present difficulties like reproducibility and testing for teams. To efficiently use notebooks in the production workflow, teams should consider code organization, reusability, and dependencies, among other factors.
    - **In-App environments:** This environment works when certain limitations or constraints exist around using data outside the application.
3. **Making models available for end users:** ML teams must choose how to make models available for their users. Most can be available on demand or deployed to edge devices.
4. **Monitoring:** The ML lifecycle continues after deployment. Deployed models must undergo constant monitoring to evaluate the performance and accuracy of models over time. Because data is in a continual state of motion and change, model degradation may occur. In this case, automating the ML workflow to monitor and retrain models constantly helps ensure the longevity of models.

8. INTERPRETABILITY

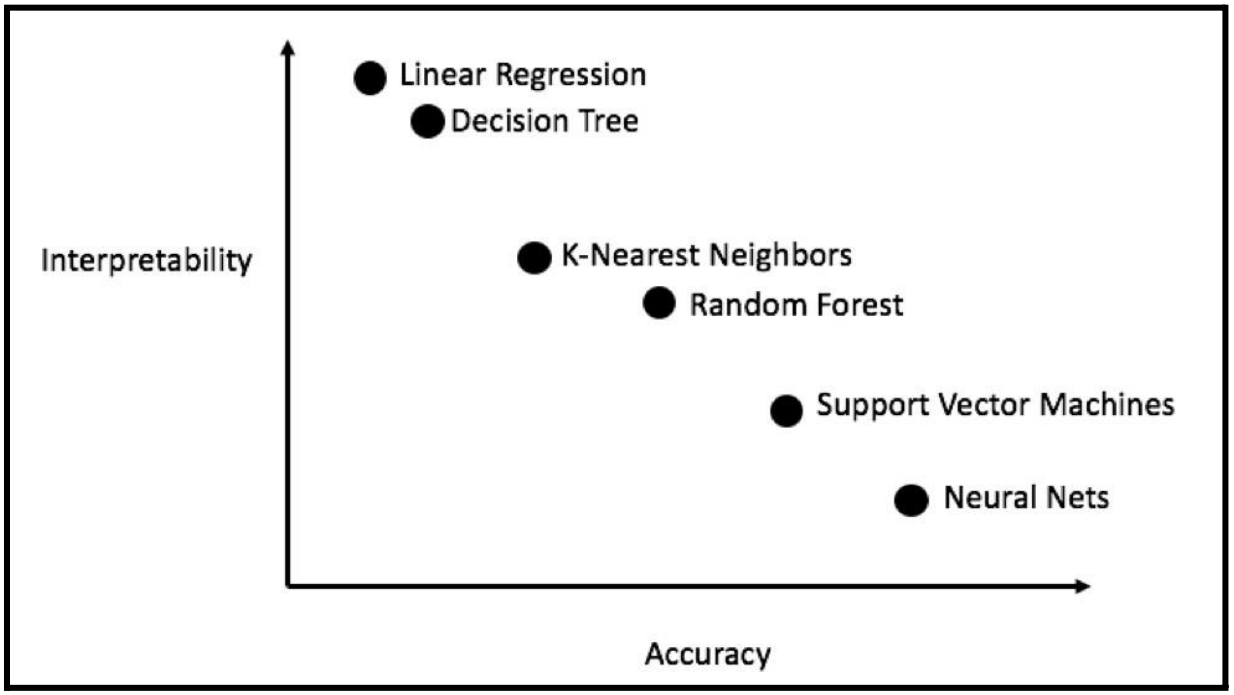
Interpretability is the degree to which a human can understand the cause of a decision. The higher the interpretability of an ML model, the easier it is to comprehend the model's predictions. Interpretability facilitates:

- Understanding
- Debugging and auditing ML model predictions
- Bias detection to ensure fair decision making
- Robustness checks to ensure that small changes in the input do not lead to large changes in the output
- Methods that provide recourse for those who have been adversely affected by model predictions

Model interpretability provides a mechanism to ensure the safety and effectiveness of ML solutions by increasing the transparency around model predictions, as well as the behavior of the underlying algorithm. Promoting transparency is a key aspect of the patient-centered approach, and is especially important for AI/ML-based SaMD, which may learn and change over time.

There is a tradeoff between *what* the model has predicted (model performance) and *why* the model has made such a prediction (model interpretability).

For some solutions, a high model performance is sufficient; in others, the ability to interpret the decisions made by the model is key. The demand for interpretability increases when there is a large cost for incorrect predictions, especially in high-risk applications.



Trade-off between performance and model interpretability

Based on the model complexity, methods for model interpretability can be classified into *intrinsic analysis* and *post hoc* analysis.

- **Intrinsic analysis** can be applied to interpret models that have low complexity (simple relationships between the input variables and the predictions). These models are based on:
    - Algorithms, such as linear regression, where the prediction is the weighted sum of the inputs
    - Decision trees, where the prediction is based on a set of if-then rules

The simple relationship between the inputs and output results in high model interpretability, but often leads to lower model performance, because the algorithms are unable to capture complex non-linear interactions.

- **Post hoc analysis** can be applied to interpret simpler models, as described earlier, as well as more complex models, such as neural networks, which have the ability to capture non-linear interactions. These methods are often model-agnostic and provide mechanisms to interpret a trained model based on the inputs and output predictions. Post hoc analysis can be performed at a *local* level, or at a *global* level.
    - **Local methods** enable you to zoom in on a single data point and observe the behavior of the model in that neighborhood. They are an essential component for debugging and auditing ML model predictions. Examples of local methods include:
    - **Local Interpretable Model-Agnostic Explanations** (LIME), which provides a sparse, linear approximation of the model behavior around a data point
    - **SHapley Additive exPlanations** (SHAP), a game theoretic approach based on Shapley values which computes the marginal contribution of each input variable towards the output

    - **Counterfactual explanations**, which describe the smallest change in the input variablesthat causes a change in the model's prediction
    - **Integrated gradients**, which provide mechanisms to attribute the model's prediction to specific input variables
    - **Saliency maps**, which are a pixel attribution method to highlight relevant pixels in an image
- **Global methods** enable you to zoom out and provide a holistic view that explains the overall behavior of the model. These methods are helpful for verifying that the model is robust and has the least possible bias to allow for fair decision making. Examples of global methods include:
    - **Aggregating local explanations**, as defined previously, across multiple data points
    - **Permutation feature importance**, which measures the importance of an input variableby computing the change in the model's prediction due to permutations of the input variable
    - **Partial dependence plots**, which plot the relationship and the marginal effect of an input variable on the model's prediction
    - **Surrogate methods**, which are simpler interpretable models that are trained to approximate the behavior of the original complex model

It is recommended to start the ML journey with a simple model that is both inherently interpretableand provides sufficient model performance. In later iterations, if you need to improve the model performance, AWS recommends increasing the model complexity and leveraging post hoc analysis methods to interpret the results.

