

DATA VISUALIZATION INSIGHTS – HANDS-ON BOOK

Sharath Kumar Jagannathan

CREATIVE COMMONS ATTRIBUTION NONCOMMERCIAL SHAREALIKE

This book is an insightful guide on the art and science of data visualization. It delves into the various types and dimensions of data, exploring the most effective chart types for visualization. The book emphasizes practical application, including exercises and case studies, with a focus on tools like Python and R programming. It is designed to transform complex data into comprehensible and actionable insights, catering to both beginners and advanced users in the field of data visualization.

PhD. Le Toan Thang

AI and Akashic Records with Data Scientist

Contents

Data Visualization Insights – Hands-on Book

I. Main Body

1. Introduction
2. Data Visualization in Python
3. Visualization using R Programming
4. Connecting R and Tableau
5. Tableau Visualization
6. Knime Analytics Platform
7. Knime Analytics - Case Studies
8. Machine Learning - Regression Model Using Designer Studio
9. Multiple regression models to predict car prices
10. Text Classification pipeline in Azure Machine Learning designer
11. Azure Machine Learning pipeline for image classification
12. Multi Dataset classifier problem using Designer Studio
13. Python scripts to predict credit risk using designer studio

Book Information

Book Description

“Data Visualization Insights – Hands-on Book” by Dr. Sharath Kumar is a thorough guide that equips readers with the skills to effectively visualize and interpret data. The book comprehensively explores various data types, including spatial, vector, and raster data, and elaborates on the use of diverse chart types for effective data representation. It emphasizes multivariate analysis and forecasting, crucial for making informed decisions in data-driven fields. Practical applications are a core component, with detailed exercises and case studies that bridge theory and real-world applications. This book is a must-read for anyone seeking to harness the power of data visualization in their professional or academic endeavors.

Author

Sharath Kumar Jagannathan

License



[Data Visualization Insights - Hands-on Book](#) Copyright © 2023 by Saint Peter's University is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#), except where otherwise noted.

Subject

Data science and analysis

Data Visualization Insights – Hands-on Book

Dr. SHARATH KUMAR JAGANNATHAN

Assistant Professor

Data Science Institute – Saint Peter’s University

This book is an insightful guide on the art and science of data visualization. It delves into the various types and dimensions of data, exploring the most effective chart types for visualization. The book emphasizes practical application, including exercises and case studies, with a focus on tools like Python and R programming. It is designed to transform complex data into comprehensible and actionable insights, catering to both beginners and advanced users in the field of data visualization.

Chapter 1

Introduction

The term “data visualization” refers to the process of displaying information and data in a graphical style, typically through the use of charts, graphs, and maps. Analytics refers to the scientific method of converting data into insights for the purpose of improving decision-making.

The most important objective of data visualization is to convey information in an understandable and efficient manner through graphical representations. The process of representing data in a manner that makes it simple to comprehend as well as manipulate and, as a result, makes the information more useful is known as “information visualization.” The information can be made more understandable through the use of visualisation by assisting in the discovery of linkages within the data and supporting (or disproving) views about the data.

Data Type

Spatial Data : The data or information that specifies the geographic location of structures and boundaries on Earth, such as natural or constructed features, oceans, and more, is referred to as spatial data. Spatial data can be either data or information. Data that can be mapped is referred to as “spatial data,” and it is typically recorded in the form of coordinates and topology.

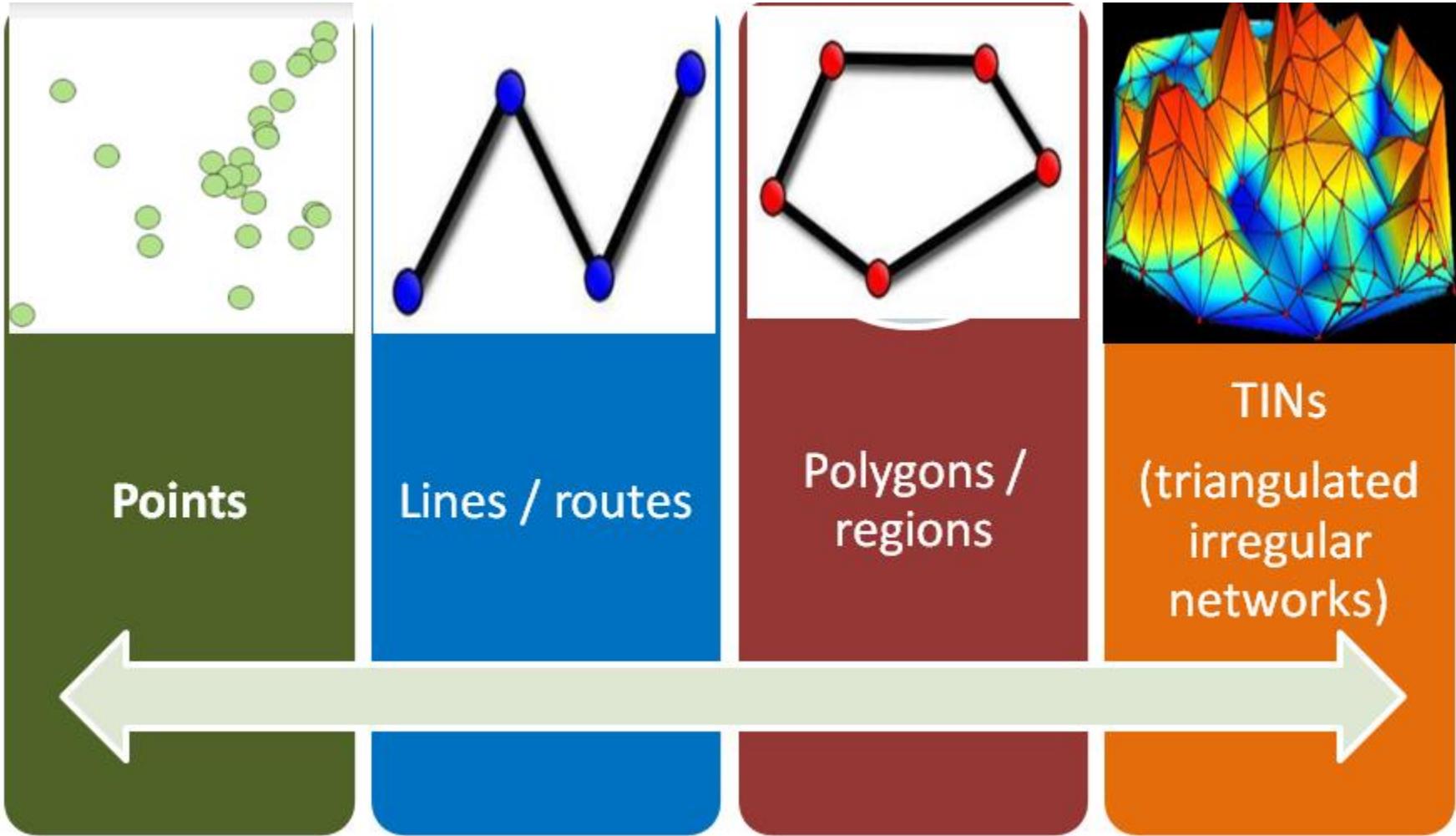
Types of Spatial Data :

There are 2 basic spatial data types namely Vector and Raster.



Vector Data:

In the GIS environment, vector data provide a means to represent real-world features. The shape of a vector feature is depicted geometrically. The geometry consists of multiple interconnected vertices. A vertex uses an x, y, and optionally z axis to characterize a position in space. The vector data model represents geographical features as shown in figure.

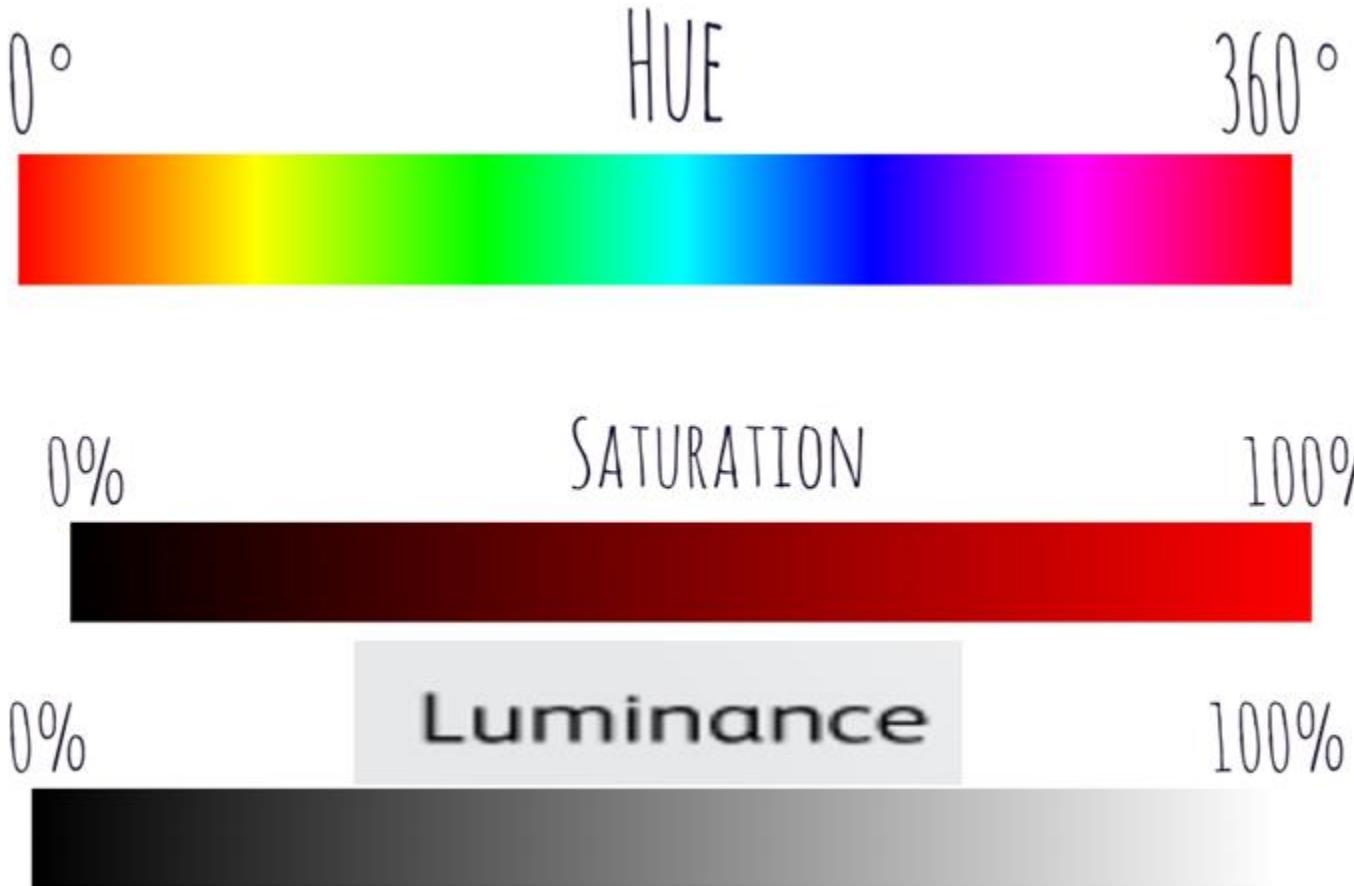


Color:

The best way to comprehend the color is through three distinct channels:

- Luminance
- Hue
- Saturation

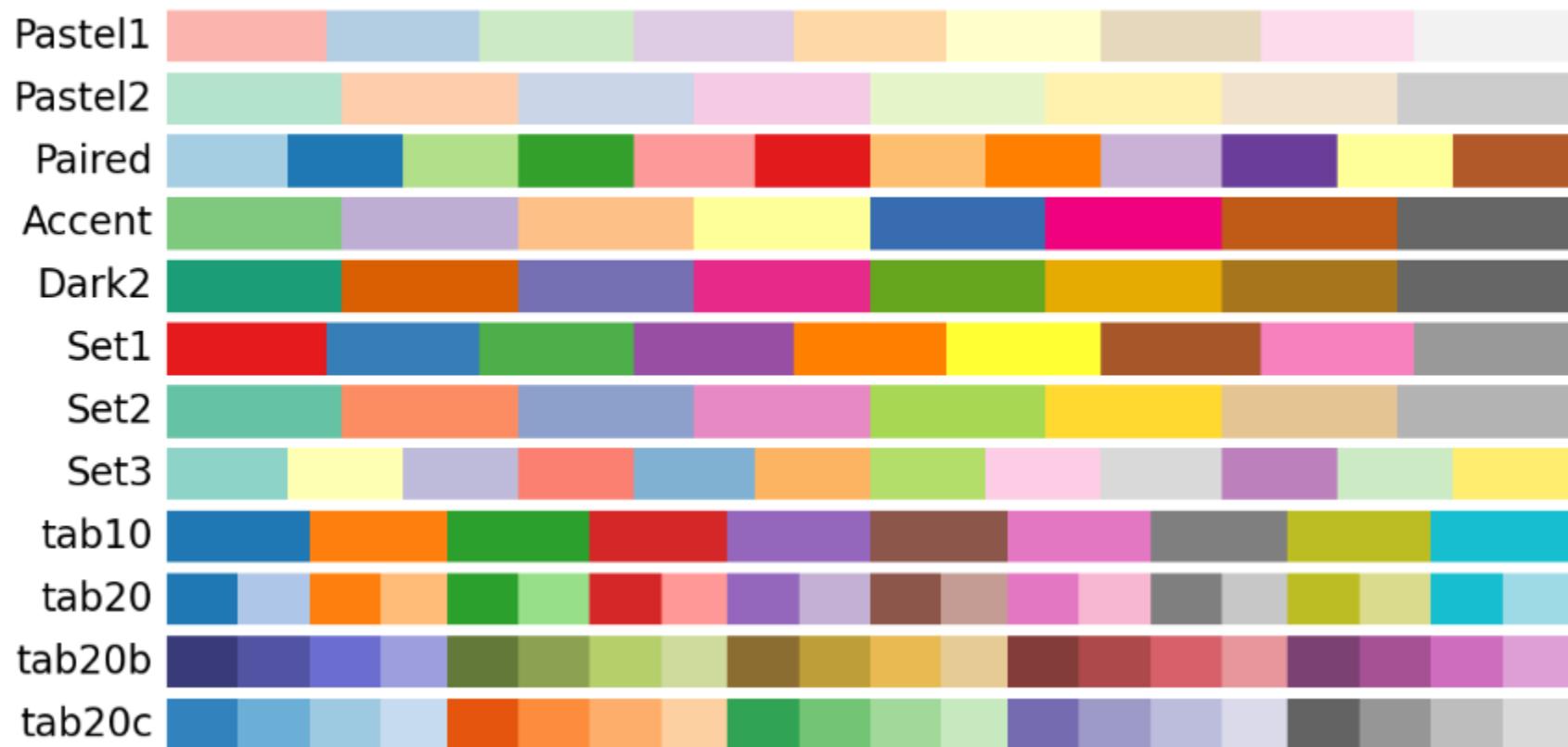
The hue channel is an identity channel, while the other two channels, luminance and saturation, are magnitude channels.



Colormaps

A colormap is a visual encoding that uses colour, and it specifies a mapping between different colours and different data values. The primary design decision that must be made before constructing a colormap is determining if the goal is to encode ordered qualities or to differentiate between categorical attributes. Colormap design offers numerous traps for the unwary, but it is a powerful and adaptable design choice. Colormaps can be ordered or categorised, and ordered colormaps can be sequential or divergent. In a categorical colormap, classifications and groupings are encoded through the use of colour. Segmentation is a common feature of categorical colormaps, they are also referred to as qualitative colormaps.

Qualitative colormaps



A colormap that is ordered is suitable for conveying ordinal or quantitative attributes. A sequential colormap varies in value from a minimum to a maximum. A diverging colormap has two hues at its extremities and a neutral or high-luminance colour, such as white, grey, or black, as its midpoint.

2. Chart Types for Data Visualization

Multivariate Analysis

Multivariate data analysis is a type of statistical analysis that involves more than two dependent variables, resulting in a single outcome. The variables in multivariate data analysis could be dependent or independent. It is important to verify the collected data and analyze the state of the variables. In multivariate data analysis, it is very important to understand the relationship between all the variables and predict the behavior of the variables based on observations.

Objectives of multivariate analysis

Multivariate data analysis (MVA) helps in the reduction and simplification of data as much as possible without losing any important details.

As MVA has multiple variables, the variables are grouped and sorted on the basis of their unique features.

It is tested to create a statistical hypothesis based on the parameters of multivariate data. This testing is carried out to determine whether or not the assumptions are true.

Example of multivariate data with case study:

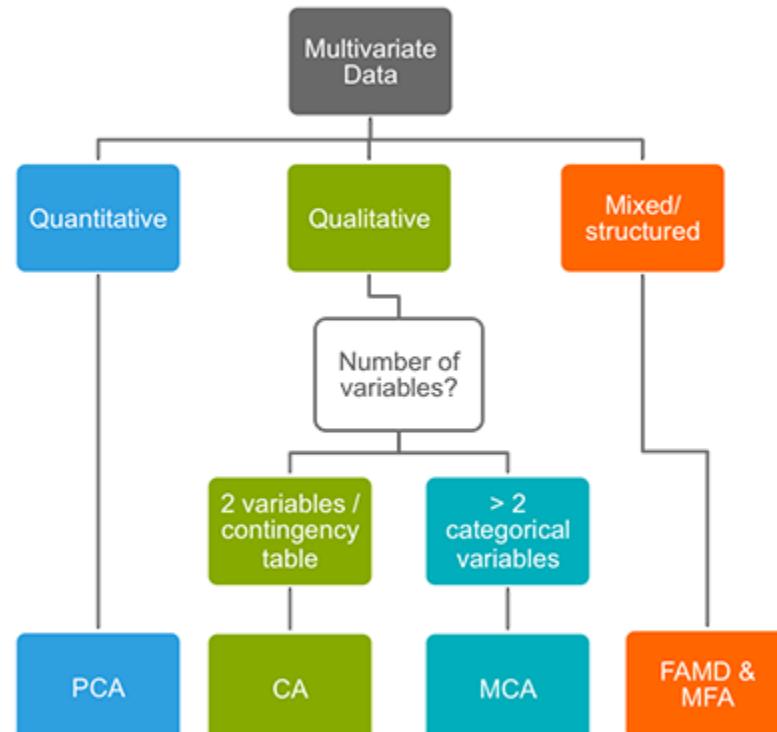
A famous Kaggle dataset is the **Titanic: Machine learning from disaster** dataset [\[link\]](#)

When researchers analyzed the titanic dataset using bivariate data analysis, it was found that the results are presumptuous since bivariate analysis assumes that the relationship between a variable X and the target variable Y is independent of the rest of the variables, (i.e) $f(X, Y)$ doesn't depend on a third variable Z. For instance, "**Women and children first**" is a naval code of conduct followed since 1852, whereby the lives of women and children were to be saved first in a life-threatening situation. As we already know, "Survival" is highly correlated with "Gender". But a third variable "Age" (child) influences the relationship between "Survival" and "Gender". This is where multivariate data analysis comes into play.

Approach to visualizing multivariate data:

DIMENSIONALITY REDUCTION

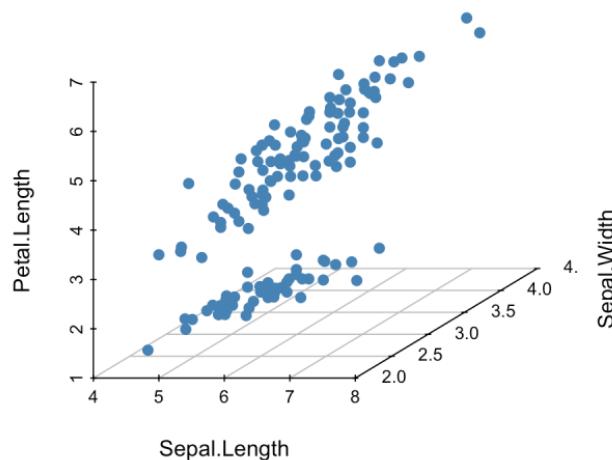
Methods to Summarize & Visualize Multivariate Data



- PCA: Principal Component Analysis
- (M) CA: (Multiple) Correspondence Analysis
- FAMD: Factor Analysis of Mixed Data
- MFA: Multiple Factor Analysis

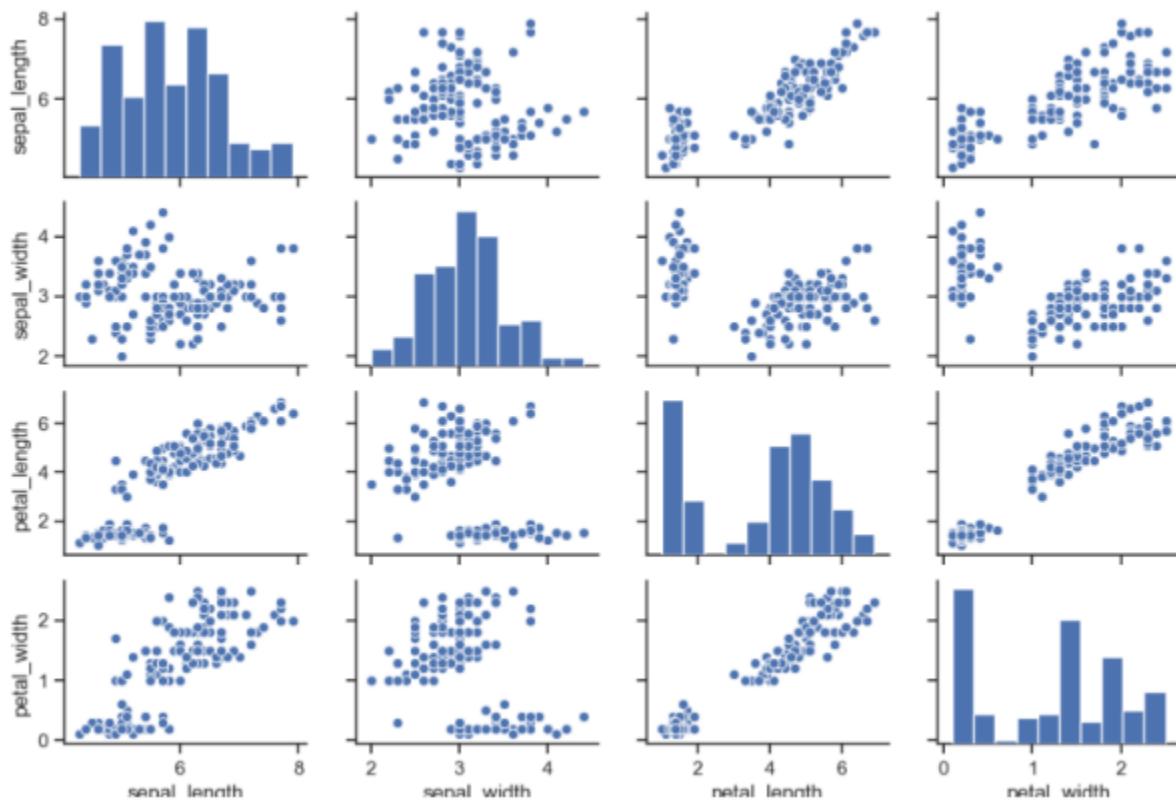
Scatter Plots / 3D Scatter plots:

Scatterplots are a way to visualize multivariate data to help classify and understand the relationships among the variables. A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.



Pair Plot

A pairplot plots pairwise relationships in a dataset. The pairplot function creates a grid of Axes such that each variable in data will be shared in the y-axis across a single row and in the x-axis across a single column.