Selecting both a local method and a global method gives you the ability to interpret the behavior ofthe model for a single data point, as well as across all data points in the dataset. It is also essential to validate the stability of model explanations, because methods in post-hoc analysis are susceptible toadversarial attacks, where small perturbations in the input could result in large changes in the outputprediction and therefore in the model explanations as well.

9. FEEDBACK LOOP

In AI, machines learn how to execute tasks that are typically performed by humans. Like humans, AI systems make mistakes during their infancy and need a feedback loop to confirm or invalidate their decisions.

Feedback loops allow AI systems to know what they did right or wrong, giving them data that enables them to adjust their parameters to perform better in the future. In the C3 AI Reliability application, operators can prioritize maintenance actions based on risk scores and trigger work orders. If users disagree with the application's recommendations, they can log their decisions tohelp the system do better next time.

AI systems need to adapt to evolving data or new patterns that appear over time. A feedback loop reinforces the model's training with fresh data. In the C3 AI Anti-Money-Laundering application, it is crucial to incorporate the latest typologies and theft modes using a closed-loop workflow to improve predictions.

ANALYSIS STEPS

DATA PREPROCESSING

Cleaning and Preprocessing the Dataset:

Handling Missing Values

Missing data can significantly affect the performance of machine learning models. There are several methods to handle missing values, including:

- *Removing Rows:* Rows with missing values can be removed, but this might result in losing valuable data.
- *Filling with Mean/Median/Mode*: Filling missing values with the mean (average), median (middle value), or mode (most frequent value) of the respective column.
- *Advanced Imputation Techniques:* Using advanced techniques such as K-nearest neighbors imputation or regression imputation to predict missing values based on other features.

IMPORTING NECESSARY LIBRARIES

In [13]:
```
import pandas as pd
import numpy as np
```

LOADING DATASET

In [3]:
```
data = pd.read_csv("C:\\Users\\AbiramiSV\\Downloads\\Dataset\\PublicTransportDataset.CSV"
```

DISPLAYING FIRST 20 ROWS

In [4]:
```
data.head(20)
```

Out[4]:

	TripID	RouteID	StopID	StopName	WeekBeginning	NumberOfBoardings
0	23631	100	14156	181 Cross Rd	2013-06-30 00:00:00	1
1	23631	100	14144	177 Cross Rd	2013-06-30 00:00:00	1
2	23632	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
3	23633	100	12266	Zone A Arndale Interchange	2013-06-30 00:00:00	2
4	23633	100	14147	178 Cross Rd	2013-06-30 00:00:00	1
5	23634	100	13907	9A Marion Rd	2013-06-30 00:00:00	1
6	23634	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
7	23634	100	13335	9A Holbrooks Rd	2013-06-30 00:00:00	1
8	23634	100	13875	9 Marion Rd	2013-06-30 00:00:00	1
9	23634	100	13045	206 Holbrooks Rd	2013-06-30 00:00:00	1
10	23635	100	13335	9A Holbrooks Rd	2013-06-30 00:00:00	1
11	23635	100	13383	8A Marion Rd	2013-06-30 00:00:00	1
12	23635	100	13586	8D Marion Rd	2013-06-30 00:00:00	2
13	23635	100	12726	23 Findon Rd	2013-06-30 00:00:00	1
14	23635	100	13813	8K Marion Rd	2013-06-30 00:00:00	1
15	23635	100	14062	20 Cross Rd	2013-06-30 00:00:00	1
16	23636	100	12780	22A Crittenden Rd	2013-06-30 00:00:00	1
17	23636	100	13383	8A Marion Rd	2013-06-30 00:00:00	1
18	23636	100	14154	180 Cross Rd	2013-06-30 00:00:00	2
19	23636	100	13524	8C Marion Rd	2013-06-30 00:00:00	3

DROPPING RECORDS HAVING DUPLICATE VALUES

In [5]:
```
data.drop_duplicates(inplace=True)
```

FILLING MISSING VALUES WITH MEAN

In [6]:
```
data.fillna(data.mean(), inplace=True)
```

PRINTING FIRST FEW ROWS

In [7]:
```
print(data.head())
```

```
   TripID RouteID  StopID                    StopName       WeekBeginning  \
0   23631     100   14156   181                Cross Rd  2013-06-30 00:00:00
1   23631     100   14144   177                Cross Rd  2013-06-30 00:00:00
2   23632     100   14132   175                Cross Rd  2013-06-30 00:00:00
3   23633     100   12266    Zone A Arndale Interchange  2013-06-30 00:00:00
4   23633     100   14147                 178 Cross Rd  2013-06-30 00:00:00

    NumberOfBoardin
gs0                 1
1                   1
2                   1
3                   2
4                   1
```

GENERATING DESCRIPTIVE STATISTICS OF DATASET

In [8]:
```
print(data.describe())
```

```
             TripID        StopID        NumberOfBoardings
count  1.085723e+07  1.085723e+07  1.085723e+07
mean   2.952100e+04  1.366132e+04  4.743737e+00
std    1.960938e+04  1.971760e+03  9.382286e+00
min    7.900000e+01  1.000100e+04  1.000000e+00
25%    1.191700e+04  1.231100e+04  1.000000e+00
50%    2.747900e+04  1.334600e+04  2.000000e+00
75%    4.885800e+04  1.491600e+04  4.000000e+00
max    6.553500e+04  1.871500e+04  9.770000e+02
```

GENERATING CONCISE SUMMARY OF DATASET

In [9]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10857234 entries, 0 to 10857233
Data columns (total 6 columns):
 #   Column              Dtype
---  ------              -----
 0   TripID              int64
 1   RouteID             object
 2   StopID              int64
 3   StopName            object
 4   WeekBeginning       object
 5   NumberOfBoardings   int64
dtypes: int64(3), object(3)
memory usage: 579.8+ MB
None
```

SHAPE OF DATASET

In [11]:
```
print(data.shape)
```

```
(10857234, 6)
```

DISPLAYING FIRST FEW ROWS AFTER PREPROCESSING

In [12]:
```
data.head()
```

Out[12]:

	TripID	RouteID	StopID	StopName	WeekBeginning	NumberOfBoardings
0	23631	100	14156	181 Cross Rd	2013-06-30 00:00:00	1
1	23631	100	14144	177 Cross Rd	2013-06-30 00:00:00	1
2	23632	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
3	23633	100	12266	Zone A Arndale Interchange	2013-06-30 00:00:00	2
4	23633	100	14147	178 Cross Rd	2013-06-30 00:00:00	1

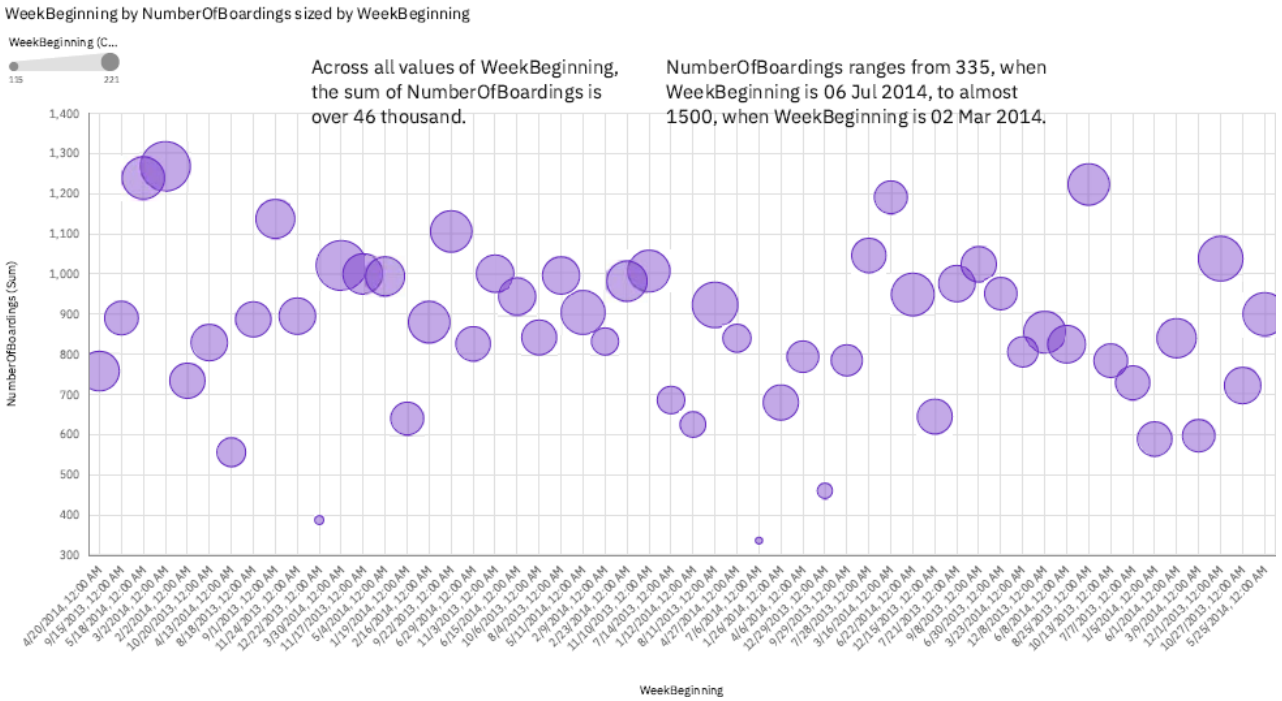
In [ ]:
```

```

VISUALIZATIONS IN COGNOS