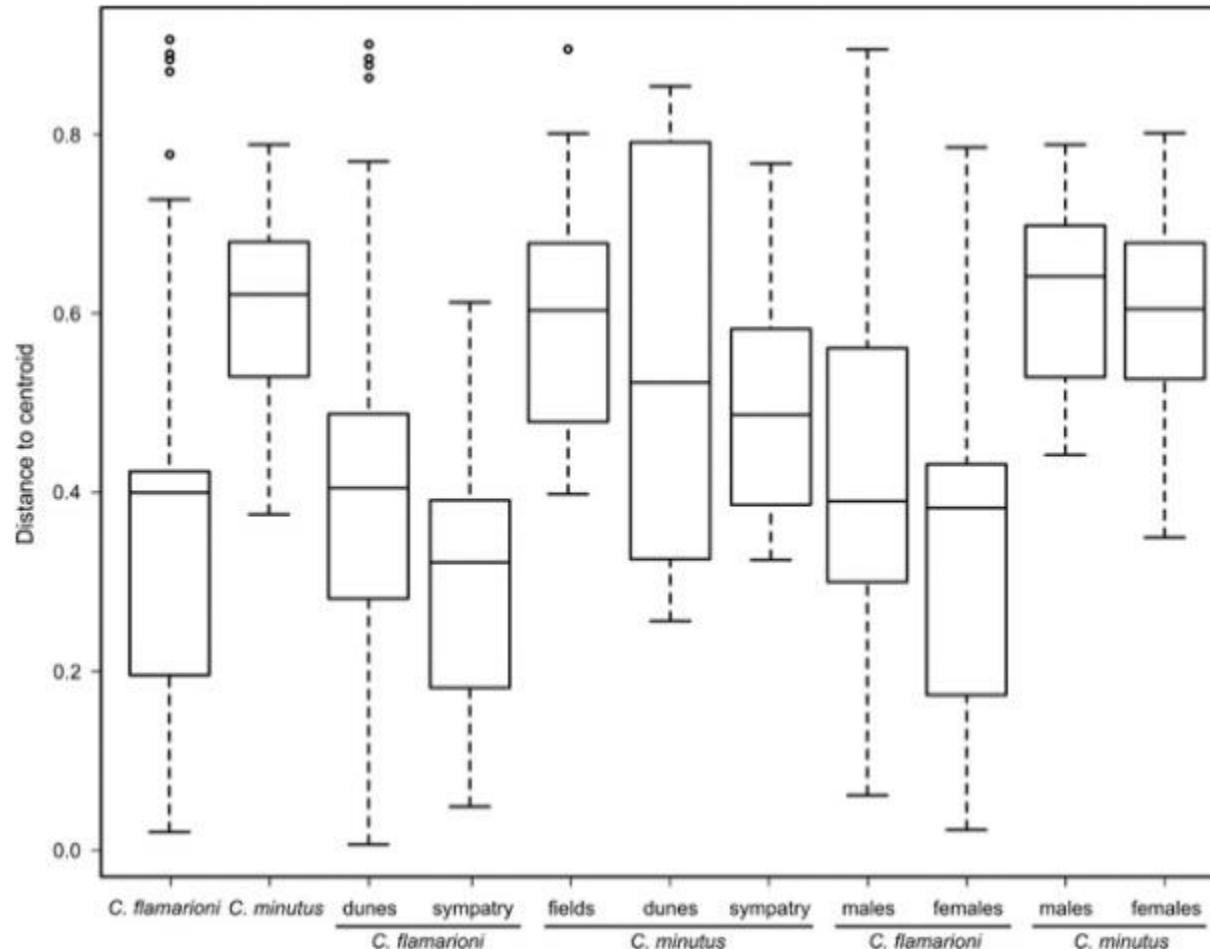


Boxplot:

Boxplots are a relatively condensed technique to visualise and summarise the primary qualities of a quantitative attribute. They do this through the utilisation of the median, the interquartile range, and any potential outliers. Two different characteristics are plotted against one another in a scatter plot. It is expandable in terms of its colour, shape, and size, among other characteristics. It works well with a limited number of points but does not perform well with a huge number of points. Add some Jitter, which is a little random value to each point, so that the points can hide one another.

Box plots for multivariate outlier detection

Multivariate outliers are typically examined when running statistical analyses with two or more independent or dependent variables. One of the first methods that can be used as a baseline for being able to detect outliers from multivariate datasets is that of boxplots and Tukey fences. They form a solid baseline for comparison against univariate and bivariate outlier analysis

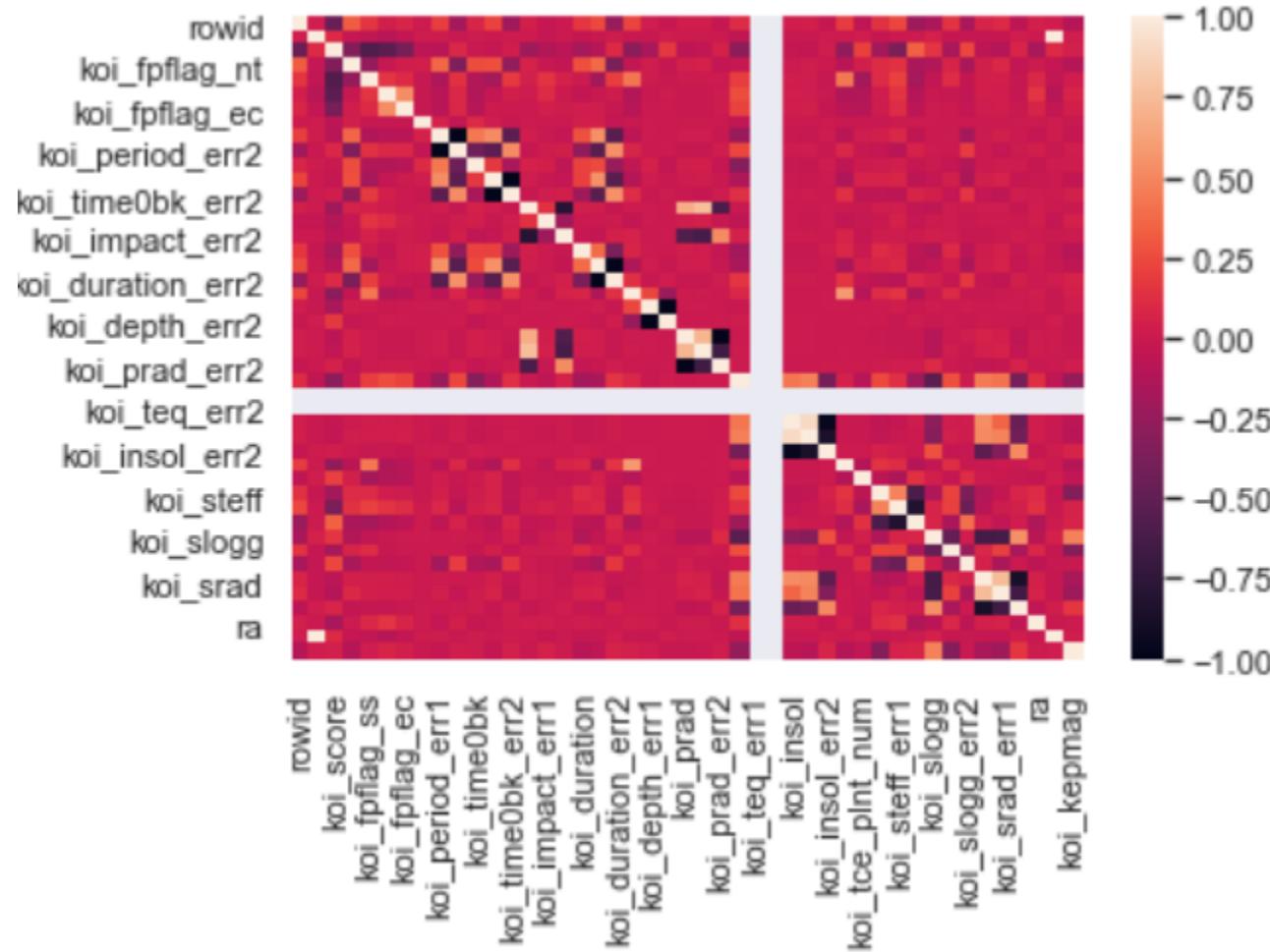


Histogram:

The frequency distribution of a numerical characteristic can be depicted using a histogram. The range of the numerical property is partitioned into a predetermined number of intervals known as bins. These bins are typically the same size. The height of a bar indicates the (absolute) frequency of values that fall into each interval, and this is represented by the interval itself.

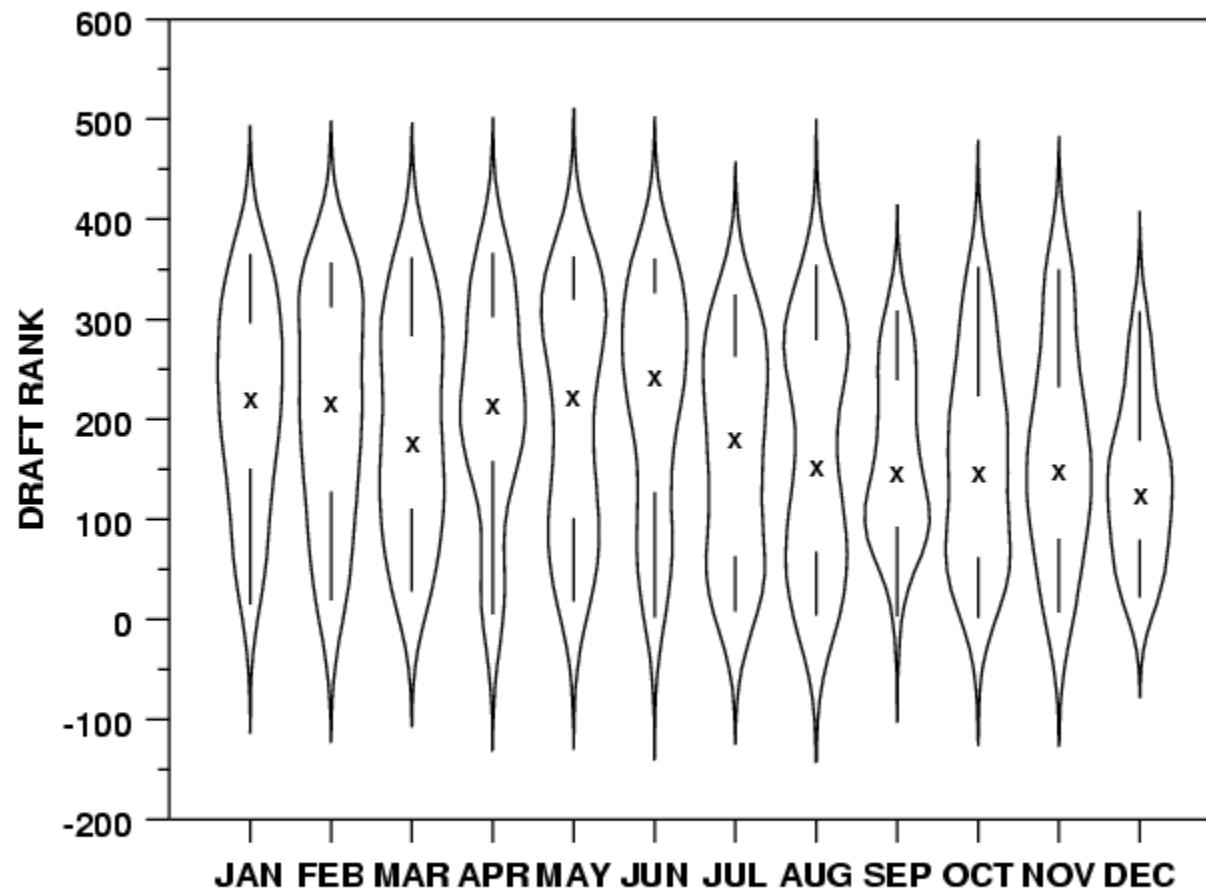
Correlation HeatMap:

A correlation heatmap is a heatmap that shows a 2D correlation matrix between two discrete dimensions, using coloured cells to represent data from usually a monochromatic scale. The values of the first dimension appear as the rows of the table while of the second dimension as a column.



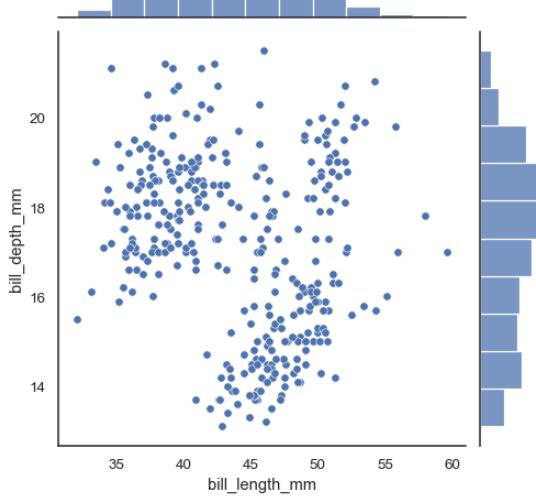
Violin Plots

- A violin plot is a method of plotting multivariate numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side. They show the probability density of the data at different values, usually smoothed by a kernel density estimator.



Joint Plots

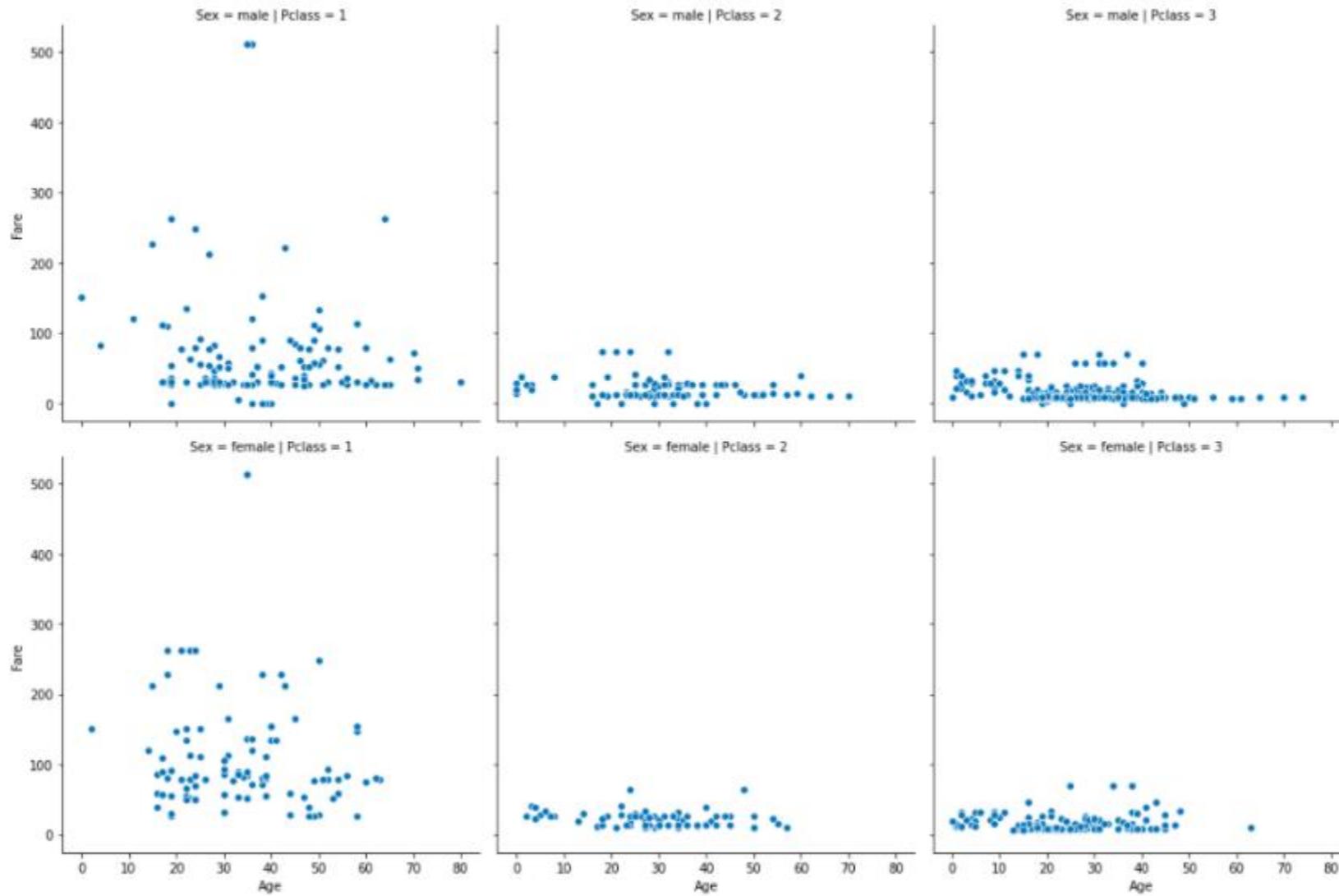
A Jointplot usually comprises of three plots. Out of the three, one plot displays a bivariate graph which shows how the dependent variable(Y) varies with the independent variable(X). Another plot is placed horizontally at the top of the bivariate graph and it shows the distribution of other influencing variables.



- **Relational Plots**

Relational plots are used for visualizing the statistical relationship between multiple data points. These plots can be used to identify the statistic relationship between all the factors considered for multivariate data analysis.

<seaborn.axisgrid.FacetGrid at 0x11fa3fee0>



Due to the inherent nature of the two-dimensional nature of a display or plot, there is a limit of no more than two axes that can be included. Utilising 3D methodologies allows for the incorporation of three axes, or qualities.

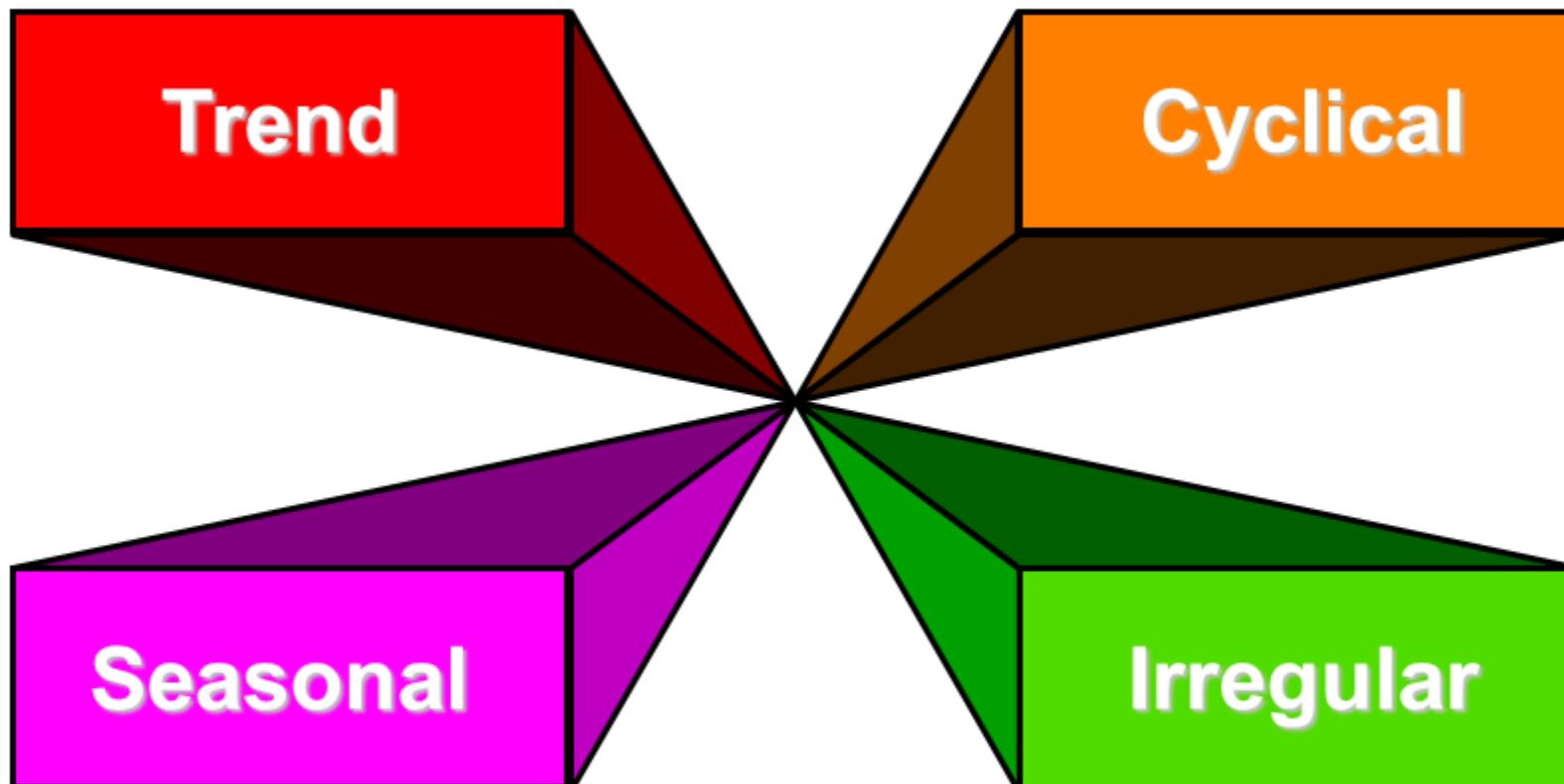
4. Forecasting

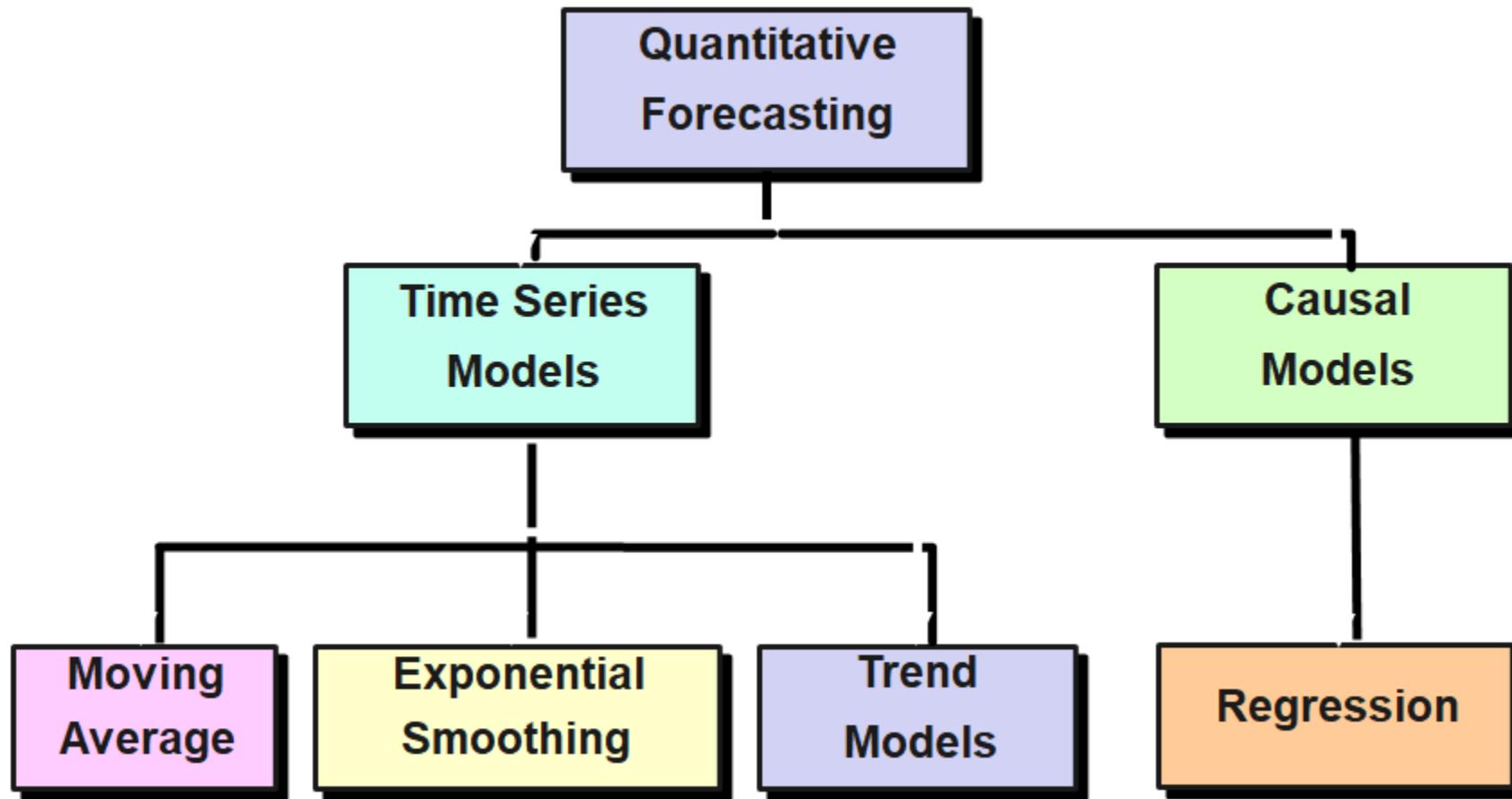
Almost any business that deals with numerical data uses time series forecasting.