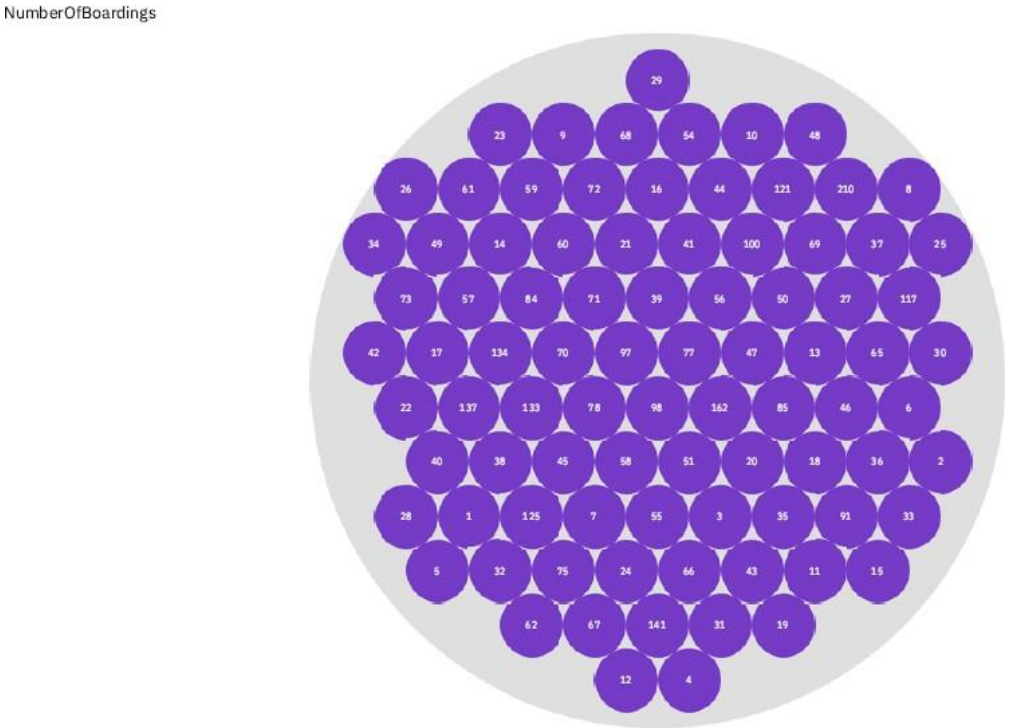
---

1. Bubble plot of WeekBeginning by NumberOfBoardings sized by WeekBeginning

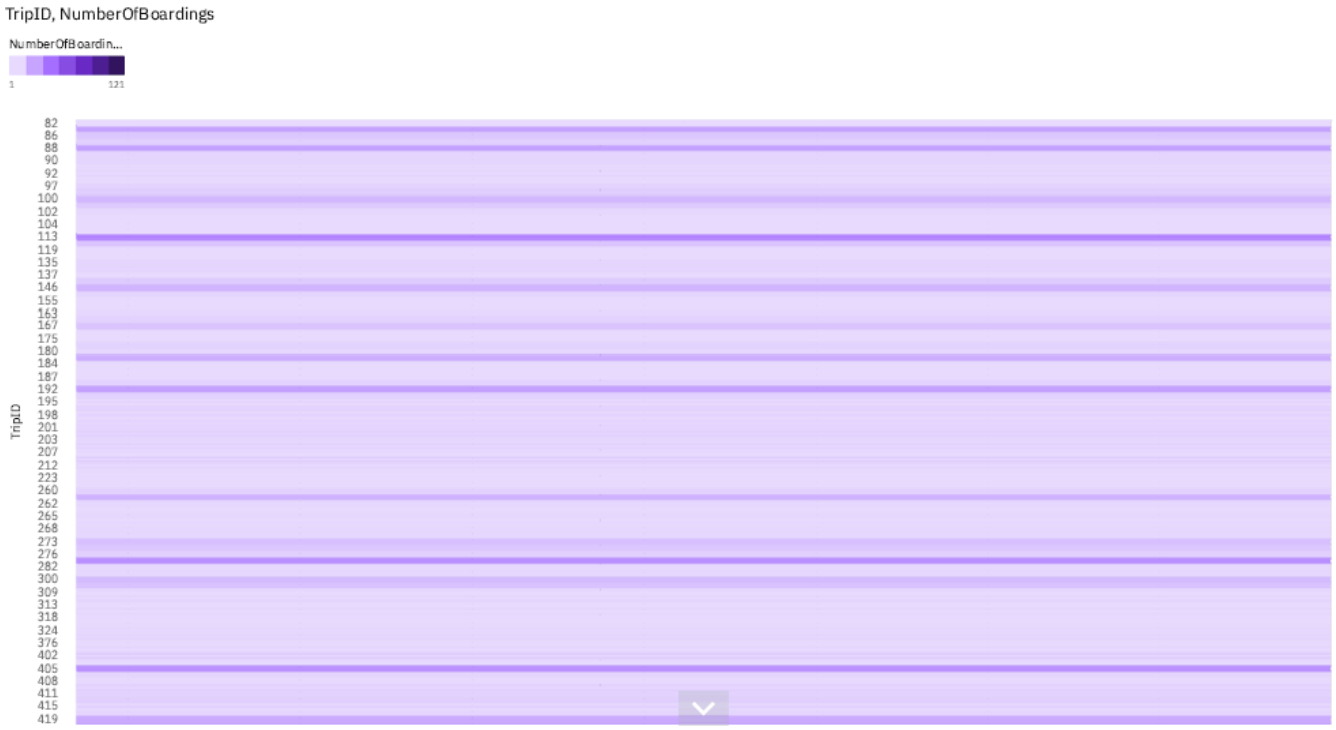


---

2. Hierarchy Bubble of NumberOfBoardings

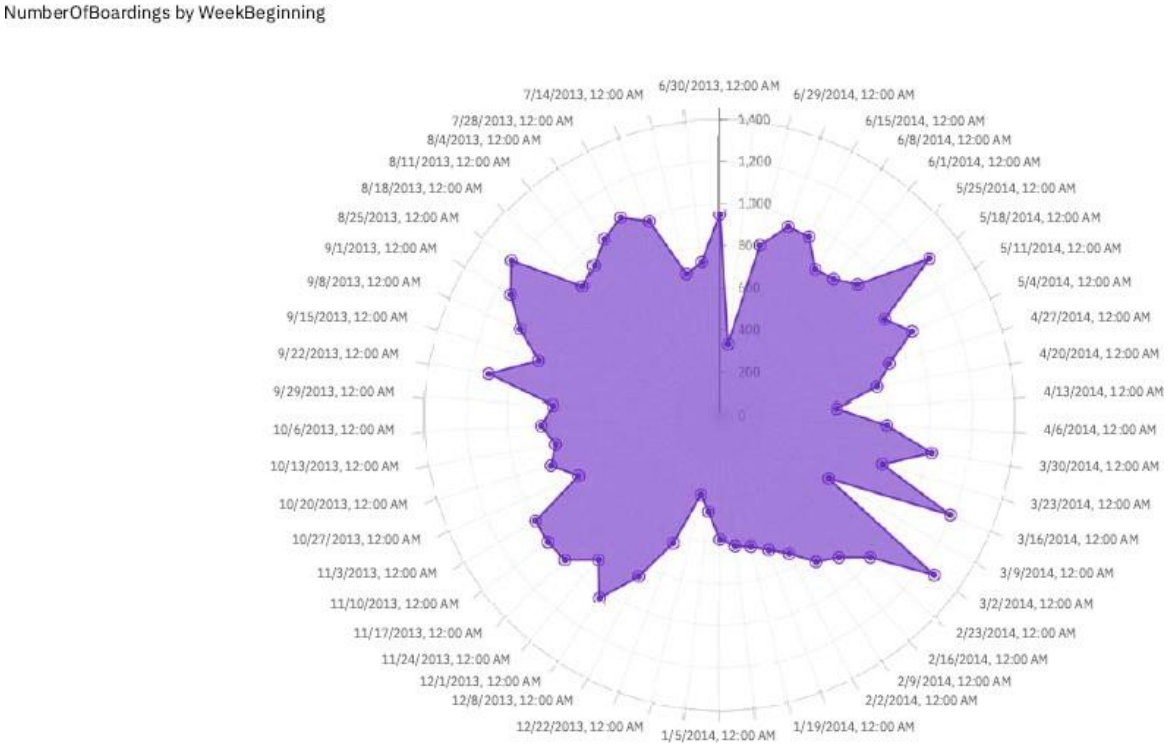


3. Heat Map of NumberOfBoardings by TripID



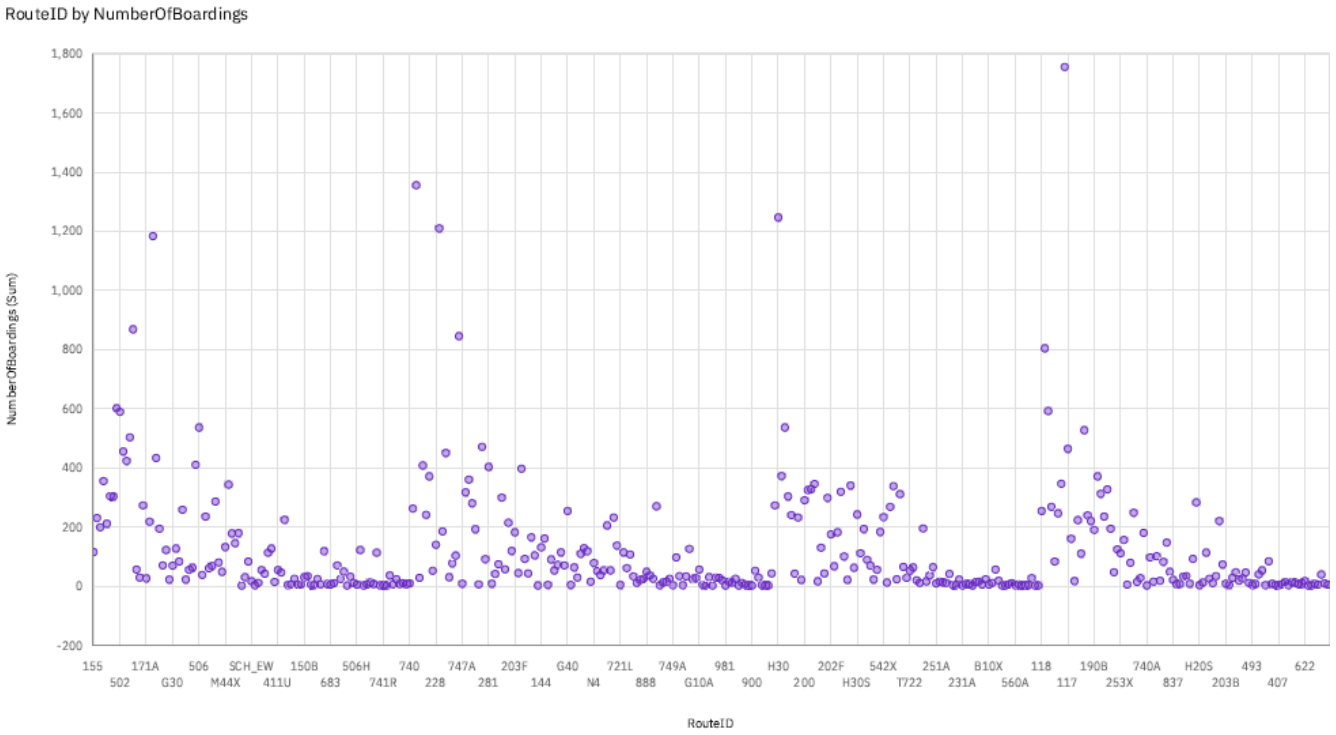
---

4. Radar of NumberOfBoardings by WeekBeginning



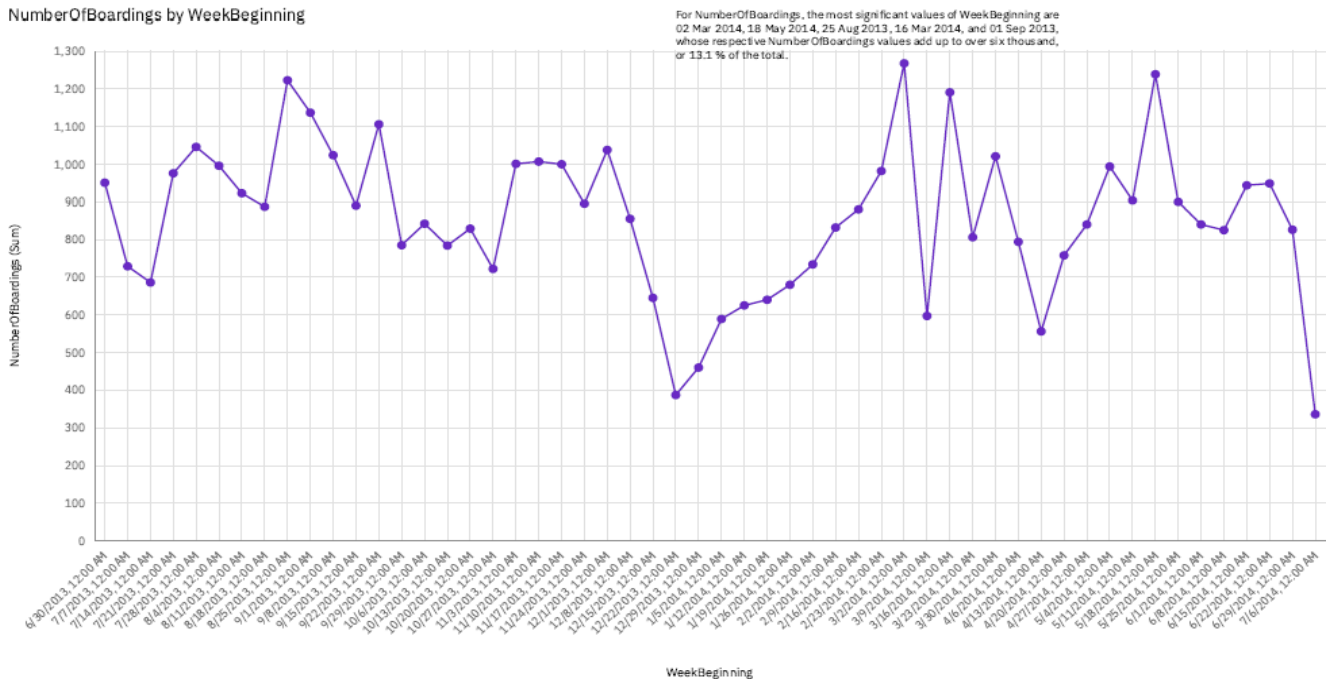
---

5. Scatter Plot of RouteID by NumberofBoardings

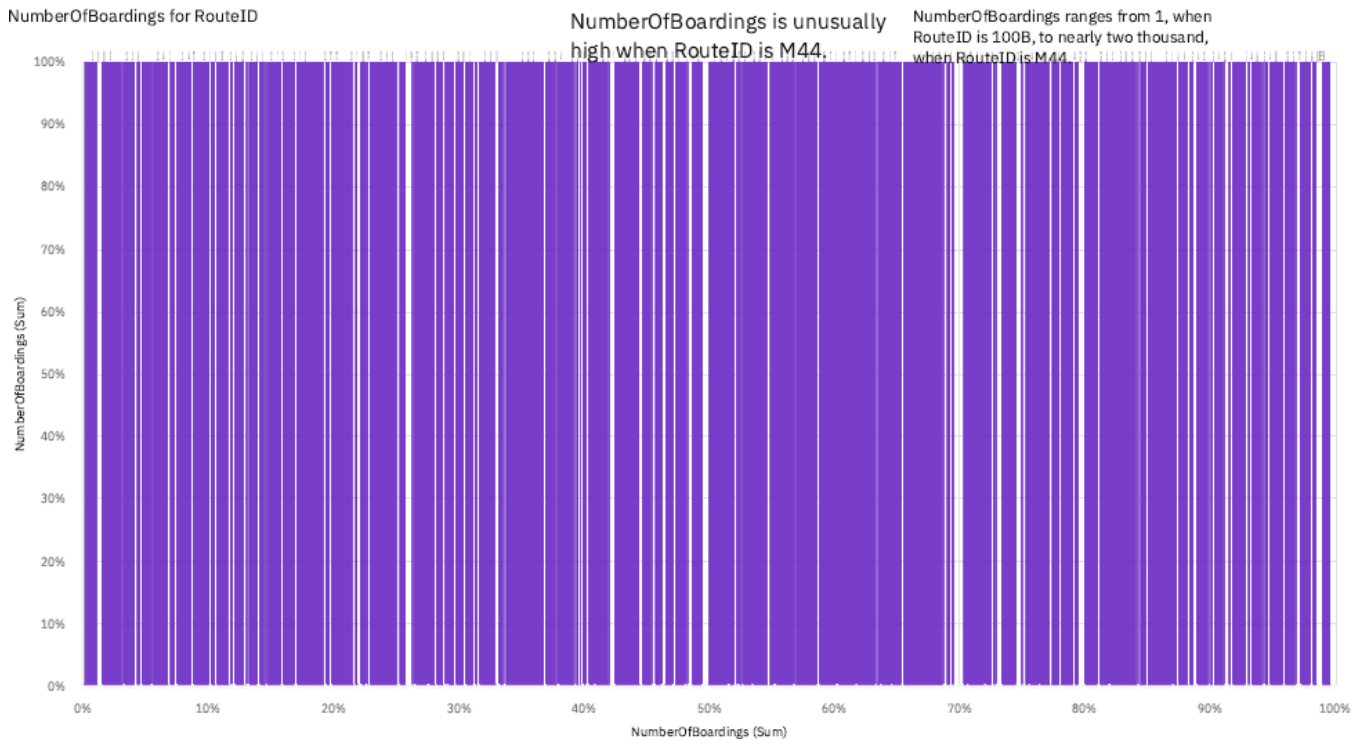