Qualitative Methods vs Quantitative Methods

Qualitative Methods	Quantitative Methods
Used when situation is vague & little data exist	Used when situation is 'stable' & historical data exist
New products	Existing products
New technology	Current technology
Involve intuition, experience	Involve mathematical techniques
e.g., forecasting sales on Internet	e.g., forecasting sales of color televisions

Time Series Components





Tools for Data Visualization Experiments, Case studies and Real Time Applications

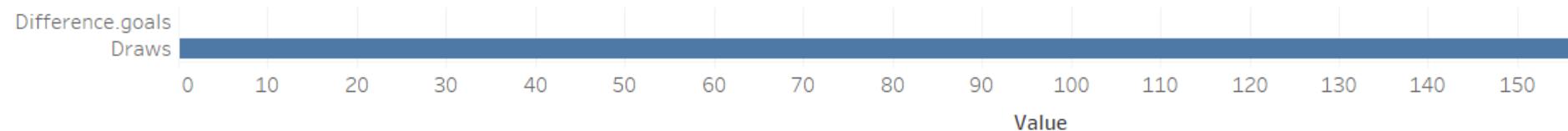
- Tableau
- Python
- R programming
- Knime Analytics Platform

TABLEAU

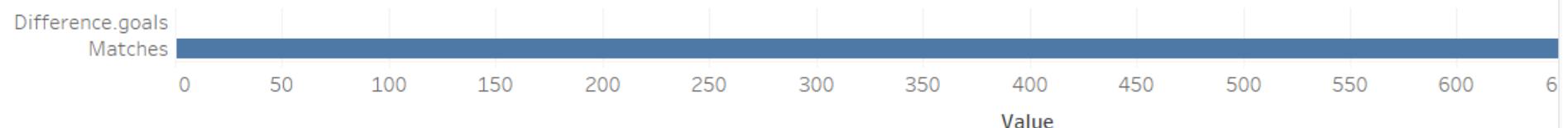
Problem and Solutions:

Create Sheets (different charts) with one Dashboard in tableau using the la_liga-2015-2016 dataset. Apply K means clustering also (integration of R and Tableau)

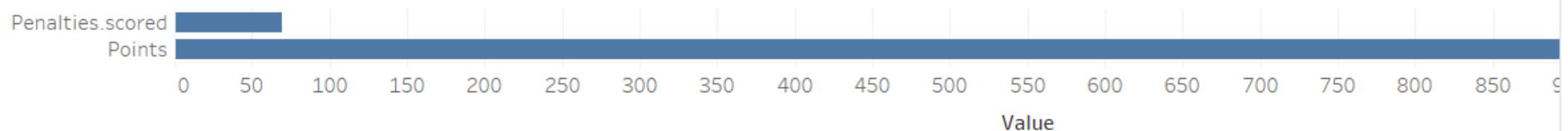
Sheet 1



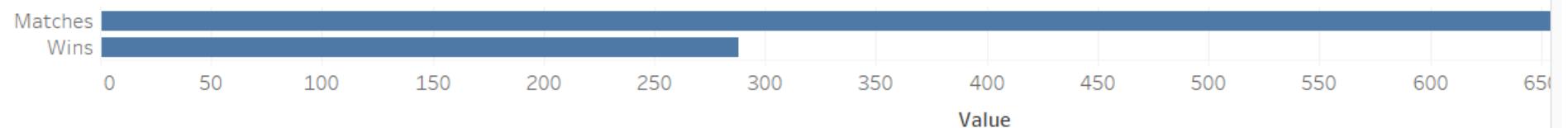
Sheet 2



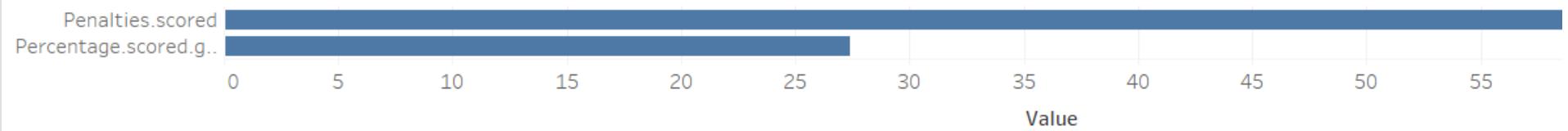
Sheet 3



Sheet 4



Sheet 5



Kmeans

kmeans

X

Results are computed along Table (across).

```
SCRIPT_INT('set.seed(42);
result <- kmeans(data.frame(.arg1,.arg2,.arg3,.arg4), 2);
result$cluster;
SUM([Penalties.scored]), SUM([Points]))
```

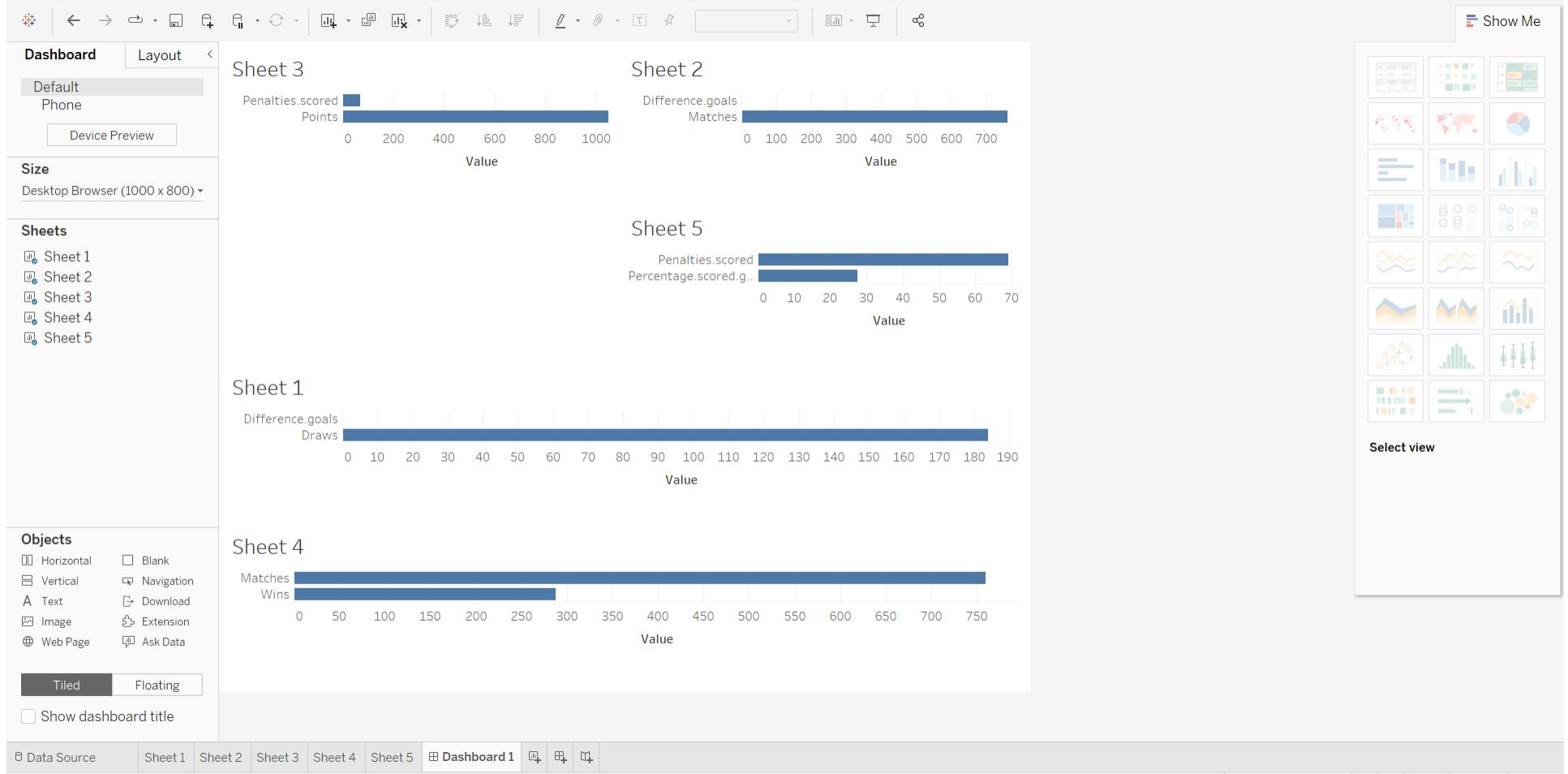


Default Table Calculation

The calculation is valid.

Apply

OK



Do it Yourself -Practice Problems

Exercise: You need to group or classify the given dataset based on the age, yearly income, cars and No. of children using R and integrate with tableau to visualize your output.

Analysis of Bike Buyer Data:

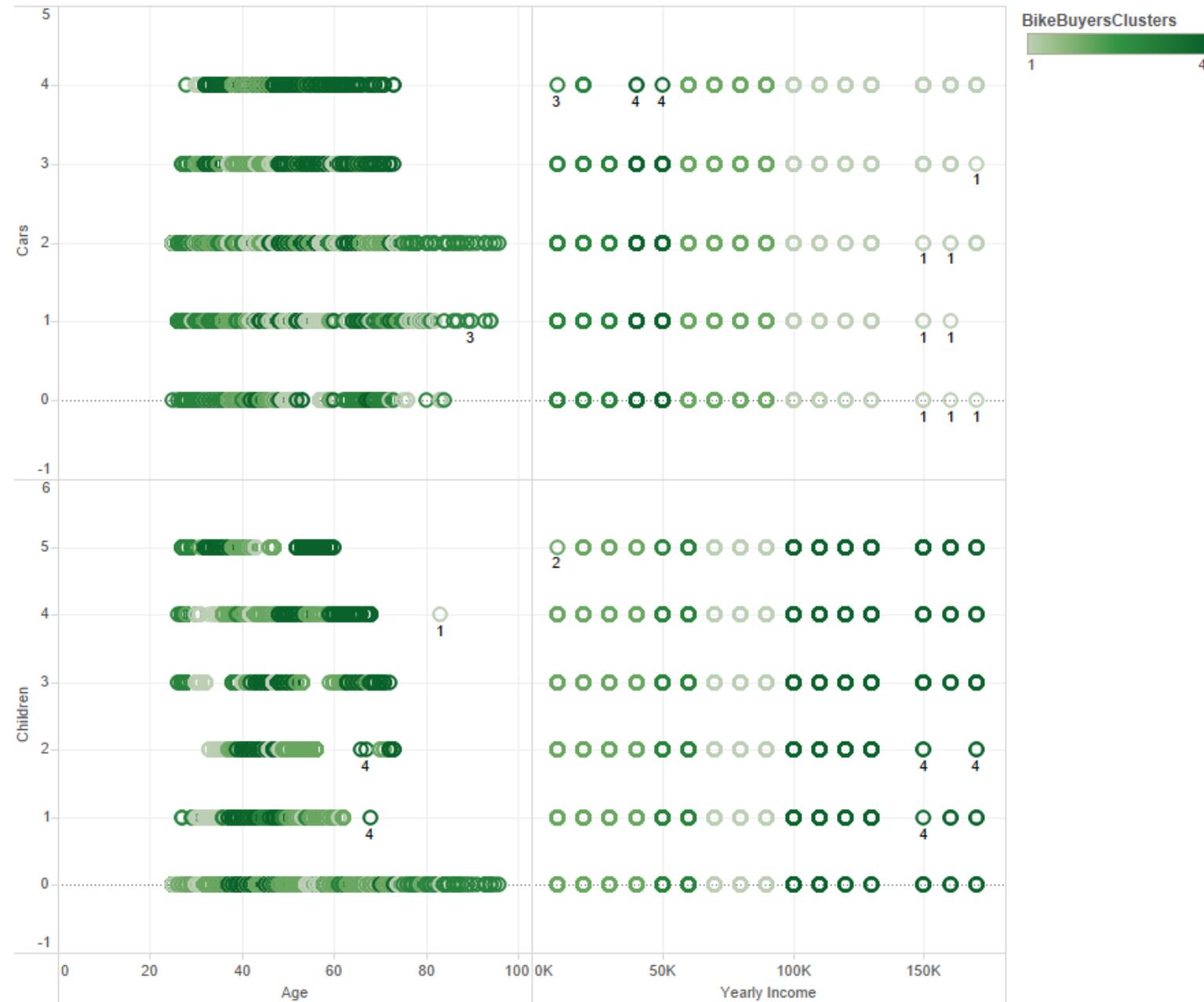
Cluster based on age, yearly income, cars and children or Classify: Use R script and integrate with tableau.

Sample Scripts for cluster:

```
SCRIPT_INT('kmeans(data.frame(.arg1,.arg2,.arg3,.arg4),4)$cluster;SUM([Age]),SUM([Cars]),SUM([Children]),SUM([Yearly Income]))
```

Sample results:

Sheet 1



Age and Yearly Income vs. Cars and Children. Color shows BikeBuyersClusters. The marks are labeled by BikeBuyersClusters.

Chapter 2

Data Visualization in Python

Libraries:

Python provides a number of plotting libraries, including Matplotlib, Seaborn, and numerous other data visualisation utilities with various features for creating informative, customized, and visually appealing plots to present data in the simplest and most effective manner.

Exercise using Python:

Visualizing the malaria in Africa dataset and modelling the expected number of malaria cases using Decision Tree Regression.

Introduction:

Africa, the world's second-largest continent, a continent with a wide array of vibrant cultures each with its own deep history, continent number 2 of largest population, and the continent is home to wonderful wildlife you can spot when you go on safari. Malaria is a common disease in Africa. The disease is transmitted to humans through infected mosquito bites. Although you can take preventive measures against malaria, it can be life-threatening. This dataset includes the malaria cases in African countries, the incidence at risk, and data on preventive treatments against malaria.

DATASET:

Link: <https://www.kaggle.com/lydia70/malaria-in-africa>

- Contain 594 rows and 27 columns

TOOLS:

Language: Python

Libraries: numpy, pandas, matplotlib, seaborn etc...

```
[1] # Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score

pio.templates.default = 'plotly_dark'
```

Reading Dataset

```
[2] # Import dataset into pandas dataframe
df = pd.read_csv('https://raw.githubusercontent.com/ahmedmeshref/Malaria-in-Africa-Analysis/main/DatasetAfricaMalaria.csv')
```

Data Exploration

```
[3] # Show first 5 rows in df
pd.set_option('display.max_columns', None)
df.head(5)
```

Country Name	Year	Country Code	Incidence of malaria (per 1,000 population at risk)	Malaria cases reported	Use of insecticide-treated bed nets (% of under-5 population)	Children with fever receiving antimalarial drugs (% of children under age 5 with fever)	Intermittent preventive treatment (IPT) of malaria in pregnancy (% of pregnant women)	People using managed drinking water services (% of population)	People using safely managed drinking water services, rural (% of rural population)	People using safely managed drinking water services, urban (% of urban population)	People using sanitation services, rural (% of rural population)	People using sanitation services, urban (% of urban population)	People using safely managed sanitation services, rural (% of rural population)	People using safely managed sanitation services, urban (% of urban population)
0	Algeria	2007	DZA	0.01	26.0	NaN	NaN	NaN	NaN	NaN	18.24	19.96	17.33	
1	Angola	2007	AGO	286.72	1533485.0	18.0	29.8	1.5	NaN	NaN	NaN	NaN	NaN	NaN
2	Benin	2007	BEN	480.24	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
[4] # Print all African countries in the dataset
afr_countries = df['Country Name'].unique()
print(f'Number of countries: {len(afr_countries)}')
print(f'Countries: {afr_countries}' )
```

Number of countries: 54

Countries: ['Algeria' 'Angola' 'Benin' 'Botswana' 'Burkina Faso' 'Burundi' 'Cabo Verde' 'Cameroon' 'Central African Republic' 'Chad' 'Comoros' 'Congo, Dem. Rep.' 'Congo, Rep.' "Cote d'Ivoire" 'Djibouti' 'Egypt, Arab Rep.' 'Equatorial Guinea' 'Eritrea' 'Eswatini' 'Ethiopia' 'Gabon' 'Gambia, The' 'Ghana' 'Guinea' 'Guinea-Bissau' 'Kenya' 'Lesotho' 'Liberia' 'Libya' 'Madagascar' 'Malawi' 'Mali' 'Mauritania' 'Mauritius' 'Morocco' 'Mozambique' 'Namibia' 'Niger' 'Nigeria' 'Rwanda' 'Sao Tome and Principe' 'Senegal' 'Seychelles' 'Sierra Leone' 'Somalia' 'South Africa' 'South Sudan' 'Sudan' 'Tanzania' 'Togo' 'Tunisia' 'Uganda' 'Zambia' 'Zimbabwe']

```
# Print all columns  
cols = df.columns  
for ind in range(len(cols)):  
    print(f'{ind+1}- {cols[ind]}')
```

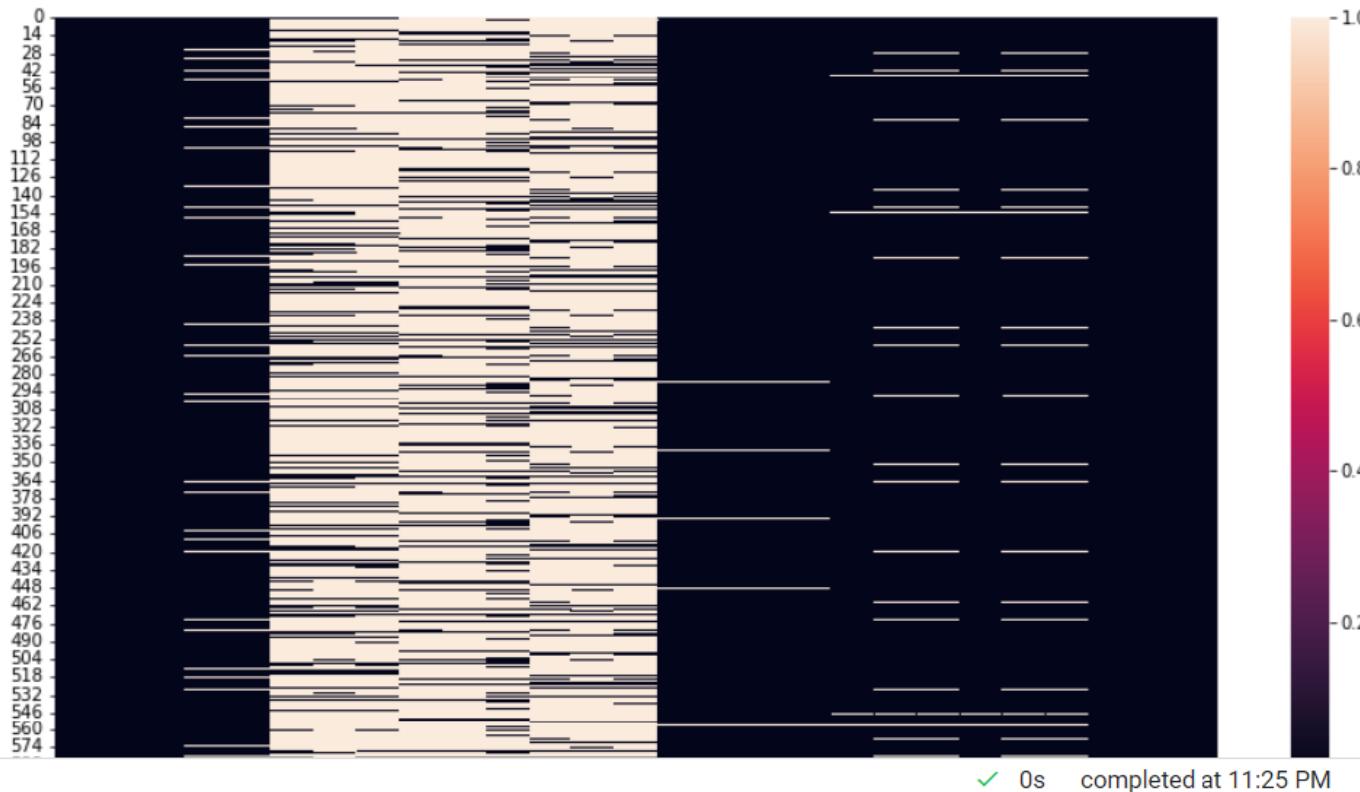
1- Country Name
2- Year
3- Country Code
4- Incidence of malaria (per 1,000 population at risk)
5- Malaria cases reported
6- Use of insecticide-treated bed nets (% of under-5 population)
7- Children with fever receiving antimalarial drugs (% of children under age 5 with fever)
8- Intermittent preventive treatment (IPT) of malaria in pregnancy (% of pregnant women)
9- People using safely managed drinking water services (% of population)
10- People using safely managed drinking water services, rural (% of rural population)
11- People using safely managed drinking water services, urban (% of urban population)
12- People using safely managed sanitation services (% of population)
13- People using safely managed sanitation services, rural (% of rural population)
14- People using safely managed sanitation services, urban (% of urban population)
15- Rural population (% of total population)
16- Rural population growth (annual %)
17- Urban population (% of total population)
18- Urban population growth (annual %)
19- People using at least basic drinking water services (% of population)
20- People using at least basic drinking water services, rural (% of rural population)
21- People using at least basic drinking water services, urban (% of urban population)
22- People using at least basic sanitation services (% of population)
23- People using at least basic sanitation services, rural (% of rural population)
24- People using at least basic sanitation services, urban (% of urban population)
25- latitude
26- longitude
27- geometry

[9] # Missing values per column
df.isnull().sum()

Country Name	0
Year	0
Country Code	0
Incidence of malaria (per 1,000 population at risk)	44
Malaria cases reported	44
Use of insecticide-treated bed nets (% of under-5 population)	462
Children with fever receiving antimalarial drugs (% of children under age 5 with fever)	472
Intermittent preventive treatment (IPT) of malaria in pregnancy (% of pregnant women)	488
People using safely managed drinking water services (% of population)	495
People using safely managed drinking water services, rural (% of rural population)	506
People using safely managed drinking water services, urban (% of urban population)	418
People using safely managed sanitation services (% of population)	462
People using safely managed sanitation services, rural (% of rural population)	484
People using safely managed sanitation services, urban (% of urban population)	462
Rural population (% of total population)	6
Rural population growth (annual %)	6
Urban population (% of total population)	6
Urban population growth (annual %)	6
People using at least basic drinking water services (% of population)	6
People using at least basic drinking water services, rural (% of rural population)	28
People using at least basic drinking water services, urban (% of urban population)	28
People using at least basic sanitation services (% of population)	6
People using at least basic sanitation services, rural (% of rural population)	28
People using at least basic sanitation services, urban (% of urban population)	28
latitude	0
longitude	0
geometry	0
dtype: int64	

```
✓ [3s] plt.figure(figsize=(15,8))  
sns.heatmap(df.isnull())
```

```
↳ <matplotlib.axes._subplots.AxesSubplot at 0x7f467e150ed0>
```



Heapmap After Data Cleaning

Heapmap After Data Cleaning

✓ [22] plt.figure(figsize=(15,8))
sns.heatmap(clean_data.isnull())

↳ <matplotlib.axes._subplots.AxesSubplot at 0x7f467b73ead0>



✓ 0s completed at 11:25 PM

Representation geographique using the longitude and latitude

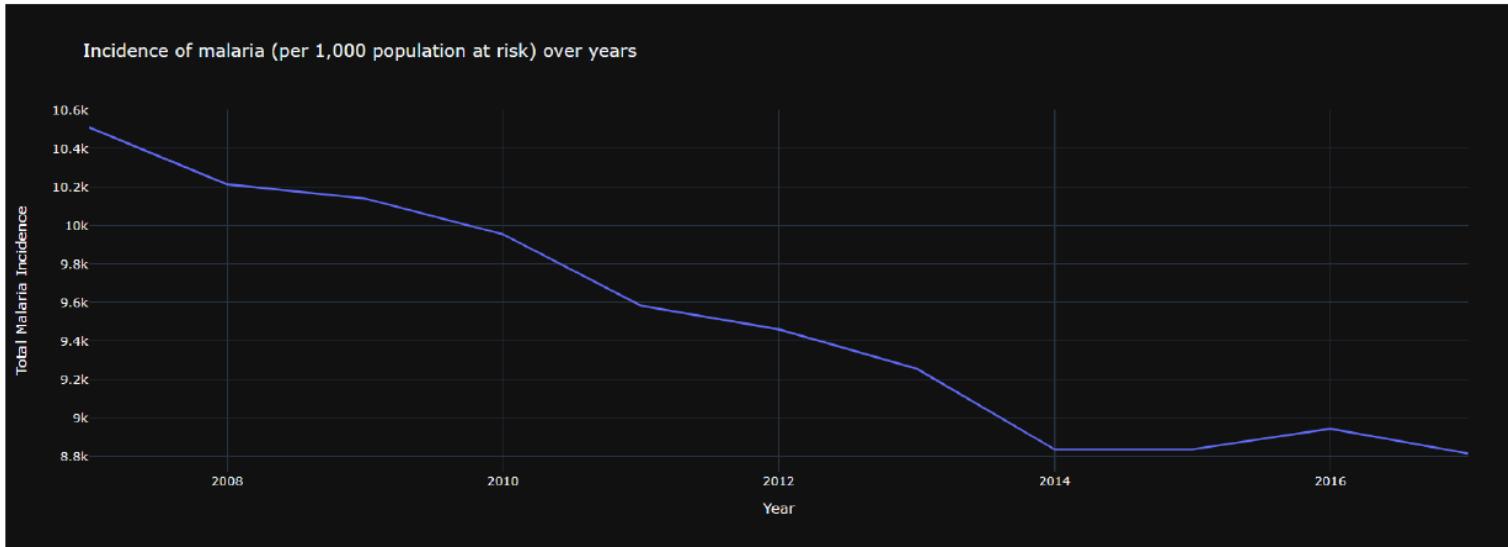
```
[38] import folium
    from folium import Choropleth, Circle, Marker
    from folium.plugins import HeatMap, MarkerCluster

    # Create a map
    m_2 = folium.Map(location=[9.000,18.000], tiles='cartodbpositron', zoom_start=3)

    # Add points to the map
    for idx, row in clean_data.iterrows():
        Marker([row['latitude'], row['longitude']]).add_to(m_2)

    # Display the map
    m_2
```





Dataset splitting and modelling using Decision Tree Regression

```
[39] # Splitting the label from other features
# Remove the year column
data = clean_data.drop(['Year', 'Country Name', 'Country Code', 'geometry'], axis=1)
# Split the target column
target = clean_data['Malaria cases reported']
print(f'Shape of features: {data.shape}, Shape of target: {target.shape}')

Shape of features: (539, 14), Shape of target: (539,)

[36] # Split training from the testing set
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape

((431, 14), (431,), (108, 14), (108,))

[37] # Training the data into the model
model = DecisionTreeRegressor()

y_pred = model.fit(X_train, y_train).predict(X_test)
print("The r-squared score of is: ", r2_score(y_pred, y_test)*100, "%")

The r-squared score of is: 92.91044215442655 %
```

RESULT:

Got an accuracy of 92.91% using decision tree regressor.

Chapter 3

Visualization using R Programming

R programming

Exercise with Solution

#1. Look at Orange using either head or as.tibble() (you'll have to run library(tidyverse) for that second option). What type of data are each of the columns?

Solution:

```
dataSet=Orange  
head(dataSet)  
  
## Tree age circumference  
## 1 1 118 30  
## 2 1 484 58  
## 3 1 664 87  
## 4 1 1004 115  
## 5 1 1231 120  
## 6 1 1372 142
```

#Dataset orange has 3 columns Tree,age,circumference with all integer values

#2. Find the mean, standard deviation, and standard error of tree circumference

Solution:

```
mean(dataSet$circumference)  
## [1] 115.8571  
  
sd(dataSet$circumference)  
## [1] 57.48818  
  
sd(dataSet$circumference)/sqrt(length(dataSet$circumference))  
## [1] 9.717276
```

#3. Make a linear model which describes circumference (the response) as a function of age (the predictor). Save it as an object with <-, then print the object out by typing its name. What do those coefficients mean?

Solution:

```
linearM=lm(dataSet$circumference ~ dataSet$age)
linearM

## 
## Call:
## lm(formula = dataSet$circumference ~ dataSet$age)
## 
## Coefficients:
## (Intercept) dataSet$age
## 17.3997 0.1068
```

Intercept variable tells us that where the linear models cut at y and the other coefficient is the slope

#4. Make another linear model describing age as a function of circumference. Save this as a different object.

Solution:

```
linearAge=lm(dataSet$age ~ dataSet$circumference)
linearAge

## 
## Call:
## lm(formula = dataSet$age ~ dataSet$circumference)
## 
## Coefficients:
## (Intercept) dataSet$circumference
## 16.604 7.816
```

#5. Call summary() on both of your model objects. What do you notice?