---

6. Line Graph of NumberOfBoardings by WeekBeginning

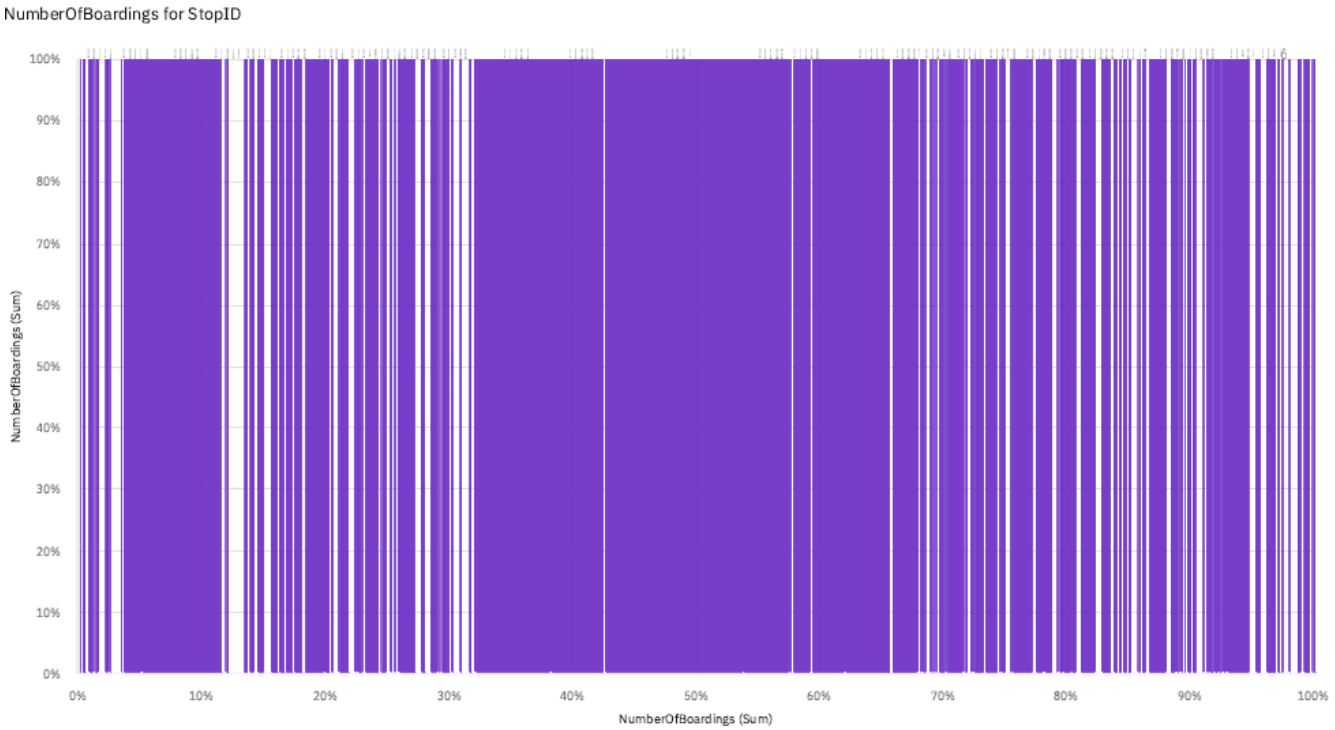


7. Waterfall Plot for NumberofBoardings for RouteID



---

8. Waterfall Plot for NumberOfBoardings for StopID



DATA ANALYSIS

Some Important external data fields calculation

- **IsHoliday** Number of public holidays within that week
- **DistanceFromCentre** Distance measure from the city centre

For Calculating Distance between centre with other bus stops by using Longitude and Latitude we have used the Haversine formula

```
from math import sin, cos, sqrt, atan2, radians
def calc_dist(lat1,lon1):
    ## approximate radius of earth in km
    R = 6373.0
    dlon = radians(138.604801) - radians(lon1)
    dlat = radians(-34.921247) - radians(lat1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(-34.921247)) * sin(dlon
        /2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    return R * c

out_geo['dist_from_centre'] = out_geo[['latitude','longitude']].apply(lambda x: calc
_dist(*x), axis=1)


# Fill the missing values with mode
out_geo['type'].fillna('street_address',inplace=True)
out_geo['type'] = out_geo['type'].apply(lambda x: str(x).split(',')[-1])
out_geo['type'].unique()


def holiday_label (row):
    if row == datetime.date(2013, 9, 1) :
          return '1'
    if row == datetime.date(2013, 10, 6) :
          return '1'
    if row == datetime.date(2013, 12, 22) :
          return '2'
    if row == datetime.date(2013, 12, 29):
          return '1'
    if row  == datetime.date(2014, 1, 26):
          return '1'
    if row == datetime.date(2014, 3, 9):
          return '1'
    if row == datetime.date(2014, 4, 13) :
          return '2'
    if row == datetime.date(2014, 4, 20):
          return '2'
    if row == datetime.date(2014, 6, 8):
          return '1'
    return '0'


data['WeekBeginning'] = pd.to_datetime(data['WeekBeginning']).dt.date
data['holiday_label'] = data['WeekBeginning'].apply (lambda row: holiday_label(row))

data= pd.merge(data,out_geo,how='left',left_on ='StopName',right_on= 'input_string')
```

```
data = pd.merge(data, route, how='left', left_on = 'RouteID', right_on = 'route_id')

col = ['TripID','RouteID','StopID','StopName', 'WeekBeginning','NumberOfBoardings','
formatted_address','latitude','longitude','postcode','type','route_desc','dist_from_
centre','holiday_label']

data = data[col]

#saving the final dataset
data.to_csv('Weekly_Boarding.csv',index=False)
```

Aggregate the Data According to Weeks and Stop names

- **NumberOfBoardings_sum** Number of Boardings within particular week for each Bus stop
- **NumberOfBoardings_count** Number of times data is recorded within week
- **NumberOfBoardings_max** Maximum number of boarding done at single time within week

```
grouped = data.groupby(['StopName','WeekBeginning','type']).agg({'NumberOfBoardings'
          : ['sum', 'count','max']})
grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]

st_week_grp = pd.DataFrame(grouped).reset_index()
st_week_grp.shape
st_week_grp.head()


#Gathering only the Stop Name which having all 54 weeks of Data
st_week_grp1 = pd.DataFrame(st_week_grp.groupby ('StopName') ['WeekBeginning']
.count()).reset_index()

aa=list(st_week_grp1[st_week_grp1['WeekBeginning'] == 54]['StopName'])
bb = st_week_grp[st_week_grp['StopName'].isin(aa)]

# save the aggregate data
bb.to_csv('st_week_grp.csv', index=False)
```

DATA EXPLORATION

Total Having 1 Year of Data from date 2013-06-30 till 2014-07-06 in a Weekly interval based.

Having Total of 4165 Stops in South Australian Metropolitan Area.