```
summary(linearM)

## 
## Call:
## lm(formula = dataSet$circumference ~ dataSet$age)
## 
## Residuals:
## Min 1Q Median 3Q Max
## -46.310 -14.946 -0.076 19.697 45.111
## 
## Coefficients:
```

```

## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.399650 8.622660 2.018 0.0518 .
## dataSet$age 0.106770 0.008277 12.900 1.93e-14 ***
## -
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
##
## Residual standard error: 23.74 on 33 degrees of freedom
## Multiple R-squared: 0.8345, Adjusted R-squared: 0.8295
## F-statistic: 166.4 on 1 and 33 DF, p-value: 1.931e-14

```

```
summary(linearAge)
```

```

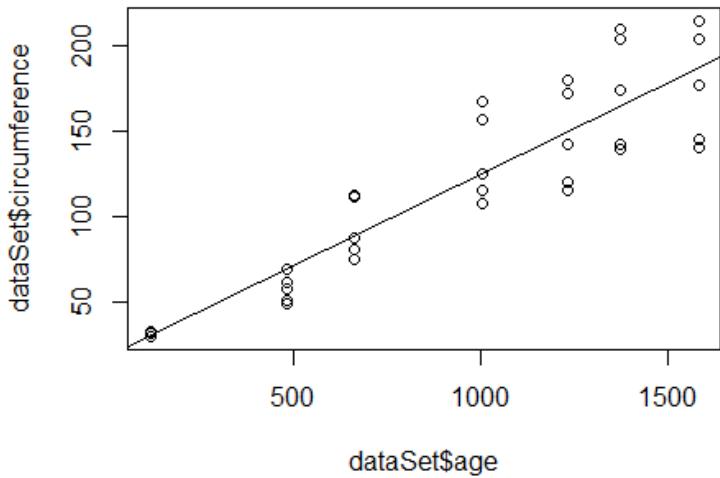
##
## Call:
## lm(formula = dataSet$age ~ dataSet$circumference)
##
## Residuals:
## Min 1Q Median 3Q Max
## -317.88 -140.90 -17.20 96.54 471.16
##
## Coefficients:
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 16.6036 78.1406 0.212 0.833
## dataSet$circumference 7.8160 0.6059 12.900 1.93e-14 ***
## -
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
##
## Residual standard error: 203.1 on 33 degrees of freedom
## Multiple R-squared: 0.8345, Adjusted R-squared: 0.8295
## F-statistic: 166.4 on 1 and 33 DF, p-value: 1.931e-14

```

#6. Does this mean that trees growing makes them get older? Does a tree getting older make it grow larger? Or are these just correlations?

Solution:

```
plot(dataSet$circumference ~ dataSet$age)
abline(linearM)
```



#its a correlation that older trees are generally bigger in size

#7. Does the significant p value prove that trees growing makes them get older? Why not?

Solution:

```
cor.test(dataSet$age, dataSet$circumference, method = "pearson")
##
## Pearson's product-moment correlation
##
## data: dataSet$age and dataSet$circumference
## t = 12.9, df = 33, p-value = 1.931e-14
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.8342364 0.9557955
## sample estimates:
## cor
## 0.9135189
```

#From the above, we get the conclusion that if the P-Value for our values is Significant then the values to #be randomised is significant. Hence, significant p value does not prove that trees growing makes them get #older.

Practice Problems

Forecasting

Problem 1:

Auto sales at Carmen's Chevrolet are shown below. Develop a 3-week moving average.

Week	Auto Sales
1	8
2	10
3	9
4	11
5	10
6	13
7	-

Problem 2:

Carmen's decides to forecast auto sales by weighting the three weeks as follows:

Weights Applied	Period
3	Last week
2	Two weeks ago
1	Three weeks ago
6	Total

Problem 3:

A firm uses simple exponential smoothing with to forecast demand. The forecast for the week of January 1 was 500 units whereas the actual demand turned out to be 450 units. Calculate the demand forecast for the week of January 8.

Problem 4:

Exponential smoothing is used to forecast automobile battery sales. Two values of are examined, and Evaluate the accuracy of each smoothing constant. Which is preferable? (Assume the forecast for January was 22 batteries.) Actual sales are given below:

Month	Actual Battery Sales	Forecast
January	20	22
February	21	
March	15	
April	14	
May	13	
June	16	

Problem 5:

Use the sales data given below to determine: (a) the least squares trend line, and (b) the predicted value for 2013 sales.

Year	Sales (Units)
2006	100
2007	110
2008	122
2009	130
2010	139
2011	152
2012	164

To minimize computations, transform the value of x (time) to simpler numbers. In this case, designate year 2006 as year 1, 2007 as year 2, etc.

Problem 6:

Given the forecast demand and actual demand for 10-foot fishing boats, compute the tracking signal and MAD.

Year	Forecast Demand	Actual Demand
1	78	71
2	75	80
3	83	101
4	84	84
5	88	60
6	85	73

Problem: 7

Over the past year Meredith and Smunt Manufacturing had annual sales of 10,000 portable water pumps. The average quarterly sales for the past 5 years have averaged: spring 4,000, summer 3,000, fall 2,000, and winter 1,000. Compute the quarterly index.

Problem: 8

Using the data in Problem 7, Meredith and Smunt Manufacturing expects sales of pumps to grow by 10% next year. Compute next year's sales and the sales for each quarter.

Solutions**Problem 1:**

Week	Auto Sales	Three-Week Moving Average
1	8	
2	10	
3	9	

4	11	$(8 + 9 + 10) / 3 = 9$
5	10	$(10 + 9 + 11) / 3 = 10$
6	13	$(9 + 11 + 10) / 3 = 10$
7	-	$(11 + 10 + 13) / 3 = 11 \frac{1}{3}$

Problem 2:

Week	Auto Sales	Three-Week Moving Average
1	8	
2	10	
3	9	
4	11	$[(3*9) + (2*10) + (1*8)] / 6 = 9 \frac{1}{6}$
5	10	$[(3*11) + (2*9) + (1*10)] / 6 = 10 \frac{1}{6}$
6	13	$[(3*10) + (2*11) + (1*9)] / 6 = 10 \frac{1}{6}$
7	-	$[(3*13) + (2*10) + (1*11)] / 6 = 11 \frac{2}{3}$

Problem 3:

Problem 4:

Month	Actual Battery Sales	Rounded Forecast with a =0.8	Absolute Deviation with a =0.8	Rounded Forecast with a =0.5	Absolute Deviation with a =0.5
January	20	22	2	22	2

February	21	20	1	21	0
March	15	21	6	21	6
April	14	16	2	18	4
May	13	14	1	16	3
June	16	13	3	14.5	1.5
			$\sum = 15$		$\sum = 16.5$
$MAD = \frac{\sum \text{deviations}}{n}$		2.5		2.75	

Based on this analysis, a smoothing constant of $a = 0.8$ is preferred to that of $a = 0.5$ because it has a smaller MAD.

Problem 5:

Year	Time Period (X)	Sales (Units) (Y)	X^2	XY
2006	1	100	1	100
2007	2	110	4	220
2008	3	122	9	366
2009	4	130	16	520
2010	5	139	25	695
2011	6	152	36	912
2012	7	164	49	1148

Therefore, the least squares trend equation is:

To project demand in 2013, we denote the year 2013 as and:

Sales in

Problem 6:

Year	Forecast Demand	Actual Demand	Error	RSFE
1	78	71	-7	-7
2	75	80	5	-2
3	83	101	18	16
4	84	84	0	16
5	88	60	-28	-12
6	85	73	-12	-24

Year	Forecast Demand	Actual Demand	Forecast Error	Cumulative Error	MAD	Tracking Signal
1	78	71	7	7	7.0	-1.0
2	75	80	5	12	6.0	-0.3
3	83	101	18	30	10.0	+1.6
4	84	84	0	30	7.5	+2.1
5	88	60	28	58	11.6	-1.0
6	85	73	12	70	11.7	-2.1

Problem 7:

Sales of 10,000 units annually divided equally over the 4 seasons is and the seasonal index for each quarter is: spring summer fall winter

Problem 8:

Next years sales should be 11,000 pumps Sales for each quarter should be 1/4 of the annual sales the quarterly index.

Exercise 1. DRINKING WATER MONITORING AND FORECASTING USING R**Aim:**

To perform analysis on drinking water dataset and use R to do time series forecasting on the data by analyzing, monitoring and plotting the obtained forecast

Problem Statement:

Getting enough water every day is important for one's health. Drinking water can prevent dehydration, a condition that can cause unclear thinking, result in mood change, cause your body to overheat, and lead to constipation and kidney stones. It is critical to examine the amount of water consumed on a regular basis in order to determine how much water has been consumed and to enhance water consumption if it is too low or vice versa.

Dataset:

The dataset <https://raw.githubusercontent.com/jbrownlee/Datasets/master/yearly-water-usage.csv> which consists of annual water consumption in Baltimore from 1885 to 1963 (unit used is liters per capita per day), is used to analyze and monitor drinking water. Time series forecasting using SMA, Holt-Winter filtering, MannKendall and data visualization are performed using the same.

Procedure:

- Install necessary libraries like Kendall, wql, etc.
- Import the dataset downloaded from <https://raw.githubusercontent.com/jbrownlee/Datasets/master/yearly-water-usage.csv>

Plot data as time series

Plot logarithmic time series

Plot SMA(Simple Moving Average) and view the time series output

Use Holt – Winters filtering and view the time series output

Forecast based on Holt – Winters

Calculate Mann-Kendall test of trend on time series and visualize the output

Perform decomposition of Additive time series

Plot decomposition of Additive time series

Convert time series to dataframe using ts2df.

CODE:

```
#R version 4.1.2 (2021-11-01)
```

```
#RStudio version 1.2.1335
```

```
#Program Execution
```

#1. Importing dataset and plotting values as a timeseries

```
df1 <- read.csv("C:\\\\Users\\\\Lenovo\\\\waterdata.csv")  
time_series <- ts(df1$Water,frequency=1, start=c(1885))  
time_series  
plot.ts(time_series)
```

#2 Plotting Logarithmic timeseries

```
log_series <- log(time_series)  
log_series  
plot.ts(log_series)
```

#3 Simple Moving Average(SMA)

```
library("TTR")  
SMA_series <- SMA(time_series,n=3)  
plot.ts(SMA_series)
```

#4 Holt-Winters filtering

```
time_series_forecasts <- HoltWinters(time_series, beta=FALSE, gamma=FALSE)  
time_series_forecasts  
time_series_forecasts$fitted  
plot(time_series_forecasts)
```

#5 Forecasting

```
time_series_forecasts$SSE  
HoltWinters(time_series, beta=FALSE, gamma=FALSE, l.start=23.56)
```

#6 MannKendall

```
library(Kendall)
```

```
MannKendall(time_series)
```

```
plot(time_series)
```

```
lines(lowess(time(time_series),time_series), col='blue')
```

#7 Decomposition of Additive timeseries

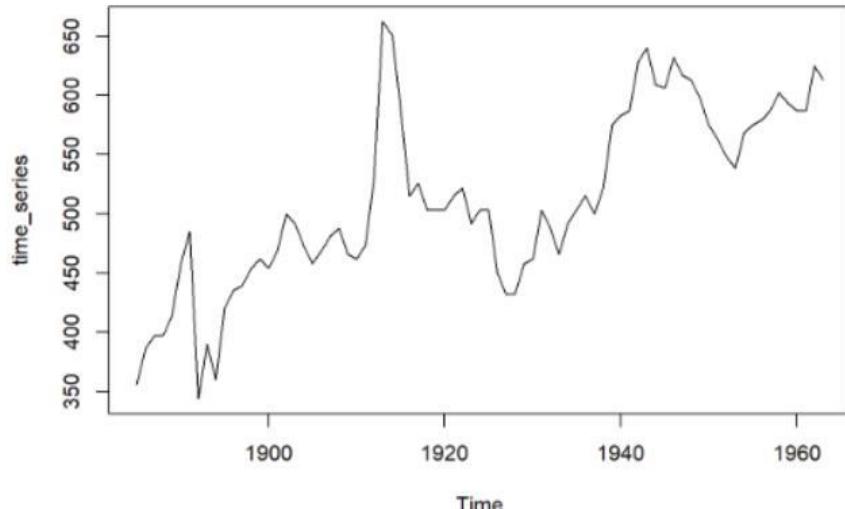
```
time_series <- ts(df1$Water,frequency=12, start=c(1885)) time_series_components <- decompose(time_series) time_series_components$seasonal
```

```
plot(time_series_components)
```

```
#1. Importing dataset and plotting values as a timeseries  
df1 <- read.csv("C:\\Users\\Lenovo\\waterdata.csv")  
time_series <- ts(df1$Water,frequency=1, start=c(1885))  
time_series
```

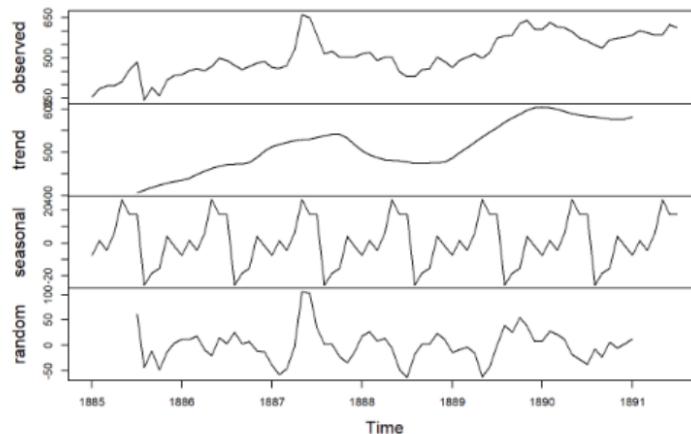
```
## Time Series:  
## Start = 1885  
## End = 1963  
## Frequency = 1  
## [1] 356 386 397 397 413 458 485 344 390 360 420 435 439 454 462 454 469 500 492  
## [20] 473 458 469 481 488 466 462 473 530 662 651 587 515 526 503 503 503 515 522  
## [39] 492 503 503 450 432 432 458 462 503 488 466 492 503 515 500 522 575 583 587  
## [58] 628 640 609 606 632 617 613 598 575 564 549 538 568 575 579 587 602 594 587  
## [77] 587 625 613
```

```
plot.ts(time_series)
```



OUTPUT:

Decomposition of additive time series



```
#8 ts2df  
library(wql)
```

```
## Warning: package 'wql' was built under R version 4.1.3
```

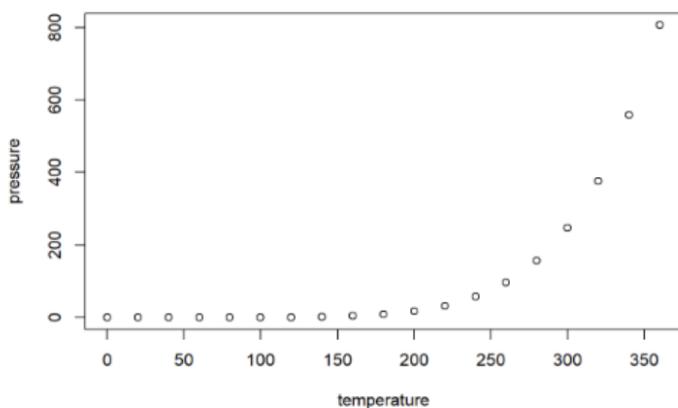
```
ts2df(time_series, monl = 1, addYr = FALSE, omit = FALSE)
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1885 356 386 397 397 413 458 485 344 390 360 420 435
## 1886 439 454 462 454 469 500 492 473 458 469 481 488
## 1887 466 462 473 530 662 651 587 515 526 503 503 503
## 1888 515 522 492 503 503 450 432 432 458 462 503 488
## 1889 466 492 503 515 500 522 575 583 587 628 640 609
## 1890 606 632 617 613 598 575 564 549 538 568 575 579
## 1891 587 602 594 587 587 625 613 NA NA NA NA NA NA
```

...

Including Plots

You can also embed plots, for example:

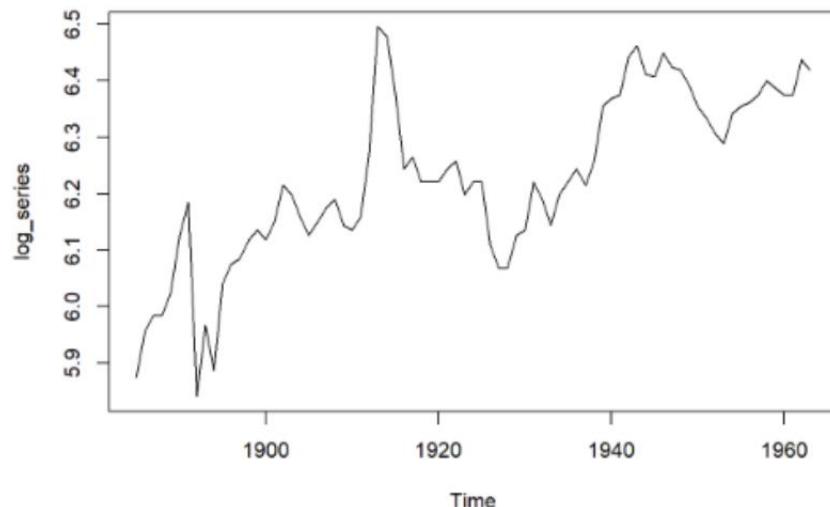


Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
#2 Plotting Logarithmic timeseries  
log_series <- log(time_series)  
log_series
```

```
## Time Series:  
## Start = 1885  
## End = 1963  
## Frequency = 1  
## [1] 5.874931 5.955837 5.983936 5.983936 6.023448 6.126869 6.184149 5.840642  
## [9] 5.966147 5.886104 6.040255 6.075346 6.084499 6.118097 6.135565 6.118097  
## [17] 6.150603 6.214608 6.198479 6.159095 6.126869 6.150603 6.175867 6.190315  
## [25] 6.144186 6.135565 6.159095 6.272877 6.495266 6.478510 6.375025 6.244167  
## [33] 6.265301 6.220590 6.220590 6.220590 6.244167 6.257668 6.198479 6.220590  
## [41] 6.220590 6.189248 6.068426 6.068426 6.126869 6.135565 6.220590 6.190315  
## [49] 6.144186 6.198479 6.220590 6.244167 6.214608 6.257668 6.354370 6.368187  
## [57] 6.375025 6.442540 6.461468 6.411818 6.406880 6.448889 6.424869 6.418365  
## [65] 6.393591 6.354370 6.335054 6.308098 6.287859 6.342121 6.354370 6.361302  
## [73] 6.375025 6.400257 6.386879 6.375025 6.375025 6.437752 6.418365
```

```
plot.ts(log_series)
```



```
#3 Simple Moving Average(SMA)  
library("TTR")  
SMA_series <- SMA(time_series,n=3)  
plot.ts(SMA_series)
```



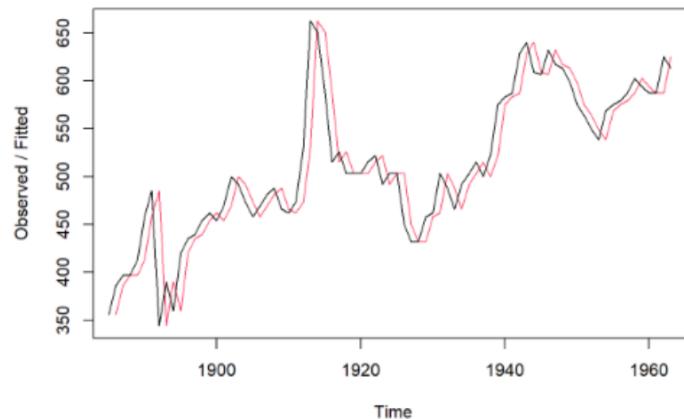
```
#4 Holt-Winters filtering
time_series_forecasts <- HoltWinters(time_series, beta=FALSE, gamma=FALSE)
time_series_forecasts
```

```
## Holt-Winters exponential smoothing without trend and without seasonal component.
##
## Call:
## HoltWinters(x = time_series, beta = FALSE, gamma = FALSE)
##
## Smoothing parameters:
## alpha: 0.9999294
## beta : FALSE
## gamma: FALSE
##
## Coefficients:
##      [,1]
## a 613.0008
```

```
time_series_forecasts$fitted
```

```
## Time Series:
## Start = 1886
## End = 1963
## Frequency = 1
##      xhat    level
## 1886 356.0000 356.0000
## 1887 385.9979 385.9979
## 1888 396.9992 396.9992
## 1889 397.0000 397.0000
## 1890 412.9989 412.9989
## 1891 457.9968 457.9968
## 1892 484.9981 484.9981
## 1893 344.0099 344.0099
## 1894 389.9968 389.9968
## 1895 360.0021 360.0021
## 1896 419.9958 419.9958
## 1897 434.9989 434.9989
## 1898 438.9997 438.9997
## 1899 453.9989 453.9989
## 1900 461.9994 461.9994
## 1901 454.0006 454.0006
## 1902 468.9989 468.9989
## 1903 499.9978 499.9978
```

Holt-Winters filtering



```
#5 Forecasting  
time_series_forecasts$SSE
```

```
## [1] 84473.46
```

```
HoltWinters(time_series, beta=FALSE, gamma=FALSE, l.start=23.56)
```

```
## Holt-Winters exponential smoothing without trend and without seasonal component.  
##  
## Call:  
## HoltWinters(x = time_series, beta = FALSE, gamma = FALSE, l.start = 23.56)  
##  
## Smoothing parameters:  
## alpha: 0.999926  
## beta : FALSE  
## gamma: FALSE  
##  
## Coefficients:  
## [1]  
## a 613.0009
```

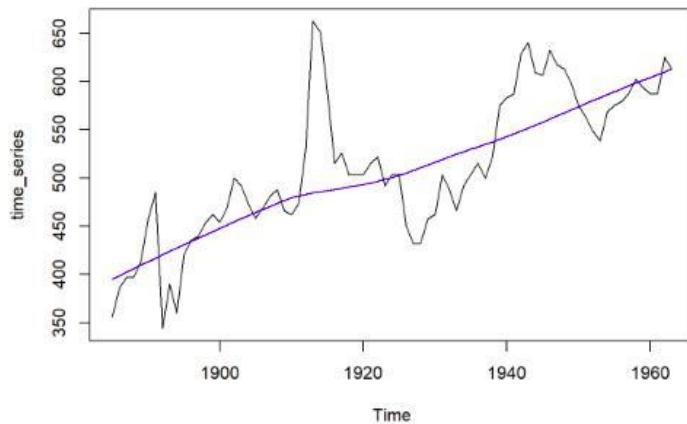
```
#6 MannKendall  
library(Kendall)
```

```
## Warning: package 'Kendall' was built under R version 4.1.3
```

```
MannKendall(time_series)
```

```
## tau = 0.599, 2-sided pvalue <= 2.22e-16
```

```
plot(time_series)  
lines(lowess(time_series), time_series), col='blue')
```



```
#7 Decomposition of Additive timeseries
time_series <- ts(dfl$Water,frequency=12, start=c(1885))
time_series_components <- decompose(time_series)
time_series_components$seasonal
```

	Jan	Feb	Mar	Apr	May	Jun
## 1885	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1886	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1887	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1888	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1889	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1890	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
## 1891	-7.551042	1.490625	-4.451042	6.182292	26.557292	17.265625
##	Jul	Aug	Sep	Oct	Nov	Dec
## 1885	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1886	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1887	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1888	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1889	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1890	17.386458	-25.551042	-18.252431	-15.439931	4.032292	-1.669097
## 1891	17.386458					

```
plot(time_series_components)
```

Exercise 2: IoT BASED HEALTH MONITORING USING R

Aim:

To formulate an IOT based healthcare application – prediction of possibility of heart attack using Generalized Linear model, Random forest and Decision trees in R

Problem Statement:

Heart disease has received a lot of attention in medical research as one of the many life-threatening diseases. The diagnosis of heart disease is a difficult task which when automated can offer better predictions about the patient's heart condition so that further treatment can be made effective. The signs, symptoms, and physical examination of the patient are usually used to make a diagnosis of heart disease. Resting blood pressure, cholesterol, age, sex, type of chest pain, fasting blood sugar, ST depression, and exercise-induced angina can all help to predict the likelihood of having a heart attack. Using models like Decision trees, Random forest and GLM to train on the given dataset and view the predicted class – 0 = less chance of heart attack, 1 = more chance of heart attack.