```
data.nunique()

data.shape
data.columns

data.head(3)
data.isnull().sum()

data['WeekBeginning'].unique()
```

VISUALIZATION

```
fig,axrr=plt.subplots(3,2,figsize=(18,18))

data['NumberOfBoardings'].value_counts().sort_index().head(20).plot.bar(ax=axrr[0][0
])
data['WeekBeginning'].value_counts().plot.area(ax=axrr[0][1])
data['RouteID'].value_counts().head(20).plot.bar(ax=axrr[1][0])
data['RouteID'].value_counts().tail(20).plot.bar(ax=axrr[1][1])
data['type'].value_counts().head(5).plot.bar(ax=axrr[2][0])
data['type'].value_counts().tail(10).plot.bar(ax=axrr[2][1])
```

Inferences:

- More than 40 lakhs times only single person board from the bus stop.
- There are average of 1.8 lakhs people travel every week by bus in adelaide metropolitan area.
- G10,B10,M44,H30 are the most busiest routes in the city while FX8,FX3,FX10,FX1,FX2 are the least.
- Most of the Bus stops are Street_Address Type while there are very few which are store or post office.

```
data['postcode'].value_counts().head(20).plot.bar()
bb_grp = data.groupby(['dist_from_centre']).agg({'NumberOfBoardings': ['sum']}).rese
t_index()

bb_grp.columns = bb_grp.columns.get_level_values(0)
bb_grp.head()
bb_grp.columns

trace0 = go.Scatter(
    x = bb_grp['dist_from_centre'],
    y = bb_grp['NumberOfBoardings'],mode = 'lines+markers',name = 'X2 King William S
t')

data1 = [trace0]
layout = dict(title = 'Distance Vs Number of boarding',
              xaxis = dict(title = 'Distance from centre'),
              yaxis = dict(title = 'Number of Boardings'))
fig = dict(data=data1, layout=layout)
iplot(fig)
```

Inferences:

- As we move away from centre the number of Boarding decreases
- There are cluster of bus stops near to the main Adelaide city as oppose to outside.so that's why most of boardings are near to center.

PLOT USING PLOTLY

```
# for finding highest number of Boarding Bus stops
bb_grp = bb.groupby(['StopName']).agg({'NumberOfBoardings_sum': ['sum']}).reset_inde
x()['NumberOfBoardings_sum'].sort_values('sum')
bb_grp[1000:1005]
```

```
bb.groupby(['StopName']).agg({'NumberOfBoardings_sum': ['sum']}).reset_index().iloc[
[2325,1528,546,1043,1905]]
source_1 = bb[bb['StopName'] == 'X2 King William St'].reset_index(drop = True)
source_2 = bb[bb['StopName'] == 'E1 Currie St'].reset_index(drop = True)
source_3 = bb[bb['StopName'] == 'I2 North Tce'].reset_index(drop = True)
source_4 = bb[bb['StopName'] == 'F2 Grenfell St'].reset_index(drop = True)
source_5 = bb[bb['StopName'] == 'D1 King William St'].reset_index(drop = True)


trace0 = go.Scatter(
    x = source_1['WeekBeginning'],
    y = source_1['NumberOfBoardings_sum'],mode = 'lines+markers',name = 'X2 King Wil
liam St')
trace1 = go.Scatter(
    x = source_2['WeekBeginning'],
    y = source_2['NumberOfBoardings_sum'],mode = 'lines+markers',name = 'E1 Currie S
t')
trace2 = go.Scatter(
    x = source_3['WeekBeginning'],
    y = source_3['NumberOfBoardings_sum'],mode = 'lines+markers',name = 'I2 North Tc
e')
trace3 = go.Scatter(
    x = source_4['WeekBeginning'],
    y = source_4['NumberOfBoardings_sum'],mode = 'lines+markers',name = 'F2 Grenfell
St')
trace4 = go.Scatter(
    x = source_5['WeekBeginning'],
    y = source_5['NumberOfBoardings_sum'],mode = 'lines+markers',name = 'D1 King Wil
liam St')

data = [trace0,trace1,trace2,trace3,trace4]
layout = dict(title = 'Weekly Boarding Total',
              xaxis = dict(title = 'Week Number'),
              yaxis = dict(title = 'Number of Boardings'),
              shapes = [{# Holidays Record: 2013-09-01
'type': 'line','x0': '2013-09-01','y0': 0,'x1': '2013-09-02','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2013-10-07
'type': 'line','x0': '2013-10-07','y0': 0,'x1': '2013-10-07','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
               {# 2013-12-25
'type': 'line','x0': '2013-12-25','y0': 0,'x1': '2013-12-26','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 3,'dash': 'dashdot'},},
              {# 2014-01-27
'type': 'line','x0': '2014-01-27','y0': 0,'x1': '2014-01-28','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2014-03-10
'type': 'line','x0': '2014-03-10','y0': 0,'x1': '2014-03-11','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2014-04-18
'type': 'line','x0': '2014-04-18','y0': 0,'x1': '2014-04-19','y1': 18000,'line': {
'color': 'rgb(55, 128, 191)','width': 3,'dash': 'dashdot'},},
              {# 2014-06-09
'type': 'line','x0': '2014-06-09','y0': 0,'x1': '2014-06-10','y1': 18000,'line': {
        'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},])
fig = dict(data=data, layout=layout)
iplot(fig)
```

Inferences:

- X2 King William St and stop near to that are the most busiest stops in the city. which having number of boardings per week more than 10k.
- Vertical lines are the indicator of holidays which came within that week.
- Whenever there is any Public holiday that week period have less than average number of people travelled from bus.

```
source_6 = bb[bb['StopName'] == '57A Hancock Rd'].reset_index(drop = True)
source_7 = bb[bb['StopName'] == '37 Muriel Dr'].reset_index(drop = True)
source_8 = bb[bb['StopName'] == '18B Springbank Rd'].reset_index(drop = True)
source_9 = bb[bb['StopName'] == '27E Sir Ross Smith Av'].reset_index(drop = True)
source_10 = bb[bb['StopName'] == '46A Baldock Rd'].reset_index(drop = True)

trace0 = go.Scatter(
    x = source_6['WeekBeginning'],
    y = source_6['NumberOfBoardings_sum'],mode = 'lines+markers',name = '57A Hancock
Rd')
trace1 = go.Scatter(
    x = source_7['WeekBeginning'],
    y = source_7['NumberOfBoardings_sum'],mode = 'lines+markers',name = '37 Muriel D
r')
trace2 = go.Scatter(
    x = source_8['WeekBeginning'],
    y = source_8['NumberOfBoardings_sum'],mode = 'lines+markers',name = '18B Springb
ank Rd')
trace3 = go.Scatter(
    x = source_9['WeekBeginning'],
    y = source_9['NumberOfBoardings_sum'],mode = 'lines+markers',name = '27E Sir Ros
s Smith Av')
trace4 = go.Scatter(
    x = source_10['WeekBeginning'],
y = source_10['NumberOfBoardings_sum'],mode = 'lines+markers',name = '46A Baldock Rd
')

data = [trace0,trace1,trace2,trace3,trace4]
layout = dict(title = 'Weekly Boarding Total',
              xaxis = dict(title = 'Week Number'),
              yaxis = dict(title = 'Number of Boardings'),
              shapes = [{# Holidays Record: 2013-09-01
'type': 'line','x0': '2013-09-01','y0': 0,'x1': '2013-09-02','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2013-10-07
'type': 'line','x0': '2013-10-07','y0': 0,'x1': '2013-10-07','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2013-12-25
'type': 'line','x0': '2013-12-25','y0': 0,'x1': '2013-12-26','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 3,'dash': 'dashdot'},},
              {# 2014-01-27
'type': 'line','x0': '2014-01-27','y0': 0,'x1': '2014-01-28','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
              {# 2014-03-10
'type': 'line','x0': '2014-03-10','y0': 0,'x1': '2014-03-11','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},
```