Procedure:

Import the packages Rplot, RColorBrewer, Rattle and randomForest

Download and read the dataset from Kaggle : <https://www.kaggle.com/datasets/nareshbhat/health-care-data-set-on-heart-attack-possibility>

View the statistics of the variables in the dataset using function “summary”.

Analyse the data, specific to resting blood pressure

Using the “cor” function, find the correlation between resting blood pressure and age

Construct a Logistic regression model using GLM and view the output plots

Encode the target values into categorical values

Split the dataset into training and testing data in the ratio 70 : 30

Construct a decision tree model 10.Target variable is categorised based on resting blood pressure, serum cholesterol and maximum heart rate achieved

Plot the decision tree and view the output

Devise a Random forest model based on the relationship between resting blood pressure, old peak and chest pain type

View the confusion matrix and importance of each predictor

CODE:

```
#Ex1- IOT based healthcare application using Generalized Linear model, Random forest and Decision trees in R
```

```
#R version 3.3.2 (2016-10-31)
```

```
#RStudio version 1.2.1335
```

Loading all the necessary Libraries

```
library(rpart) #used for building classification and regression trees.
```

```
library(rpart.plot)
```

Installing the necessary packages

```
In [1]: install.packages("rpart.plot")
install.packages("rattle")
install.packages("randomForest")

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'bitops', 'XML'

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
```

Loading all the necessary Libraries

```
In [2]: library(rpart) #used for building classification and regression trees.
library(rpart.plot)
library(RColorBrewer) # help you choose sensible colour schemes for figures in R.
library(rattle) # provides a collection of utilities functions for a data scientist.
library(randomForest) #has the function randomForest() which is used to create and analyse random forests.

Loading required package: tibble
Loading required package: bitops

Rattle: A free graphical interface for data science with R.
Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.

randomForest 4.7-1

Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'

The following object is masked from 'package:rattle':
  importance
library(RColorBrewer) #
```

help you choose sensible colour schemes for figures
library(rattle) # provides a collection of utilities functions for a data scientist.
library(randomForest) #Used to create and analyse random forests.

Loading the dataset

```
data = read.csv("heart_health.csv")
```

Loading and Understanding the dataset

```
In [4]: data = read.csv("/content/heart_health.csv")
```

```
In [5]: print("Minimum resting blood pressure")
min(data$trestbps)
print("Maximum resting blood pressure")
max(data$trestbps)
print("Summary of Dataset")
summary(data)

[1] "Minimum resting blood pressure"
94
[1] "Maximum resting blood pressure"
200
[1] "Summary of Dataset"
  age       sex      cp      trestbps 
Min. :29.00  Min. :0.0000  Min. :0.0000  Min. : 94.0 
1st Qu.:47.50  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:120.0 
Median :55.00  Median :1.0000  Median :1.0000  Median :130.0 
Mean   :54.37  Mean   :0.6832  Mean   :0.967  Mean   :131.6 
3rd Qu.:61.00  3rd Qu.:1.0000  3rd Qu.:2.000  3rd Qu.:140.0 
Max.  :77.00  Max.  :1.0000  Max.  :3.000  Max.  :200.0 
  chol      fbs      restecg     thalach 
Min. :126.0  Min. :0.0000  Min. :0.0000  Min. : 71.0 
1st Qu.:121.0 1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:133.5 
Median :124.0  Median :0.0000  Median :1.0000  Median :153.0 
Mean   :126.3  Mean   :0.1485  Mean   :0.5281  Mean   :149.6 
3rd Qu.:127.5 3rd Qu.:1.0000  3rd Qu.:1.0000  3rd Qu.:166.0 
Max.  :156.0  Max.  :1.0000  Max.  :2.0000  Max.  :202.0 
  exang     oldpeak    slope      ca 
Min. :0.0000  Min. :0.00  Min. :0.0000  Min. :0.0000 
1st Qu.:0.0000  1st Qu.:0.00  1st Qu.:0.0000  1st Qu.:0.0000 
Median :0.0000  Median :0.80  Median :1.0000  Median :0.0000 
Mean   :0.3267  Mean   :1.04  Mean   :1.399  Mean   :0.7294 
3rd Qu.:1.0000  3rd Qu.:1.60  3rd Qu.:2.0000  3rd Qu.:1.0000 
Max.  :3.0000  Max.  :4.20  Max.  :2.0000  Max.  :4.0000 
  thal      target    
Min. :0.0000  Min. :0.0000 
1st Qu.:2.0000  1st Qu.:0.0000 
Median :2.0000  Median :1.0000 
Mean   :2.314  Mean   :0.5446 
3rd Qu.:3.0000  3rd Qu.:1.0000 
Max.  :3.0000  Max.  :1.0000
```

```
In [6]: print("Range of resting blood pressure")
max(data$trestbps) - min(data$trestbps)

quantile(data$trestbps, c(0.25, 0.5, 0.75))
print("Column names of the Data")
names(data)
print("Attributes of the Data")
str(data)
print("Number of Rows and Columns:")
dim(data)
```

```
[1] "Range of resting blood pressure"
106
25%:120 50%:130 75%:140

[1] "Column names of the Data"
'age' 'sex' 'cp' 'trestbps' 'chol' 'fbs' 'restecg' 'thalach' 'exang' 'oldpeak' 'slope' 'ca' 'thal' 'target'

[1] "Attributes of the Data"
'data.frame': 303 obs. of 14 variables:
 $ age   : int 63 37 41 56 57 56 44 52 57 ...
 $ sex   : int 1 1 0 1 0 1 0 1 1 1 ...
 $ cp    : int 0 1 0 1 0 0 0 0 0 2 ...
 $ trestbps: int 145 138 152 120 128 140 140 120 172 150 ...
 $ chol   : int 233 250 204 236 354 192 294 263 199 168 ...
 $ fbs   : int 0 0 0 0 0 0 0 1 0 0 ...
 $ restecg: int 0 1 0 1 1 0 1 1 1 ...
 $ thalach: int 150 187 172 178 163 148 153 173 162 174 ...
 $ exang  : int 0 0 0 0 1 0 0 0 0 0 ...
 $ oldpeak: num 2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
 $ slope  : num 0 1.4 2.3 3 2 2 2 2 ...
 $ ca    : int 0 0 0 0 0 0 0 0 0 0 ...
 $ thal   : int 1 2 2 2 2 1 2 3 3 2 ...
 $ target : int 1 1 1 1 1 1 1 1 1 1 ...

[1] "Number of Rows and Columns:"
303 14
```

Analyze the Correlation between resting BP and age

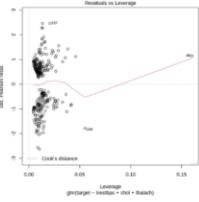
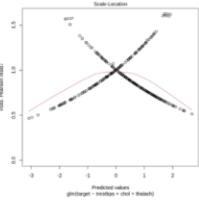
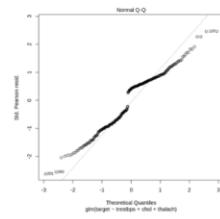
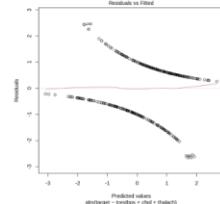
```
In [7]: print("Correlation between the resting blood pressure and the age")
cor(data$trestbps, data$age, method = "pearson")
cor.test(data$trestbps, data$age, method = "pearson")

[1] "Correlation between the resting blood pressure and the age"
0.279350906561288
Pearson's product-moment correlation

data: data$trestbps and data$age
t = 5.0475, df = 301, p-value = 7.762e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.1720897 0.3800657
sample estimates:
cor
0.2793509
```

Constructing the Logistic regression Model

```
In [1]:  
print("Constructing the Logistic regression Model")  
glm(target~trestbps+restecg + fbs, data = data, family=binomial())  
model <- glm(target~trestbps+ chol + thalach, data = data, family=binomial())  
plot(model)  
  
[1] "Constructing the Logistic regression Model"  
Call: glm(formula = target ~ trestbps + restecg + fbs, family = binomial(),  
        data = data)  
Coefficients:  
(Intercept) trestbps restecg fbs  
1.98546 -0.01566 0.48160 0.03337  
Degrees of Freedom: 302 Total (i.e. Null); 299 Residual  
Null Deviance: 417.6  
Residual Deviance: 406.6 AIC: 414.6
```



```
In [1]: # Make dependent variable as a factor (categorical)  
data$target = as.factor(data$target)
```

```
In [1]: # Splitting the dataset into test and train  
print("Train Test Split") # 70/30 Split  
dt = sort(sample(nrow(data), nrow(data)*.7))  
train<-data[dt,]  
val<-data[-dt,]
```

```
[1] "Train Test Split"
```

```
In [1]: # No. of rows in Train and Val Dataset  
nrow(train)  
nrow(val)
```

```
212  
91
```

Analyzing the data in the dataset

```
print("Minimum resting blood pressure") min(data$trestbps)  
print("Maximum resting blood pressure") max(data$trestbps)  
print("Summary of Dataset") summary(data)
```

```
print("Range of resting blood pressure") max(data$trestbps) – min(data$trestbps) quantile(data$trestbps, c(0.25, 0.5, 0.75)) print("Column name of the Data")
names(data)

print("Attributes of the Data") str(data)

print("Number of Rows and Columns:") dim(data)

# Analyze the Correlation between resting BP and age

print("Correlation between the resting blood pressure and the age")
cor(data$trestbps, data$age, method = "pearson")
cor.test(data$trestbps, data$age, method = "pearson")

# Constructing the GLM

print("Constructing the Logistic regression Model")
glm(target~ trestbps+ restecg + fbs, data = data, family=binomial())
model <- glm(target~trestbps+ chol + thalach, data = data, family=binomial())
plot(model)

Make dependent variable as a factor (categorical) data$target = as.factor(data$target)

Splitting the dataset into test and train print("Train Test Split") # 70/30 Split
dt = sort(sample(nrow(data), nrow(data)*.7)) train<-data[dt,]
val<-data[-dt,] nrow(train) nrow(val)

Constructing the Decision Tree Model print("Construction of the Decision Tree Model") mtree <- rpart(target ~ trestbps + chol + thalach, data = train,
method="class",
control = rpart.control(minsplit = 20, minbucket = 7, maxdepth = 10, usesurrogate = 2,
xval =10))
mtree

Plotting the Decision Tree for the dataset print("Plotting the Decision Tree") plot(mtree)
text(mtree)
par(xpd = NA, mar = rep(0.7, 4)) plot(mtree, compress = TRUE)
text(mtree, cex = 0.7, use.n = TRUE, fancy = FALSE, all = TRUE) prp(mtree, faclen = 0,box.palette = "Reds", cex = 0.8, extra = 1)
```

Constructing the Random Forest model

```
rf <- randomForest(target ~ trestbps + oldpeak + cp, data = data)
```

View the forest results print("Random Forest Results:")

```
print(rf)
```

Importance of each predictor print("Importance of each predictor:") print(importance(rf,type = 2))

Plot the Random Forest

```
plot(rf)
```

#Conclusion

#Models like Decision trees, Random forest and GLM were trained on the given dataset and the predictions were visualised successfully

Constructing the Decision Tree Model

```
In [--]
print("Construction of the Decision Tree Model")
mtree <- rpart(
  target ~ trestbps + chol + thalach,
  data = train,
  method="class",
  control = rpart.control(
    minsplit = 20,
    minbucket = 7,
    maxdepth = 10,
    usesurrogate = 2,
    xval =10
  )
)
mtree

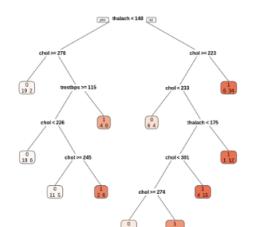
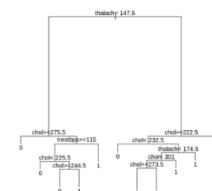
[1] "Construction of the Decision Tree Model"
n= 212

node), split, n, loss, yval, (yprob)
 * denotes terminal node

1) root 212 91 1 (0.42924528 0.57075472)
  2) thalach< 171.5 81 21 (0.66666667 0.33333333)
    4) chol< 275.5 60 25 0 (0.58233233 0.41666667)
      9) chol< 275.5 60 25 0 (0.58233233 0.41666667)
      10) trestbps=>115 48 17 0 (0.64583333 0.35416667)
        20) chol< 225.5 28 6 0 (0.75000000 0.25000000) *
        21) chol=>225.5 24 11 0 (0.54166667 0.45833333)
        22) chol=>244.5 18 5 0 (0.68750000 0.31250000) *
        43) chol< 244.5 8 2 0 (0.25000000 0.75000000) *
    11) thalach>=171.5 15 4 0 (0.58233233 0.41666667) *
  3) thalach>=147.5 91 31 1 (0.34065934 0.65934066)
  12) chol< 232.5 10 4 0 (0.60000000 0.40000000) *
  13) chol=>232.5 80 25 1 (0.30864190 0.69135802)
  26) thalach< 174.5 68 24 1 (0.35294118 0.64705882)
    52) chol< 301 49 20 1 (0.40816321 0.59183673)
      104) chol=>273.5 20 10 0 (0.59411765 0.40588235) *
      105) chol=>301 19 4 1 (0.21052632 0.78947368) *
  53) thalach>=174.5 13 1 1 (0.07692308 0.92307692) *
  27) chol< 222.5 40 6 1 (0.15000000 0.85000000) *
```

```
In [--]
# Plotting the Decision Tree for the dataset
print("Plotting the Decision Tree")
plot(mtree)
text(mtree)
par(xpd = NA, mar = rep(0.7, 4))
plot(mtree, compress = TRUE)
text(mtree, cex = 0.7, use.n = TRUE, fancy = FALSE, all = TRUE)
prp(mtree, faclen = 0, box.palette = "Reds", cex = 0.8, extra = 1)

[1] "Plotting the Decision Tree"
```



OUTPUT:

Exercise 3: TRAFFIC PATTERN RECOGNITION USING R

Aim:

To formulate an IOT based Traffic pattern recognition using Decision tree, correlation study, Naïve Bayes classification and Time series forecasting in R

Problem Statement:

The term “traffic patterns recognition” refers to the process of recognising a user’s current traffic pattern, which can be applicable to transportation planning, location-based services, social networks, and a range of other applications.

Dataset:

Using the dataset from Kaggle <https://www.kaggle.com/datasets/utathya/smart-city-traffic-patterns> called “Smart City traffic patterns”, perform pattern recognition and prediction using Decision Tree classifiers and Naïve Bayes classification. Moreover, time series analysis and simple moving average, Arima and exponential smoothing are performed.

Procedure:

Import required packages after installing

Load and read the data set

Pre-process the data appropriately

Use summary method to see the characteristics of the data set

Use the Simple Moving Average forecasting model and visualize the output

Use the Exponential smoothing forecasting model and see the output

Use the Arima forecasting model and view the output

Get the correlation between the columns

Split the data set into training and testing in the ratio of 70:30

Perform Decision tree classification and view the results in tree format

Perform Naïve Bayes and view the results in confusion matrix

CODE:

```
#R version 3.6.1
```

```
#RStudio version 1.2.1335
```

```
#Import required packages after installing
```

```
library("e1071")
```

```
library("caTools")
library("caret")
library("party")
library("dplyr")
library("magrittr")
library("TTR")
library("data.table")

#Load the data set
data <- read.csv("traffic.csv")
data

#Pre-process the data appropriately
data$DateTime = strtrim(data$DateTime,15)
data

#Use summary method to see the characteristics of the data set
print("Summary of Dataset")
summary(data)

# correlation study
print("Correlation between Traffic and Junction")
cor(data$Vehicles,data$Junction,method = "pearson")
cor.test(data$Vehicles,data$Junction,method = "pearson")

#Use the Simple Moving Average forecasting model and visualize the output
t_col1 <- fread("traffic.csv",select = c("Vehicles"))

t_col1series <- ts(t_col1,frequency=12, start=c(2015,1))

t_col1series[is.na(t_col1series)]<-mean(t_col1series,na.rm=TRUE) #Replace NA with mean

t_col1seriesSMA3 <- SMA(t_col1series,n=12)

plot.ts(t_col1seriesSMA3)
```

```

#Use the Exponential smoothing forecasting model and visualize the output t_col1 <- fread("traffic.csv",select = c("Junction"))

t_col1series <- ts(t_col1,frequency=12, start=c(2015,1))

t_col1series[is.na(t_col1series)]<-mean(t_col1series,na.rm=TRUE) #Replace NA with mean

t_col1seriesforecasts <- HoltWinters(t_col1series, beta=FALSE, gamma=FALSE)

t_col1seriesforecasts

t_col1seriesforecasts$SSE

HoltWinters(t_col1series, beta=FALSE, gamma=FALSE, l.start=23.56)

#Use the Arima forecasting model and view the output library("TTR")

v1 <- data[[4]]

datats <- ts(v1)

partition into train and test train_series=datats[1:40] test_series=datats[41:50]

make arima models

arimaModel_1=arima(train_series, order=c(0,1,2))

arimaModel_2=arima(train_series, order=c(1,1,0))

arimaModel_3=arima(train_series, order=c(1,1,2))

## look at the parameters

print(arimaModel_1);print(arimaModel_2);print(arimaModel_3)

#Split the data set into training and testing in the ratio of 70:30

split <- sample.split(data, SplitRatio = 0.7)

train_cl <- subset(data, split == "TRUE")

test_cl <- subset(data, split == "FALSE")

#Perform Decision tree classification and view the results in tree format model<- ctree(Vehicles ~ Junction, train_cl) plot(model)

#Perform Naïve Bayes and view the results in confusion matrix set.seed(120) # Setting Seed

classifier_cl <- naiveBayes(Junction ~ ., data = train_cl)

# Predicting on test data

```

```
y_pred <- predict(classifier_cl, newdata = test_cl)

# Confusion Matrix

cm <- table(test_cl$Junction, y_pred)

cm

confusionMatrix(cm)

plot(cm)
```

2. Load the data set

```
data <- read.csv("traffic.csv")
data
```

DateTime	Junction	Vehicles	ID
2015-11-01 00:00:00	1	15	20151101001
2015-11-01 01:00:00	1	13	20151101011
2015-11-01 02:00:00	1	10	20151101021
2015-11-01 03:00:00	1	7	20151101031
2015-11-01 04:00:00	1	9	20151101041
2015-11-01 05:00:00	1	6	20151101051
2015-11-01 06:00:00	1	9	20151101061
2015-11-01 07:00:00	1	8	20151101071
2015-11-01 08:00:00	1	11	20151101081
2015-11-01 09:00:00	1	12	20151101091

1-10 of 10,000 rows Previous [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) ... [1000](#) Next

3. Pre-process the data appropriately

```
data$DateTime = trimws(data$DateTime, 15)
data
```

DateTime	Junction	Vehicles	ID
2015-11-01 00:0	1	15	20151101001
2015-11-01 01:0	1	13	20151101011
2015-11-01 02:0	1	10	20151101021
2015-11-01 03:0	1	7	20151101031
2015-11-01 04:0	1	9	20151101041
2015-11-01 05:0	1	6	20151101051
2015-11-01 06:0	1	9	20151101061
2015-11-01 07:0	1	8	20151101071
2015-11-01 08:0	1	11	20151101081
2015-11-01 09:0	1	12	20151101091

1-10 of 10,000 rows Previous [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) ... [1000](#) Next

4. Use summary method to see the characteristics of the data set

```
print("Summary of Dataset")
```

```
## [1] "Summary of Dataset"
```

```
summary(data)
```

```
##   DateTime       Junction      Vehicles        ID
##   Length:48120   Min.   :1.000   Min.   : 1.00   Min.   :2.015e+10
##   Class :character 1st Qu.:1.000   1st Qu.: 9.00   1st Qu.:2.016e+10
##   Mode  :character Median :2.000   Median :15.00   Median :2.016e+10
##               Mean  :2.181   Mean  : 22.79   Mean  :2.016e+10
##               3rd Qu.:3.000   3rd Qu.: 29.00   3rd Qu.:2.017e+10
##               Max.  :4.000   Max.  :180.00   Max.  :2.017e+10
```

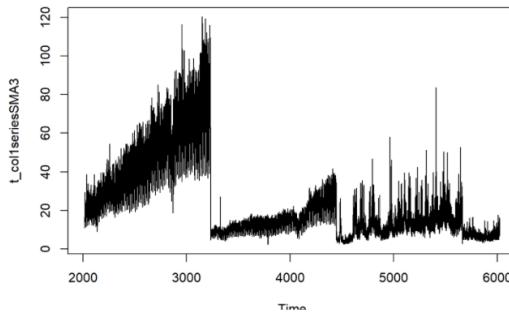
OUTPUT:

5. Use the Simple Moving Average forecasting model and visualize the output

```
t_coll <- fread("traffic.csv",select = c("Vehicles"))
t_collseries <- ts(t_coll,frequency=12, start=c(2015,1))

t_collseries[is.na(t_collseries)]<-mean(t_collseries,na.rm=TRUE) #Replace NA with
mean

t_collseriesSMA3 <- SMA(t_collseries,n=12)
plot.ts(t_collseriesSMA3)
```



6. Use the Exponential smoothing forecasting model and visualize the output

```
#Forecast using Exponential smoothing
t_coll <- fread("traffic.csv",select = c("Junction"))
t_collseries <- ts(t_coll,frequency=12, start=c(2015,1))
t_collseries[is.na(t_collseries)]<-mean(t_collseries,na.rm=TRUE) #Replace NA with
mean

t_collseriesforecasts <- HoltWinters(t_collseries, beta=FALSE, gamma=FALSE)
t_collseriesforecasts

## Holt-Winters exponential smoothing without trend and without seasonal component
.
##
## Call:
## HoltWinters(x = t_collseries, beta = FALSE, gamma = FALSE)
##
## Smoothing parameters:
##   alpha: 0.9999188
##   beta : FALSE
##   gamma: FALSE
##
## Coefficients:
##   [1]
## a    4

t_collseriesforecasts$SSE

## [1] 3

HoltWinters(t_collseries, beta=FALSE, gamma=FALSE, l.start=23.56)

## Holt-Winters exponential smoothing without trend and without seasonal component
.
##
## Call:
## HoltWinters(x = t_collseries, beta = FALSE, gamma = FALSE, l.start = 23.56)
##
## Smoothing parameters:
##   alpha: 0.9999188
##   beta : FALSE
##   gamma: FALSE
##
## Coefficients:
##   [1]
## a    4
```

7.Use the Arima forecasting model and view the output

```
library("TTR")
v1 <- data[[4]]
datats <- ts(v1)
## partition into train and test
train_series=datats[1:40]
test_series=datats[41:50]
## make arima models
arimaModel_1=arima(train_series, order=c(0,1,2))
arimaModel_2=arima(train_series, order=c(1,1,0))
arimaModel_3=arima(train_series, order=c(1,1,2))
## look at the parameters
print(arimaModel_1);print(arimaModel_2);print(arimaModel_3)
```

```
##
## Call:
## arima(x = train_series, order = c(0, 1, 2))
##
## Coefficients:
##          ma1      ma2
##     0.0005  0.0005
## s.e.  0.0061  0.0061
##
## sigma^2 estimated as 15294:  log likelihood = -243.23,  aic = 492.45
```

```
##
## Call:
## arima(x = train_series, order = c(1, 1, 0))
##
## Coefficients:
##          ar1
##     0.0005
## s.e.  0.0061
##
## sigma^2 estimated as 15297:  log likelihood = -243.23,  aic = 490.46
```

```
##
## Call:
## arima(x = train_series, order = c(1, 1, 2))
##
## Coefficients:
##          ar1      ma1      ma2
##     4e-04  2e-04  0.0005
## s.e.    NaN    NaN  0.0061
##
## sigma^2 estimated as 15294:  log likelihood = -243.23,  aic = 494.45
```

```
forecast1=predict(arimaModel_1, 10)
forecast2=predict(arimaModel_2, 10)
forecast3=predict(arimaModel_3, 10)
forecast1
```

```
## $pred
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 20151102151 20151102151 20151102151 20151102151 20151102151 20151102151
## [7] 20151102151 20151102151 20151102151 20151102151
##
## $se
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 123.6684 174.9410 214.3161 247.5044 276.7408 303.1707 327.4745 350.0951
## [9] 371.3403 391.4341
```

```
forecast2
```

```
## $pred
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 20151102151 20151102151 20151102151 20151102151 20151102151 20151102151
## [7] 20151102151 20151102151 20151102151 20151102151
##
## $se
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 123.6808 174.9585 214.2989 247.4622 276.6787 303.0918 327.3808 349.9882
## [9] 371.2213 391.3040
```

```
forecast3
```

```
## $pred
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 20151102151 20151102151 20151102151 20151102151 20151102151 20151102151
## [7] 20151102151 20151102151 20151102151 20151102151
##
## $se
## Time Series:
## Start = 41
## End = 50
## Frequency = 1
## [1] 123.6684 174.9409 214.3159 247.5042 276.7405 303.1705 327.4742 350.0948
## [9] 371.3399 391.4337
```