```
            {# 2014-04-18
'type': 'line','x0': '2014-04-18','y0': 0,'x1': '2014-04-19','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 3,'dash': 'dashdot'},},
            {# 2014-06-09
'type': 'line','x0': '2014-06-09','y0': 0,'x1': '2014-06-10','y1': 80,'line': {
       'color': 'rgb(55, 128, 191)','width': 1,'dash': 'dashdot'},},])
fig = dict(data=data, layout=layout)
iplot(fig)
```

Inferences:

- Same decreasing affect of Holidays on number of people travelling through bus can be seen in other city bus stops also.
- The width of vertical blue line shows the number of holidays come within that week period.
- Two thickest blue lines shows Christmas and New year period while other one was easter & Good friday period.on both the occassion number of public holidays within week period was 3.

PLOT USING BUBBLY

```
bb1=bb.copy()

# Label encode the Date type for easy Plotting
le = LabelEncoder()
bb1['WeekBeginning'] = le.fit_transform(bb1['WeekBeginning'])

figure = bubbleplot(dataset=bb1, x_column='NumberOfBoardings_sum', y_column='NumberOfBoardings_count',
    bubble_column='StopName', time_column='WeekBeginning', size_column='NumberOfBoardings_max',
    color_column='type',
    x_title="Total Boardings", y_title="Frequency Of Boardings",show_slider=True,
    title='Adelaide Weekly Bus Transport Summary 2D',x_logscale=True, scale_bubble=2,height=650)

iplot(figure, config={'scrollzoom': True})
```

In the graph above, the size corresponds to the maximum number of people board at single time and the Total boardings and Frequency of boardings with stop name can be seen by hovering over the cursor on the bubbles.

The animated bubble charts convey a great deal of information since they can accomodate upto seven variables in total, namely:

- X-axis (Total Boardings per week)
- Y-axis (Frequency of Bus Boarding)
- Bubbles (Bus stop name)
- Time (in week period)
- Size of bubbles (maximum number of people board at single time)
- Color of bubbles (Type of Bus stop)

PROPOSITIONS

Rate of change in the traffic pattern in all different bus stops.

```
d=[]
for i in bb['StopName'].unique():
    d.append({'StopName': i,'Boarding_sum':np.sum(bb[bb['StopName'] == i]['NumberOfBoardings_sum'].pct_change())/54,
              'Boarding_count':np.sum(bb[bb['StopName'] == i]['NumberOfBoardings_count'].pct_change())/54,
              'Boarding_max':np.sum(bb[bb['StopName'] == i]['NumberOfBoardings_max'].pct_change())/54})
pct_chng = pd.DataFrame(d)

pct_chng['Boarding_sum'].nlargest(5)
pct_chng['Boarding_sum'].nsmallest(5)
pct_chng[pct_chng['Boarding_sum']<0].shape
pct_chng.iloc[[3110,2134,214,1538,1290]]
```

Inferences:

- These 5 stops W Grote St,52 Taylors Rd,13 Tutt Av,37A Longwood Rd,32A Frederick Rd having the largest percent increase.
- There are 27 Bus stops where number of boardings have decreased.
- The number of busses can be found by taking the number of boarding divided by bus capacity which can take as 50.

PREDICTIVE MODELLING

Get info like RouteID, latitude, longitude, postcode, dist_from_centre & holiday_label 6 features from the main dataset.

```
bb1 = pd.merge(bb, out_geo, how='left', left_on = 'StopName', right_on = 'input_string')
bb1['holiday_label'] = bb1['WeekBeginning'].apply (lambda row: holiday_label(row))

#Final 11 features have been used for the forecastng.
cols = ['StopName','WeekBeginning','type_x','NumberOfBoardings_sum','NumberOfBoardings_count','NumberOfBoardings_max','latitude','longitude','postcode','dist_from_centre','holiday_label']
bb1=bb1[cols]
bb1.shape
bb1.head()

#Replace all Nan by Mode
for i in bb1.columns:
    bb1[i].fillna(bb1[i].mode()[0], inplace=True)
bb1[["postcode", "holiday_label"]] = bb1[["postcode", "holiday_label"]].apply(pd.to_numeric)
```

Label Encode the Categorical data

```
le = LabelEncoder()
bb1['StopName'] = le.fit_transform(bb1['StopName'])
bb1['type_x'] = le.fit_transform(bb1['type_x'])
```

Split into Train Test for Modelling further

```
train = bb1[bb1['WeekBeginning'] < datetime.date(2014, 6, 1)]
test = bb1[bb1['WeekBeginning'] >= datetime.date(2014, 6, 1)]
train.shape
test.shape

le = LabelEncoder()
train['WeekBeginning'] = le.fit_transform(train['WeekBeginning'])
test['WeekBeginning'] = le.fit_transform(test['WeekBeginning'])

tr_col = ['StopName', 'WeekBeginning', 'type_x', 'latitude',
       'longitude', 'postcode', 'dist_from_centre', 'holiday_label']
train_sum_y = train[['StopName','NumberOfBoardings_sum']]
train_count_y = train[['StopName','NumberOfBoardings_count']]
train_max_y = train[['StopName','NumberOfBoardings_max']]
train_x = train[tr_col]
test_x = test[tr_col]

test_sum_y = test[['StopName','NumberOfBoardings_sum']]
test_count_y = test[['StopName','NumberOfBoardings_count']]
test_max_y = test[['StopName','NumberOfBoardings_max']]
```

MODELLING USING REGRESSION MODELS

USING LightGbm

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=700, min_samples_leaf=3, max_features=0.5
,n_jobs=-1)
model.fit(train_x.values,train_sum_y['NumberOfBoardings_sum'].values)
preds = model.predict(test_x.values)

rms = sqrt(mean_squared_error(test_sum_y['NumberOfBoardings_sum'].values, preds))
rms
```

```
test_sum_y.values[:15]
preds[:15]

fig, ax = plt.subplots(figsize=(6,10))
lgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
ax.grid(False)
plt.title("LightGBM - Feature Importance", fontsize=15)
plt.show()
plt.figure(figsize=(15,5))
plt.plot(test_sum_y['NumberOfBoardings_sum'].values, label='true')
plt.plot(preds, label='pred')
plt.ylabel("Total Number of Boarding")
plt.xlabel("Index")
```

```
plt.title("Comparison Between Prediction & True Values")
plt.legend()
plt.show()
```

Results

- We have trained the model for 48 weeks and test on last 6 weeks for all stopping points.
- High Rmse value came because we didn't scale the values.so we got the actual prediction instead of scaled prediction