8. Correlation study

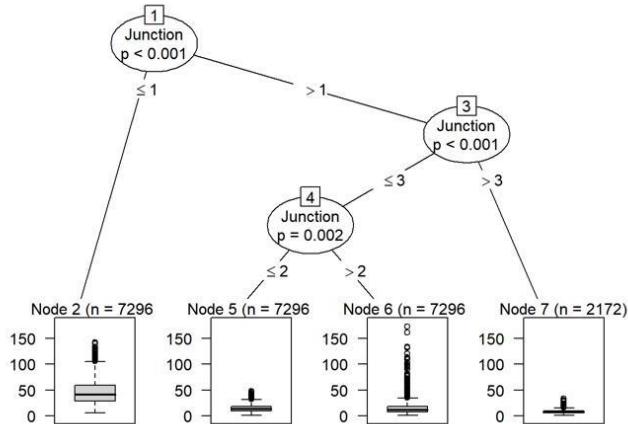
```
print("Correlation between Traffic and Junction")
## [1] "Correlation between Traffic and Junction"
cor(data$Vehicles,data$Junction,method = "pearson")
## [1] -0.6137872
cor.test(data$Vehicles,data$Junction,method = "pearson")
##
## Pearson's product-moment correlation
##
## data: data$Vehicles and data$Junction
## t = -170.54, df = 48118, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## -0.6193256 -0.6081877
## sample estimates:
## cor
## -0.6137872
```

9. Split the data set into training and testing in the ratio of 70:30

```
split <- sample.split(data, SplitRatio = 0.7)
train_cl <- subset(data, split == "TRUE")
test_cl <- subset(data, split == "FALSE")
```

10. Perform Decision tree classification and view the results in tree format

```
model<- ctree(Vehicles ~ Junction, train_cl)
plot(model)
```



11. Perform Naive Bayes and view the results in confusion matrix

```
set.seed(120) # Setting Seed
classifier_cl <- naiveBayes(Junction ~ ., data = train_cl)
# Predicting on test data
y_pred <- predict(classifier_cl, newdata = test_cl)
# Confusion Matrix
cm <- table(test_cl$Junction, y_pred)
cm
```

```
##      y_pred
##      1     2     3     4
## 1  5146    9   821 1320
## 2   408   686   361 5841
## 3   397   510   466 5923
## 4     1   36    2 2133
```

```
confusionMatrix(cm)
```

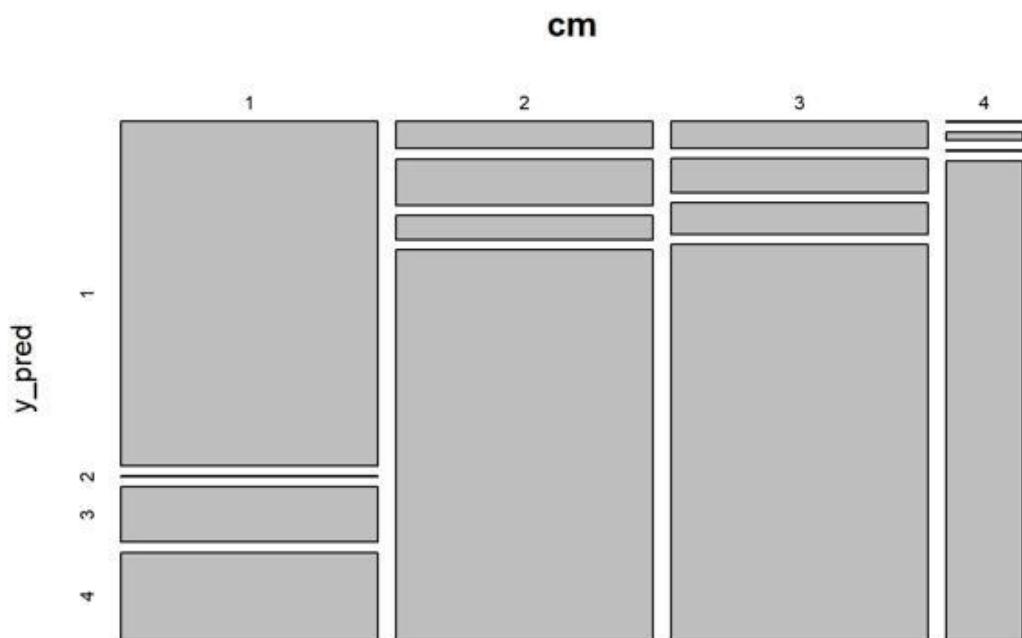
```
## Confusion Matrix and Statistics
##
##      y_pred
##      1     2     3     4
## 1  5146    9   821 1320
## 2   408   686   361 5841
## 3   397   510   466 5923
## 4     1   36    2 2133
##
## Overall Statistics
##
##              Accuracy : 0.3504
##              95% CI : (0.3444, 0.3565)
##  No Information Rate : 0.6325
##  P-Value [Acc > NIR] : 1
```

```

## 
##  Mcnemar's Test P-Value : <2e-16
## 
## Statistics by Class:
## 
##          Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity      0.8646  0.55278  0.28242  0.14017
## Specificity      0.8813  0.71033  0.69523  0.99559
## Pos Pred Value   0.7053  0.09402  0.06387  0.98204
## Neg Pred Value   0.9519  0.96689  0.92937  0.40223
## Prevalence        0.2474  0.05158  0.06858  0.63246
## Detection Rate   0.2139  0.02851  0.01937  0.08865
## Detection Prevalence 0.3032  0.30324  0.30324  0.09027
## Balanced Accuracy 0.8729  0.63155  0.48882  0.56788

```

plot(cm)



#Conclusion

Traffic pattern recognition with Decision trees, correlation study, Naïve Bayes classification and Time series forecasting was successfully implemented and visualised using R

Exercise 4: POWER DATAANALYSIS AND VISUALISATION FOR PROJECT IN HOME POWER IN RASPBERRY PI USING R

AIM:

To perform analysis on power data collected using Raspberry Pi and use R to do time series forecasting on the data by analyzing it and plotting the obtained forecast

PROBLEM STATEMENT:

The data on household power consumption can not only show the current state of household power consumption, but it can also bring awareness to the power sector, assisting in the understanding of power supply. With the proliferation of smart electricity metres and the widespread deployment of electricity producing technology such as solar panels, there is a lot of data about electricity usage available.

The Kaggle Dataset named “Household Electric Power Consumption” present in the link “<https://www.kaggle.com/datasets/uciml/electric-power-consumption-data-set>”, is a multivariate time series of power-related variables that can be used to model and even forecast future electricity consumption. Time series forecasting, exploratory data analytics and data visualization is performed using the same.

PROCEDURE:

Import the table, dplyr, lubridate, plotly and forecast packages

Import the data and print a short summary of it using the head(), glimpse() and summary() functions.

Check for the missing values in the dataset and remove if there are any.

Convert the date and time to a standard format.

Extract the Year, Week and Day from the dataset.

Visualize the granularity of the submetering using the plot() function

Filter the data for any particular year and visualize the submetering for that particular year.

Use the plotly package to plot the submetering across a day of usage.

Reduce the number of observations for that day and again plot the graph using plotly.

For the time series analysis extract the weekly time series data for all the submeters and plot them.

Fit the time series data into a time series linear regression model.

View the summary of the model.

Plot the forecast for all 3 submeters.

CODE:

```
#R version 3.6.1
#RStudio version 1.2.1335

Import Packages library(data.table)
library(dplyr)
library(lubridate)
library(plotly)
library(forecast)

# Import Data
data <- fread("household_power_consumption.txt")
head(data)
glimpse(data)
summary(data)

# Data Preprocessing
data <- data[complete.cases(data)]
sum(is.na(data))

data$datetime <- paste(data$Date,data$Time)

data$datetime <- as.POSIXct(data$datetime, format="%d/%m/%Y %H:%M:%S") attr(data$datetime, "tzone") <- "Europe/Paris" str(data)

data$year <- year(data$datetime)
data$week <- week(data$datetime)
data$day <- day(data$datetime)
data$month <- month(data$datetime)
data$minute <- minute(data$datetime)

Data Visualization plot(data$Sub_metering_1) ann <- filter(data, year == 2006) plot(ann$Sub_metering_1) plot(ann$Sub_metering_2)
plot(ann$Sub_metering_3)

houseDay <- filter(data, year == 2008 & day == 10 & month==1)
```

```

plot_ly(houseDay, x = ~houseDay$datetime, y = ~houseDay$Sub_metering_1, type = 'scatter', mode = 'lines')

dtDay <- filter(data, year == 2009 & day == 2 & month==2)

plot_ly(dtDay, x = ~dtDay$datetime, y = ~dtDay$Sub_metering_1, name = 'Kitchen', type = 'scatter', mode = 'lines') %>%
add_trace(y = ~dtDay$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
add_trace(y = ~dtDay$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%

layout(title = "Power Consumption Feb 2th, 2009",
xaxis = list(title = "Time"),
yaxis = list (title = "Power (watt-hours)"))

houseDay10 <- filter(data, year == 2009 & month == 2 & day == 2 & (minute == 0 | minute == 10 | minute == 20 | minute == 30 | minute == 40 | minute == 50))

plot_ly(houseDay10, x = ~houseDay10$datetime, y = ~houseDay10$Sub_metering_1, name = 'Kitchen', type = 'scatter', mode = 'lines') %>%
add_trace(y = ~houseDay10$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
add_trace(y = ~houseDay10$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%

layout(title = "Power Consumption Feb 2th, 2009", xaxis = list(title = "Time"),
yaxis = list (title = "Power (watt-hours)"))

data$minute <- minute(data$datetime)

houseDay10 <- filter(data, year == 2008 & month == 5 & day == 10 & (minute == 0 |
minute == 10 | minute == 20 | minute == 30 | minute == 40 | minute == 50))

plot_ly(houseDay10, x = ~houseDay10$datetime, y = ~houseDay10$Sub_metering_1,
name = 'Kitchen', type = 'scatter', mode = 'lines') %>%
add_trace(y = ~houseDay10$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
add_trace(y = ~houseDay10$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%

layout(title = "Power Consumption May 10th, 2008", xaxis = list(title = "Time"),
yaxis = list (title = "Power (watt-hours)"))

# Time Series Analysis

data$hour <- hour(data$datetime)

```

```

houseweekly <- filter(data, week == 2 & hour == 20 & minute == 1)

tsSM3_weekly <- ts(houseweekly$Sub_metering_3, frequency=52, start=c(2007,1))

plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 3")

tsSM3_weekly <- ts(houseweekly$Sub_metering_1, frequency=52, start=c(2007,1))

plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 1")

tsSM3_weekly <- ts(houseweekly$Sub_metering_2, frequency=52, start=c(2007,1))

plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 2")

house070809weekly <- filter(data, year==2008, hour == 20 & minute == 1)

tsSM3_070809weekly <- ts(house070809weekly$Sub_metering_3, frequency=52,
start=c(2008,3))

tsSM2_070809weekly <- ts(house070809weekly$Sub_metering_2, frequency=52, start=c(2008,3))

tsSM1_070809weekly <- ts(house070809weekly$Sub_metering_1, frequency=52, start=c(2008,3))

fit3 <- tslm(tsSM3_070809weekly ~ trend + season)

fit2 <- tslm(tsSM2_070809weekly ~ trend + season)

fit1 <- tslm(tsSM1_070809weekly ~ trend + season)

summary(fit3)

forecastfitSM3c <- forecast(fit3, h=20, level=c(80,90))

forecastfitSM2c <- forecast(fit2, h=20, level=c(80,90))

forecastfitSM1c <- forecast(fit1, h=20, level=c(80,90))

plot(forecastfitSM3c, ylim = c(0, 20), ylab= "Watt-Hours", xlab="Time")

plot(forecastfitSM2c, ylim = c(0, 20), ylab= "Watt-Hours", xlab="Time")

plot(forecastfitSM1c, ylim = c(0, 20), ylab= "Watt-Hours", xlab="Time")

```

#Conclusion

Time series forecasting, exploratory data analytics and data visualization using the Household Electric Power Consumption dataset was successfully implemented and visualised using R

OUTPUT:

1. Import all the above mentioned packages

```
In [..]
library(data.table)
library(dplyr)
library(lubridate)
library(plotly)
library(forecast)
```

2. Import the data and print a short summary of it using the head(), glimpse() and summary() functions

```
In [..]
data <- fread("household_power_consumption.txt")
```

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

A data.table: 6 x 9								
Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>
16/12/2006	17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17
16/12/2006	17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16
16/12/2006	17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17
16/12/2006	17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17
16/12/2006	17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17
16/12/2006	17:29:00	3.520	0.522	235.020	15.000	0.000	2.000	17

```
In [..]
glimpse(data)
```

```
Rows: 2,075,259
Columns: 9
$ Date      <chr> "16/12/2006", "16/12/2006", "16/12/2006", ...
$ Time      <chr> "17:24:00", "17:25:00", "17:26:00", "17:27:00", ...
$ Global_active_power <chr> "4.216", "5.360", "5.374", "5.388", "3.666", "3...
$ Global_reactive_power <chr> "0.418", "0.436", "0.498", "0.502", "0.528", "0...
$ Voltage    <chr> "234.840", "233.630", "233.290", "233.740", "235...
$ Global_intensity <chr> "18.400", "23.000", "23.000", "23.000", "15.800...
$ Sub_metering_1   <chr> "0.000", "0.000", "0.000", "0.000", "0.000", "0...
$ Sub_metering_2   <chr> "1.000", "1.000", "2.000", "1.000", "1.000", "2...
$ Sub_metering_3   <dbl> 17, 16, 17, 17, 17, 17, 16, 17, 17, 16, 17, 17, -
```

```
In [..]
summary(data)
```

```
Date        Time      Global_active_power
Length:2075259  Length:2075259  Length:2075259
Class :character Class :character Class :character
Mode  :character Mode :character Mode :character
```

```
Global_reactive_power  Voltage      Global_intensity Sub_metering_1
Length:2075259  Length:2075259  Length:2075259  Length:2075259
Class :character Class :character Class :character Class :character
Mode  :character Mode :character Mode :character Mode :character
```

```
Sub_metering_2      Sub_metering_3
Length:2075259      Min.   : 0.000
Class :character      1st Qu.: 0.000
Mode  :character      Median : 1.000
                           Mean : 6.458
                           3rd Qu.:17.000
                           Max. :31.000
                           NA's :25979
```

3. Check for the missing values in the dataset and remove if there are any

```
In [..]
data <- data[complete.cases(data)]
```

```
In [..]
sum(is.na(data))
```

```
0
```

4. Convert the date and time to a standard format

```
In [..]
data$datetime <- paste(data$Date,data$Time)
data$datetime <- as.POSIXct(data$datetime, format="%d/%m/%Y %H:%M:%S")
```

```
In [..]
attr(data$datetime, "tzone") <- "Europe/Paris"
```

```
In [..]
str(data)
```

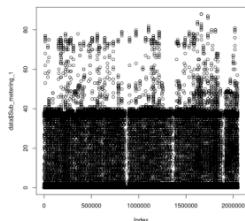
```
Classes 'data.table' and 'data.frame': 2049280 obs. of 10 variables:
$ Date      : chr "16/12/2006" "16/12/2006" "16/12/2006" ...
$ Time      : chr "17:24:00" "17:25:00" "17:26:00" "17:27:00" ...
$ Global_active_power : chr "4.216" "5.360" "5.374" "5.388" ...
$ Global_reactive_power : chr "0.418" "0.436" "0.498" "0.502" ...
$ Voltage    : chr "234.840" "233.630" "233.290" "233.740" ...
$ Global_intensity : chr "18.400" "23.000" "23.000" "23.000" ...
$ Sub_metering_1   : chr "0.000" "0.000" "0.000" "0.000" ...
$ Sub_metering_2   : chr "1.000" "1.000" "2.000" "1.000" ...
$ Sub_metering_3   : num 17 16 17 17 17 17 17 16 ...
$ datetime     : POSIXct, format: "2006-12-16 12:54:00" "2006-12-16 12:55:00" ...
- attr(*, ".internal.selfref")=<externalptr>
```

5. Extract the Year, Week and Day from the dataset

```
In [..]
data$year <- year(data$datetime)
data$week <- week(data$datetime)
data$day <- day(data$datetime)
data$month <- month(data$datetime)
data$minute <- minute(data$datetime)
```

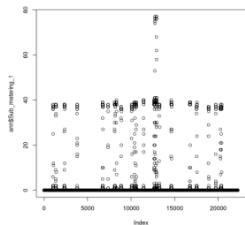
6. Visualize the granularity of the submetering using the plot() function

In [...]: `plot(data$Sub_metering_1)`

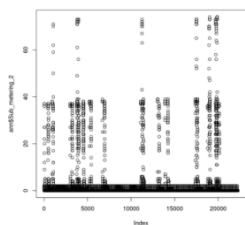


7. Filter the data for any particular year and visualize the submetering for that particular year

In [...]: `# Sub_metering_1
ann <- filter(data, year == 2006)
plot(ann$Sub_metering_1)`

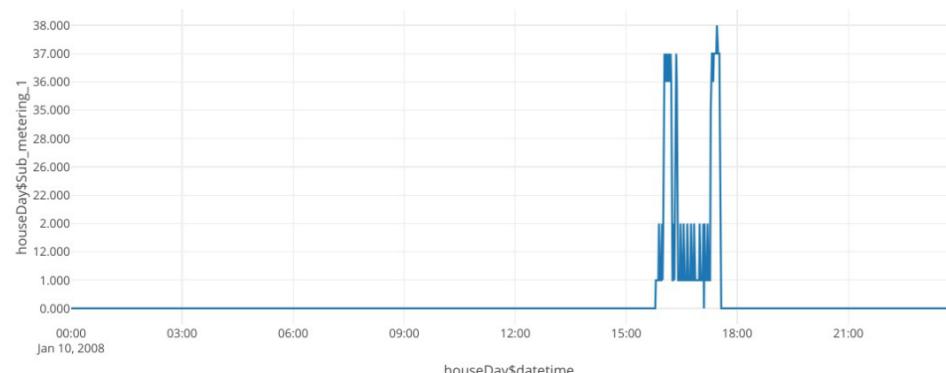


In [...]: `# Sub_metering_2
plot(ann$Sub_metering_2)`



8. Use the plotly package to plot the submetering across a day of usage

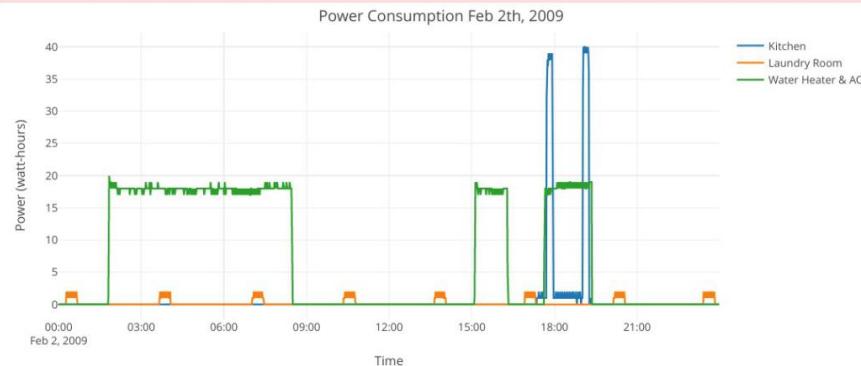
In [16]: `houseDay <- filter(data, year == 2008 & day == 10 & month==1)
plot_ly(houseDay, x = ~houseDay$datetime, y = ~houseDay$Sub_metering_1, type = 'scatter', mode = 'lines')`



```
In [17]: dtDay <- filter(data, year == 2009 & day == 2 & month==2)

In [18]: plot_ly(dtDay, x = ~dtDay$datetime, y = ~dtDay$Sub_metering_1, name = 'Kitchen', type = 'scatter', mode = 'lines') %>
  add_trace(y = ~dtDay$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
  add_trace(y = ~dtDay$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%
  layout(title = "Power Consumption Feb 2th, 2009",
  xaxis = list(title = "Time"),
  yaxis = list (title = "Power (watt-hours)"))

Warning message:
"Can't display both discrete & non-discrete data on same axis"
Warning message:
"Can't display both discrete & non-discrete data on same axis"
```

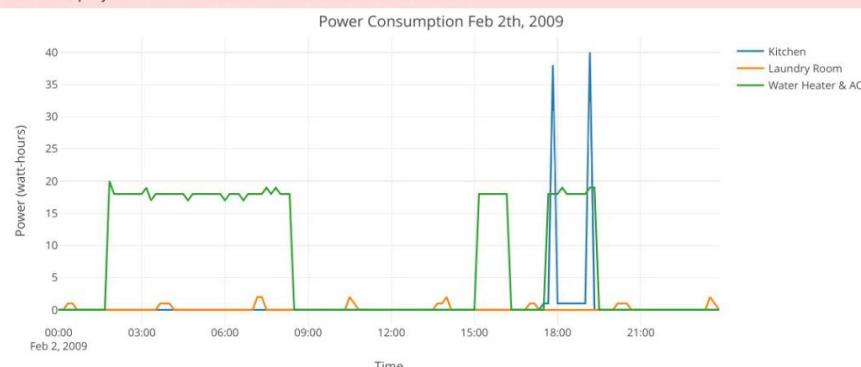


9. Reduce the number of observations for that day and again plot the graph using plotly

```
In [19]: houseDay10 <- filter(data, year == 2009 & month == 2 & day == 2 & (minute == 0 | minute == 10 | minute == 20 | minute == 30))

In [20]: plot_ly(houseDay10, x = ~houseDay10$datetime, y = ~houseDay10$Sub_metering_1, name = 'Kitchen', type = 'scatter', mode = 'lines') %>
  add_trace(y = ~houseDay10$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
  add_trace(y = ~houseDay10$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%
  layout(title = "Power Consumption Feb 2th, 2009",
  xaxis = list(title = "Time"),
  yaxis = list (title = "Power (watt-hours)"))

Warning message:
"Can't display both discrete & non-discrete data on same axis"
Warning message:
"Can't display both discrete & non-discrete data on same axis"
```

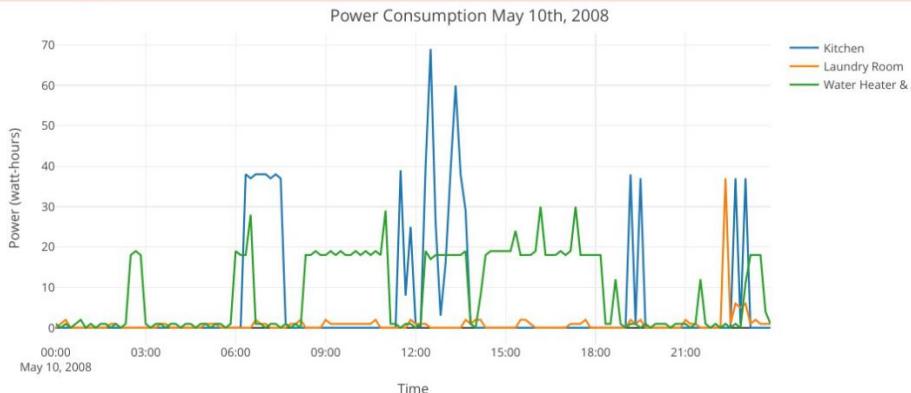


```
In [21]: data$minute <- minute(data$datetime)
```

```
In [22]: houseDay10 <- filter(data, year == 2008 & month == 5 & day == 10 & (minute == 0 | minute == 10 | minute == 20 | minute == 30))
```

```
In [23]: plot_ly(houseDay10, x = ~houseDay10$datetime, y = ~houseDay10$Sub_metering_1, name = 'Kitchen', type = 'scatter', mode = 'lines') %>
  add_trace(y = ~houseDay10$Sub_metering_2, name = 'Laundry Room', mode = 'lines') %>%
  add_trace(y = ~houseDay10$Sub_metering_3, name = 'Water Heater & AC', mode = 'lines') %>%
  layout(title = "Power Consumption May 10th, 2008",
  xaxis = list(title = "Time"),
  yaxis = list (title = "Power (watt-hours)"))
```

```
Warning message:  
"Can't display both discrete & non-discrete data on same axis"  
Warning message:  
"Can't display both discrete & non-discrete data on same axis"
```

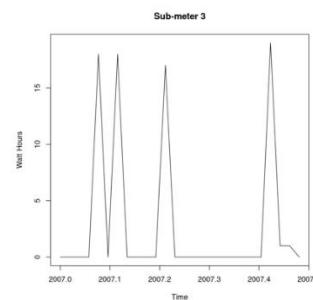


10. For the time series analysis extract the weekly time series data for all the submeters and plot them

```
In [...] data$hour <- hour(data$datetime)
```

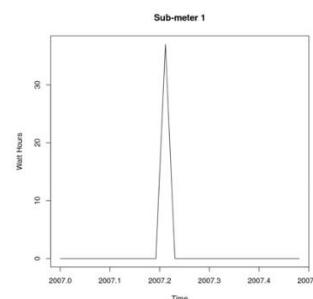
```
In [...] houseweekly <- filter(data, week == 2 & hour == 20 & minute == 1)  
tsSM3_weekly <- ts(houseweekly$Sub_metering_3, frequency=52, start=c(2007,1))
```

```
In [...] plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 3")
```



```
In [...] tsSM3_weekly <- ts(houseweekly$Sub_metering_1, frequency=52, start=c(2007,1))
```

```
In [...] plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 1")
```

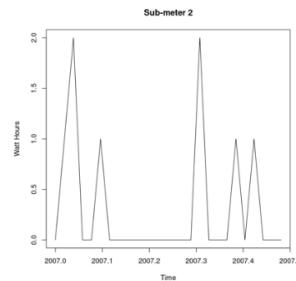


```
In [...] tsSM3_weekly <- ts(houseweekly$Sub_metering_2, frequency=52, start=c(2007,1))
```

```
In [...] plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 2")
```

```
In [...] tsSM3_weekly <- ts(houseweekly$Sub_metering_2, frequency=52, start=c(2007,1))

In [...] plot(tsSM3_weekly, xlab = "Time", ylab = "Watt Hours", main = "Sub-meter 2")
```



11. Fit the time series data into a time series linear regression model

```
In [...] house070809weekly <- filter(data, year == 2008, hour == 20 & minute == 1)
tsSM3_070809weekly <- ts(house070809weekly$Sub_metering_3, frequency=52, start=c(2008,3))
tsSM2_070809weekly <- ts(house070809weekly$Sub_metering_2, frequency=52, start=c(2008,3))
tsSM1_070809weekly <- ts(house070809weekly$Sub_metering_1, frequency=52, start=c(2008,3))

In [...] fit3 <- tslm(tsSM3_070809weekly ~ trend + season)
fit2 <- tslm(tsSM2_070809weekly ~ trend + season)
fit1 <- tslm(tsSM1_070809weekly ~ trend + season)
```

12. View the summary of the model

```
In [...] summary(fit3)

Call:
tslm(formula = tsSM3_070809weekly ~ trend + season)

Residuals:
    Min      1Q      Median      3Q      Max 
-8.0997 -3.6021 -2.0930  0.3787 25.2358 

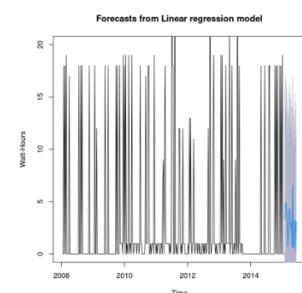
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 5.2300695  2.6910798  1.943  0.0529 .  
trend       0.0009589  0.0034285  0.280  0.7799    
season2     -5.0009589  3.6710347 -1.362  0.1741    
season3     -3.0305567  3.5554148 -0.852  0.3947    
season4      0.9684843  3.5553371  0.272  0.7855    
season5     -2.2396848  3.6747199 -0.609  0.5426    
season6     -0.6692152  3.6745680 -0.182  0.8556    
season7     -0.6132917  3.4788169 -0.221  0.8556    
season50    -2.7114089  3.6710475 -0.739  0.4607    
season51     0.1447750  3.6710395  0.039  0.9686    
season52     -4.9990411  3.6710347 -1.362  0.1743    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.868 on 313 degrees of freedom
Multiple R-squared:  0.09806,   Adjusted R-squared:  -0.05179 
F-statistic: 0.6544 on 52 and 313 DF,  p-value: 0.9678
```

13. Plot the forecast for all 3 submeters

```
In [...] forecastfitSM3c <- forecast(fit3, h=20, level=c(80,90))
forecastfitSM2c <- forecast(fit2, h=20, level=c(80,90))
forecastfitSM1c <- forecast(fit1, h=20, level=c(80,90))

In [...] plot(forecastfitSM3c, ylim = c(0, 20), ylab= "Watt-Hours", xlab="Time")
```



Constructing the Random Forest Model

```
In [...]
rf <- randomForest(target ~ trestbps + oldpeak + cp, data = data)
# View the forest results
print("Random Forest Results:")
print(rf)

[1] "Random Forest Results:

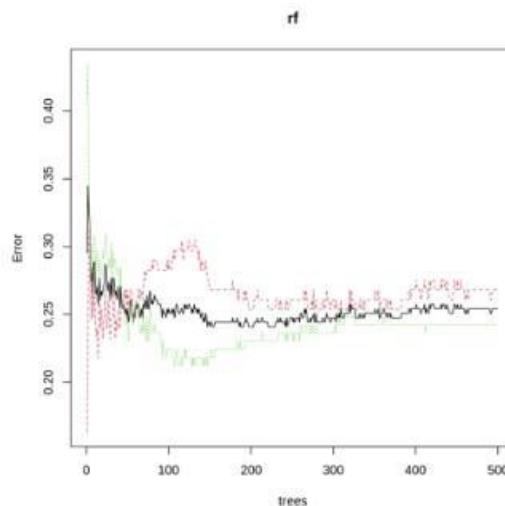
Call:
randomForest(formula = target ~ trestbps + oldpeak + cp, data = data)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 1

      OOB estimate of  error rate: 25.41%
Confusion matrix:
  0   1 class.error
0 101  37  0.2681159
1  40 125  0.2424242
```

```
In [...]
# Importance of each predictor
print("Importance of each predictor:")
print(importance(rf,type = 2))
```

```
# Plot the Random Forest
plot(rf)
```

```
[1] "Importance of each predictor:"
  MeanDecreaseGini
trestbps           21.85269
oldpeak            35.36782
cp                 36.12227
```



Case Studies 1: Use ggplot2 in R to visualize the distribution of petal length with respect to Species in default iris dataset. And apply correlation plot (Marginal Histogram / Boxplot, Correlogram, Diverging bars, Diverging Lollipop Chart, Diverging Dot Plot and Area Chart) in R to show the correlation among the all petal width and sepal width attributes in Iris dataset.

Create Sheets with one Dashboard in tableau to explore the above dataset. Apply R and Tableau integration to group the species type based on the attributes and show the results with visual impact.

```
#Exploratory Analysis of the Iris Dataset
#number of rows
nrow(iris)
## [1] 150

#Length of the dataset
length(iris)
## [1] 5

#Mean of an attribute
mean(iris$Sepal.Length)
## [1] 5.843333

#variance
var(iris$Sepal.Length)
## [1] 0.6856935

#Standard deviation
sd(iris$Sepal.Length)
```

```
## [1] 0.8280661
```

#Standard error

```
standard_error = sd(iris$Sepal.Length)/sqrt(length(iris  
$Sepal.Length))  
standard_error
```

```
## [1] 0.06761132
```

#Mean absolute error

```
mean_absolute_deviation = mad(iris$Sepal.Length)  
mean_absolute_deviation
```

```
## [1] 1.03782
```

#median

```
median(iris$Sepal.Length)
```

```
## [1] 5.8
```

#min

```
min(iris$Sepal.Length)
```

```
## [1] 4.3
```

#max

```
max(iris$Sepal.Length)
```

```
## [1] 7.9
```

#Quantile

```
quantile(iris$Sepal.Length,c(0.25,0.50,0.75))
```

```
## 25% 50% 75%
## 5.1 5.8 6.4

#IQR
IQR(iris$Sepal.Length)

## [1] 1.3

#Correlation between Sepal Length and petal Length
cor(iris$Sepal.Length,iris$Petal.Length, method = "pearson") #->
pearson, Kendall, spearman

## [1] 0.8717538

typeof(iris)

## [1] "list"

#Structure of the dataset
```

```
str(iris)

## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 4.6 5 4.4 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 3.4 2.9
3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4
1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2
0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...
1 1 1 1 1 1 1 1 1 1 ...
```

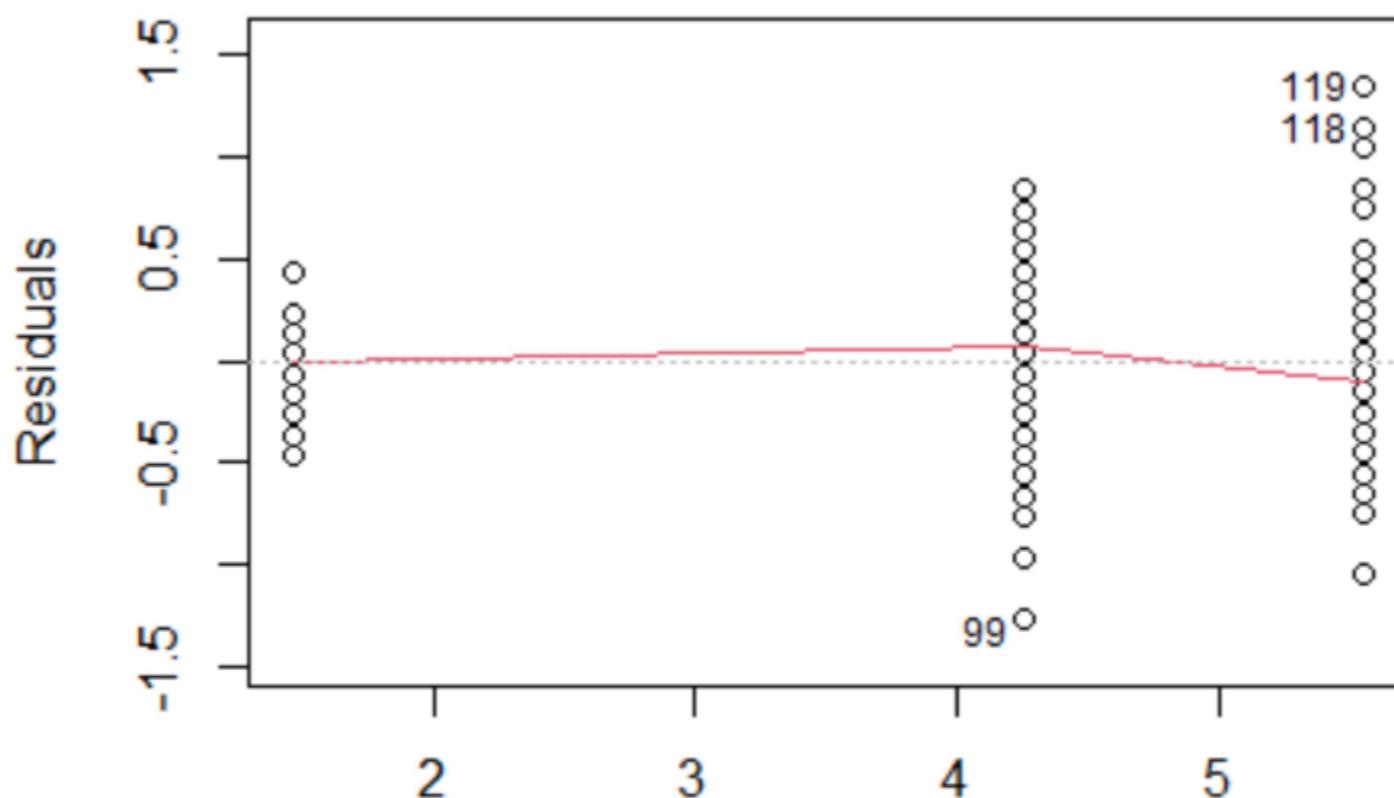
#Linear model with the Iris dataset

```
library(ggplot2)
model<-lm(iris$Petal.Length~iris$Species)
model

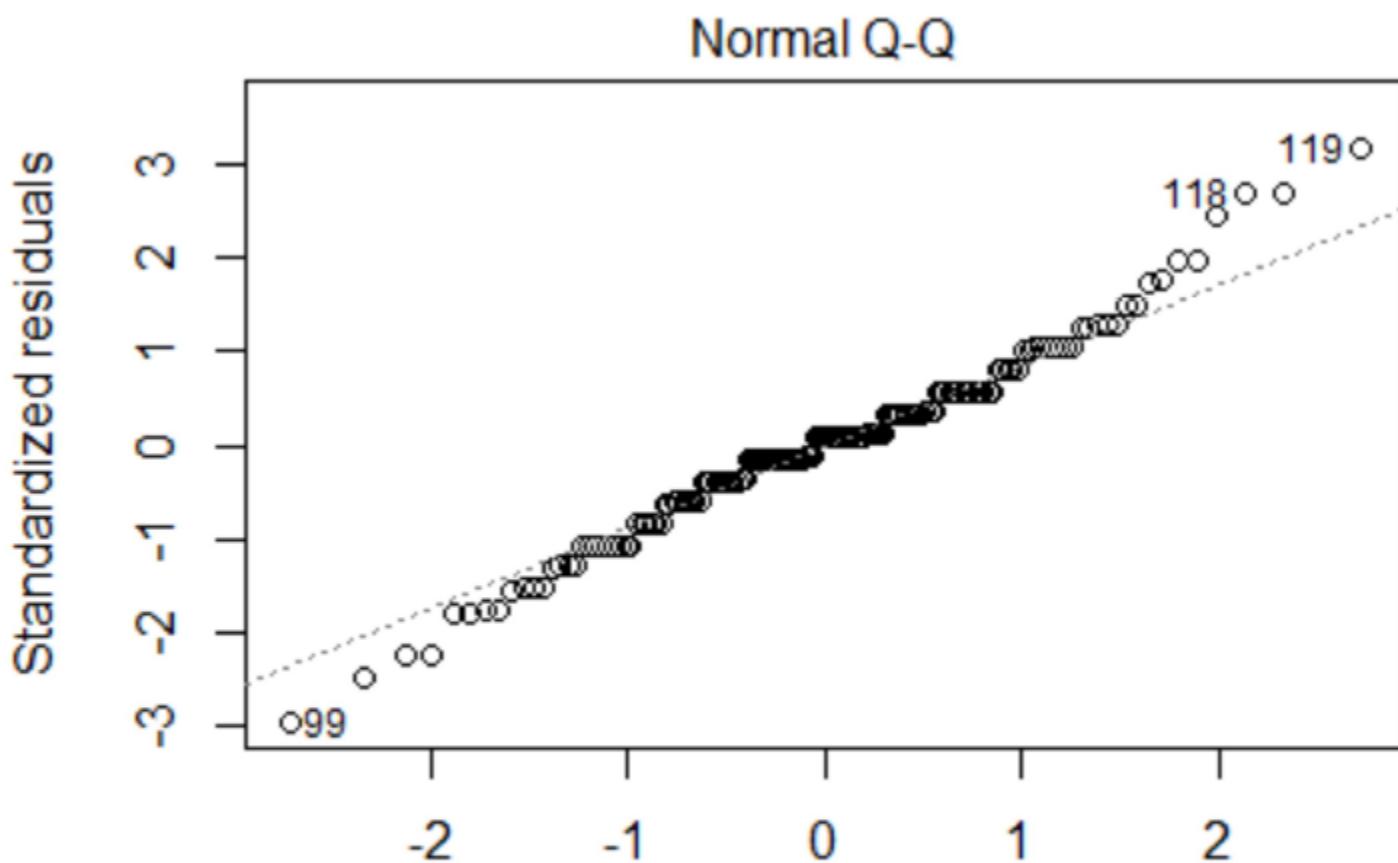
##
## Call:
## lm(formula = iris$Petal.Length ~ iris$Species)
##
## Coefficients:
## (Intercept)  iris$Speciesversicolor  iris
## $Speciesvirginica
##                               1.462                  2.798
## 4.090

plot(model)
```

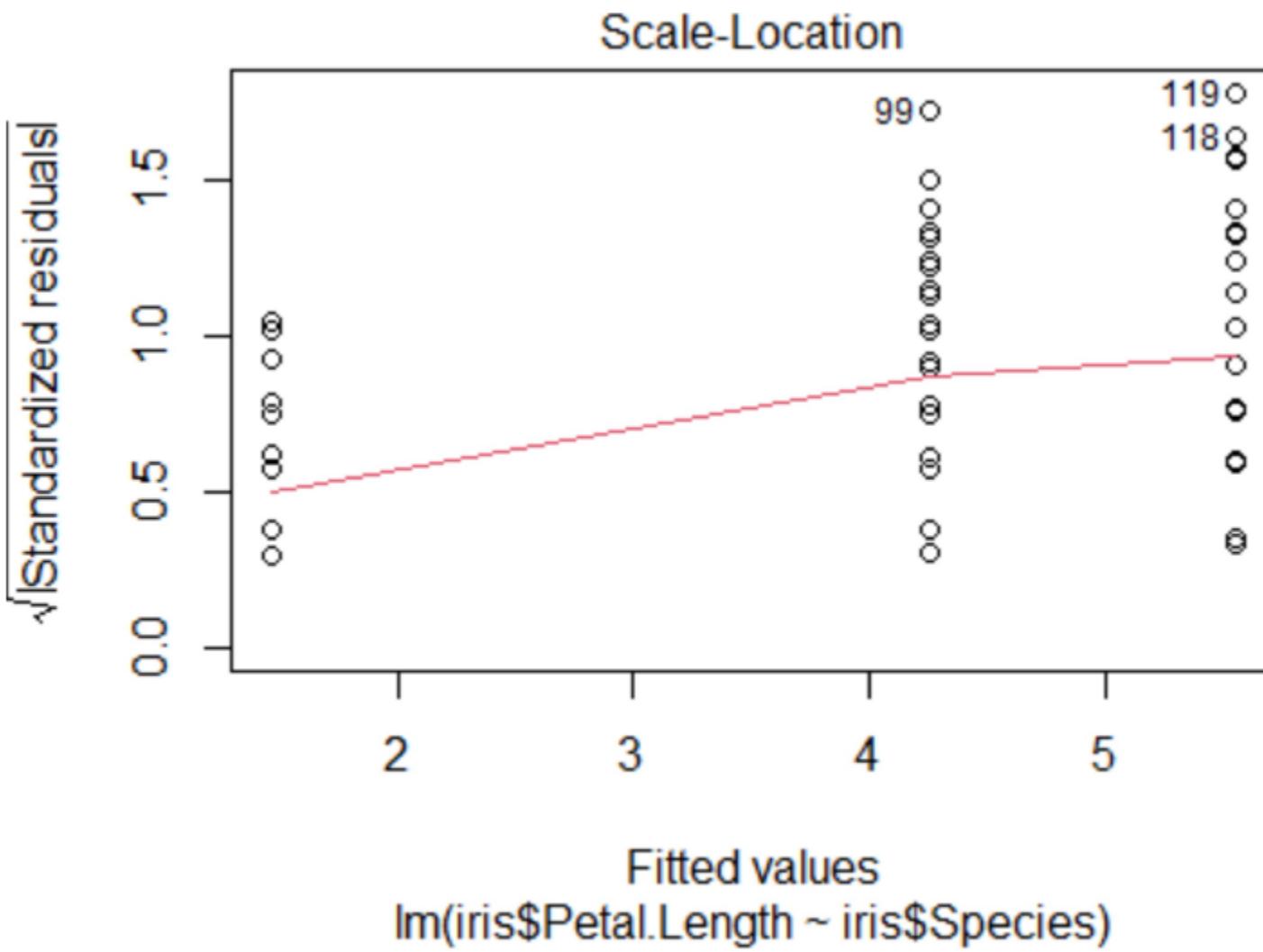
Residuals vs Fitted



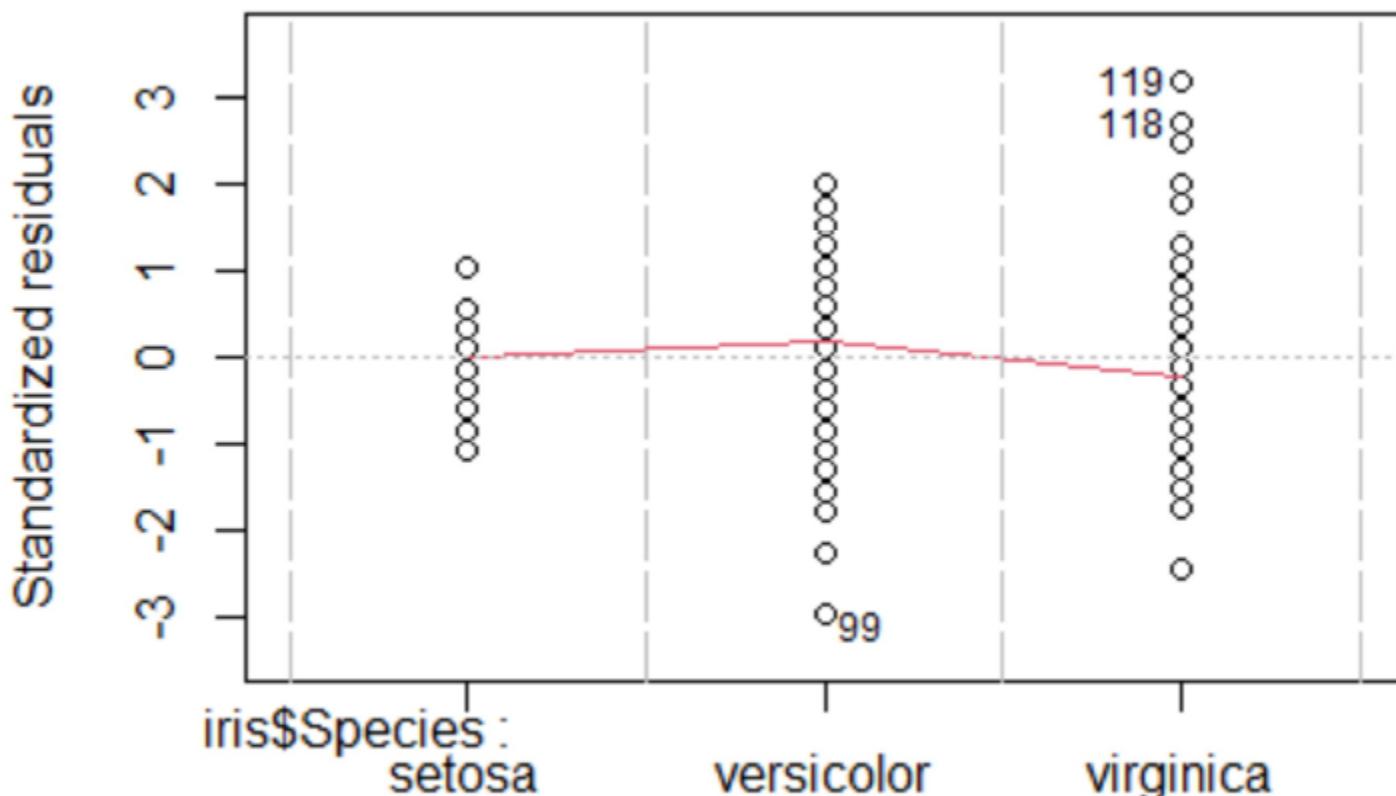
Fitted values
`lm(iris$Petal.Length ~ iris$Species)`



Theoretical Quantiles
 $\text{lm}(\text{iris\$Petal.Length} \sim \text{iris\$Species})$



Constant Leverage: Residuals vs Factor Levels



Factor Level Combinations

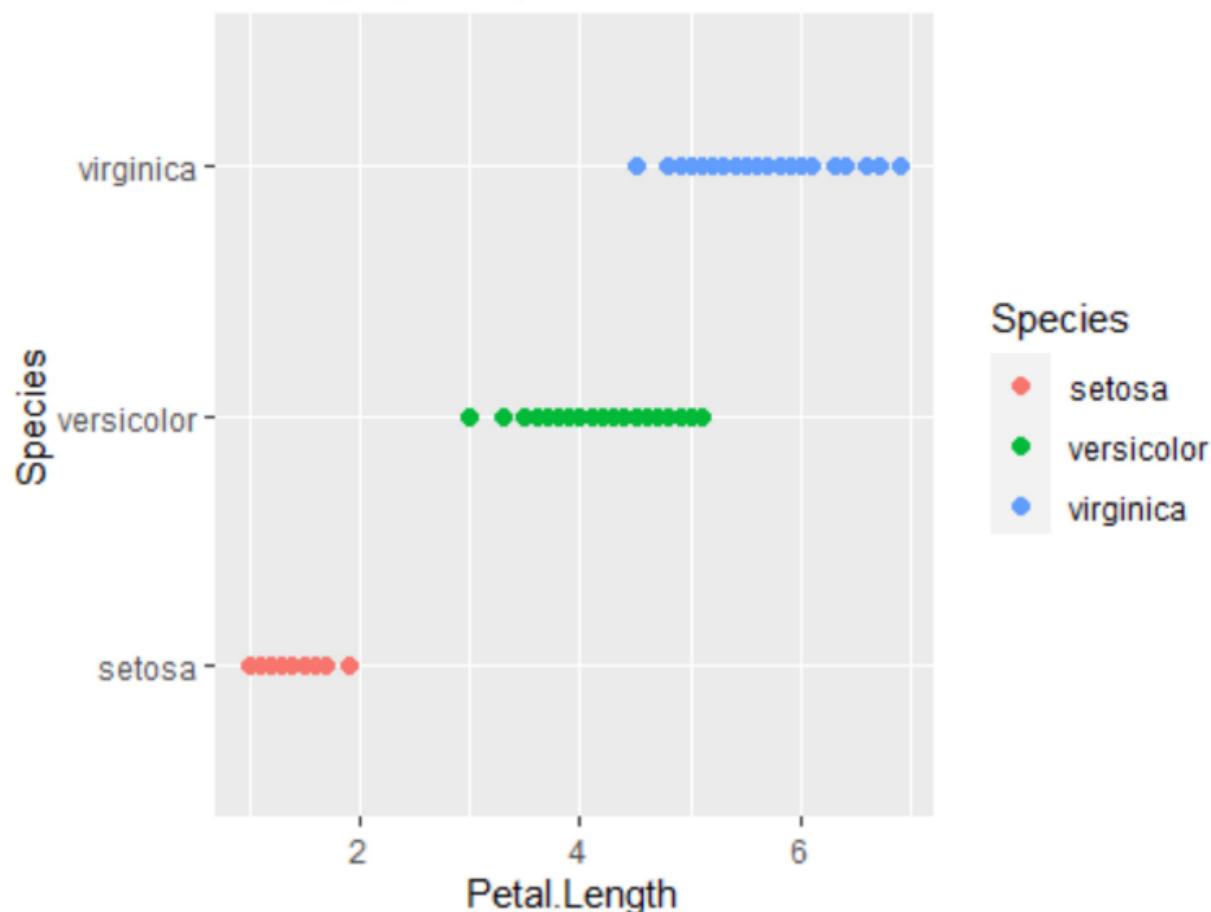
/

```
summary(model)
```

```
##
```

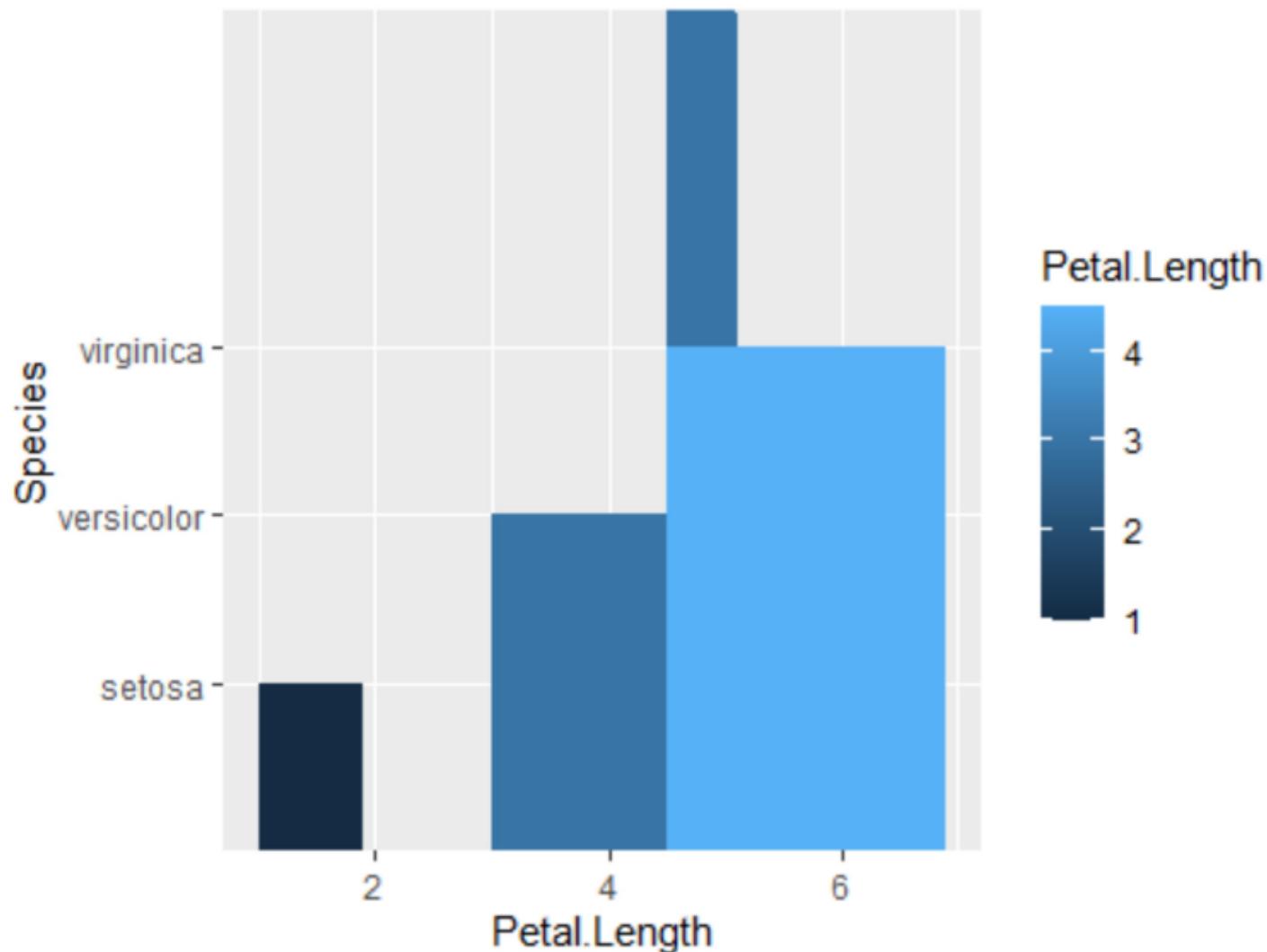
```
## Call:  
## lm(formula = iris$Petal.Length ~ iris$Species)  
##  
## Residuals:  
##      Min      1Q Median      3Q     Max  
## -1.260 -0.258  0.038  0.240  1.348  
##  
## Coefficients:  
##                               Estimate Std. Error t value Pr(>|t|)  
## (Intercept)             1.46200   0.06086  24.02 <2e-16  
***  
## iris$Speciesversicolor 2.79800   0.08607  32.51 <2e-16  
***  
## iris$Speciesvirginica  4.09000   0.08607  47.52 <2e-16  
***  
  
## ---  
## Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 0.4303 on 147 degrees of freedom  
## Multiple R-squared:  0.9414, Adjusted R-squared:  0.9406  
## F-statistic: 1180 on 2 and 147 DF,  p-value: < 2.2e-16  
  
#diverging dot plot  
ggplot(iris, aes(x=Petal.Length, y=Species))  
+geom_point(stat='identity', aes(col=Species), size=2) +  
labs(title = "Diverging Dot plot")
```

Diverging Dot plot



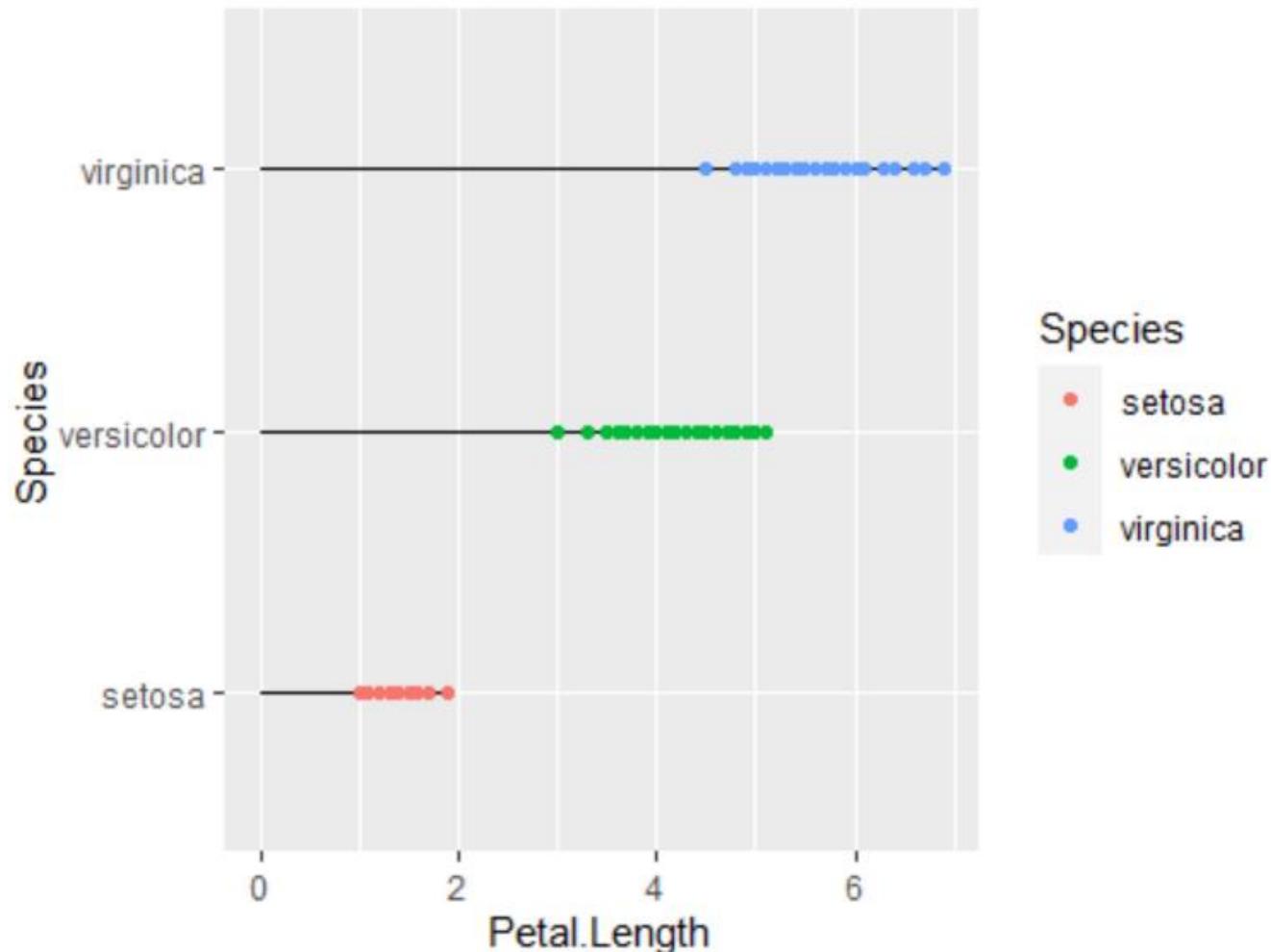
```
#Area plot
#Area chart
plot<-
ggplot(iris,aes(x=Petal.Length,y=Species,fill=Petal.Length))+geom_area()+labs(title = "Area Chart")
plot
```

Area Chart



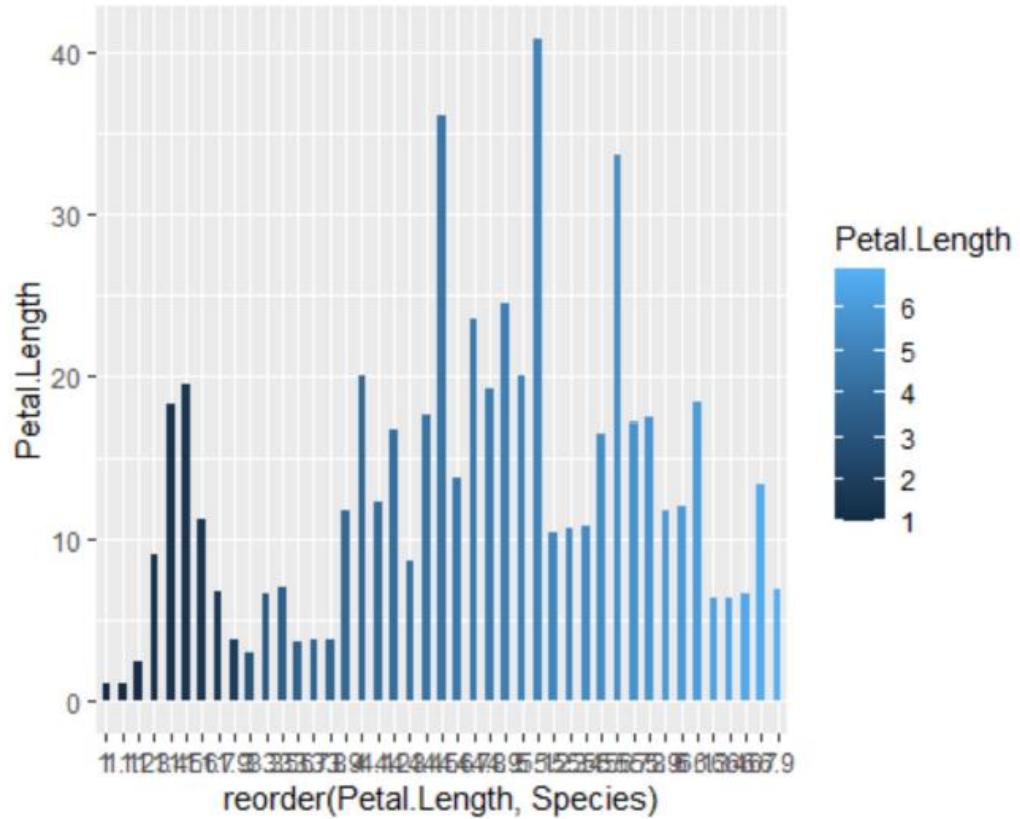
```
#Diverging Lollipop chart
ggplot(iris,aes(x=Petal.Length,y=Species))+geom_segment(aes(x=
0,y=Species,xend=Petal.Length,yend=Species))+geom_point(aes(col=S
pecies))+labs(title = "Diverging lollipop chart")
```

Diverging lollipop chart



#Diverging Bar

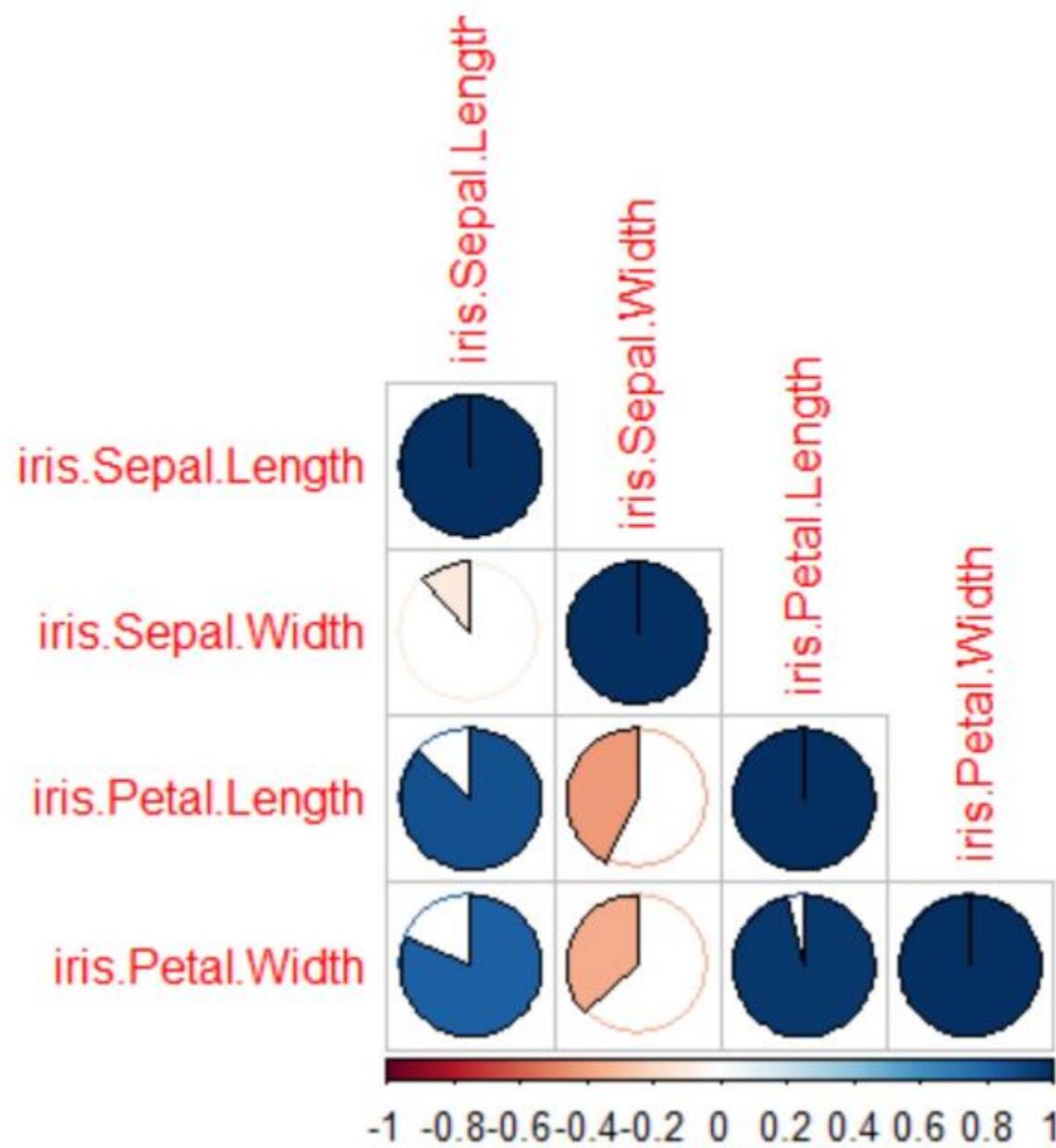
```
library(ggplot2)
ggplot(iris,aes(x=reorder(Petal.Length,Species),fill=Petal.Length
,y=Petal.Length))+geom_bar(stat='identity',
width=0.5)
```



```
#Correlation plot
# correlogram
library(corrplot)

## corrplot 0.92 loaded

df = data.frame(iris$Sepal.Length,iris$Sepal.Width, iris
$Petal.Length,iris$Petal.Width)
c = cor(df)
corrplot(c, method = "pie", type = "lower")
```



Chapter 4

Connecting R and Tableau

R and Tableau

K-means clustering R code:

```
SCRIPT_INT("
kmeans(data.frame(.arg1,.arg2,.arg3,.arg4), centers = 4)$cluster
",
SUM([Petal.Length]),
SUM([Petal.Width]),
SUM([Sepal.Length]),
SUM([Sepal.Width])
)
```

Clustering

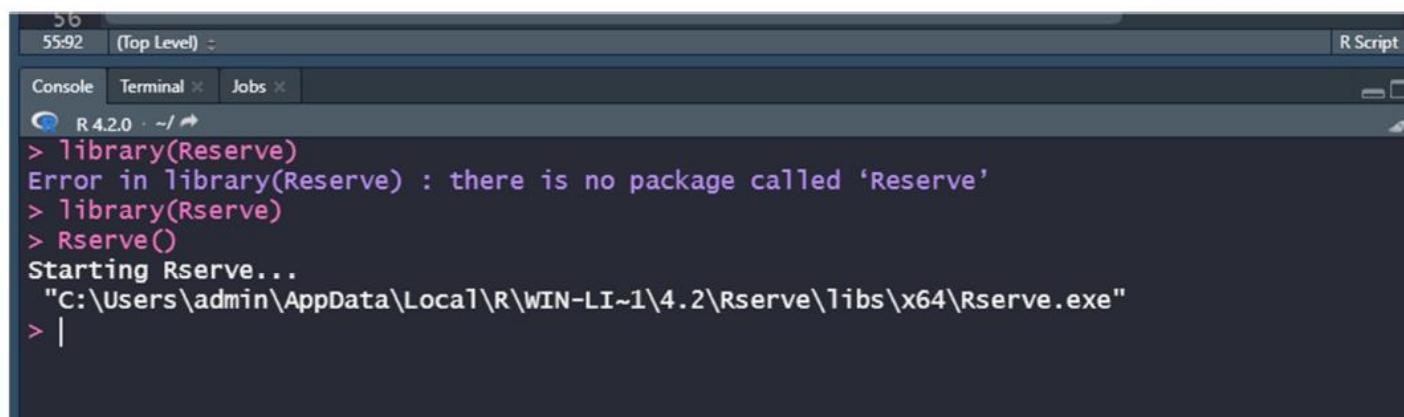
Role: Continuous Measure
Type: Calculated Field
Status: Valid

Formula

```
SCRIPT_INT("OUTPUT<-
kmeans(.arg1,centers=c('setosa','virginica','versicolor');result$cluster",SUM
```

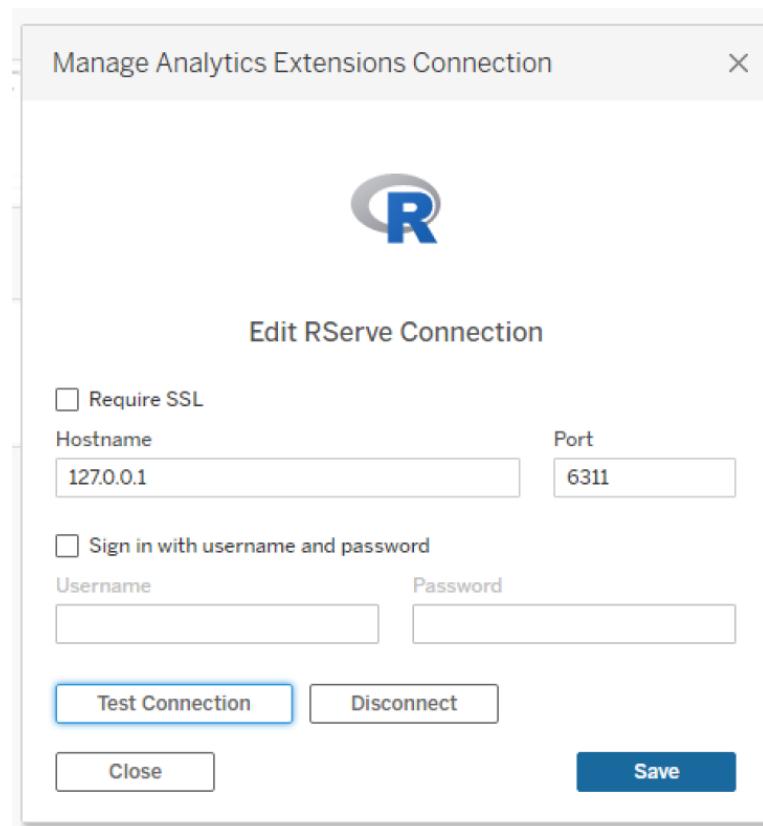
```
SCRIPT_INT("OUTPUT<-
kmeans(.arg1,centers=c('setosa','virginica','versicolor');result$cluster",SUM([Petal.Length]))
```

Q2:Integration of r with tableau using rserve package



The screenshot shows the RStudio interface with the R console tab selected. The session title is "56" and the status is "5592 (Top Level)". The R version is "R 4.2.0". The console output shows the following text:

```
> library(Reserve)
Error in library(Reserve) : there is no package called 'Reserve'
> library(Rserve)
> Rserve()
Starting Rserve...
"C:\Users\admin\AppData\Local\R\WIN-LI~1\4.2\Rserve\libs\x64\Rserve.exe"
> |
```

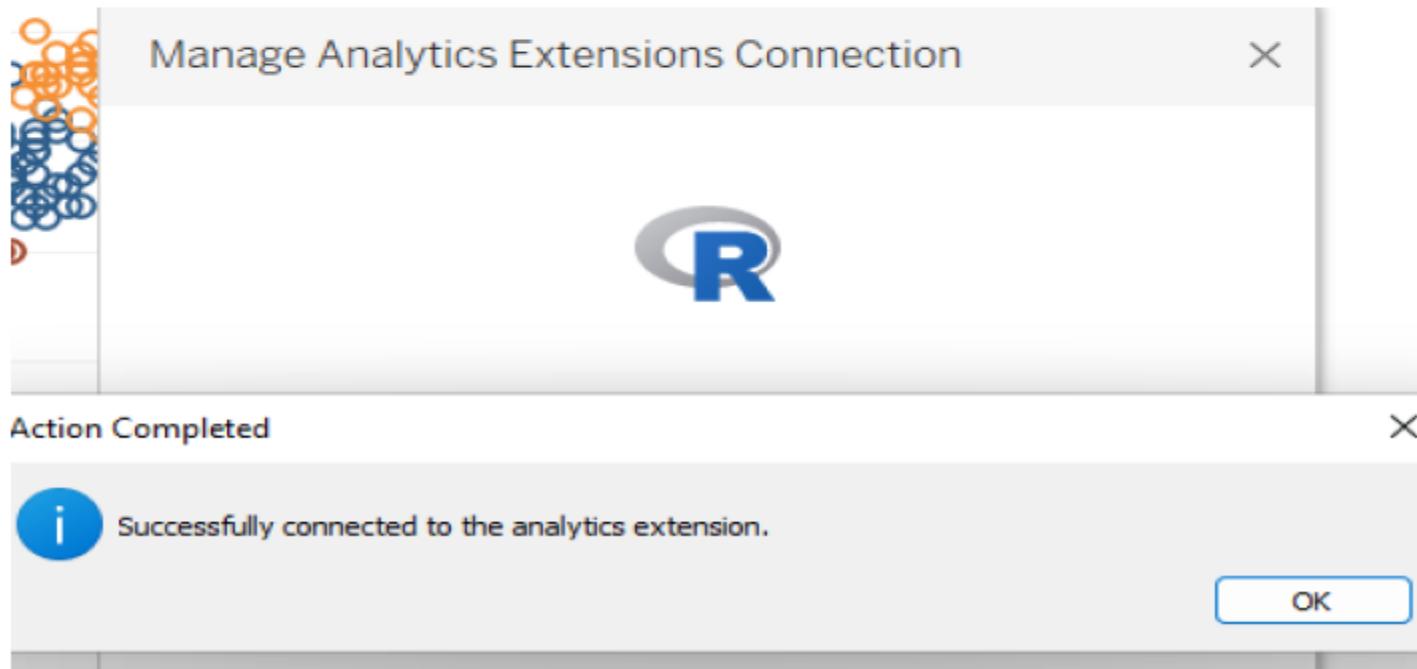


Connection:

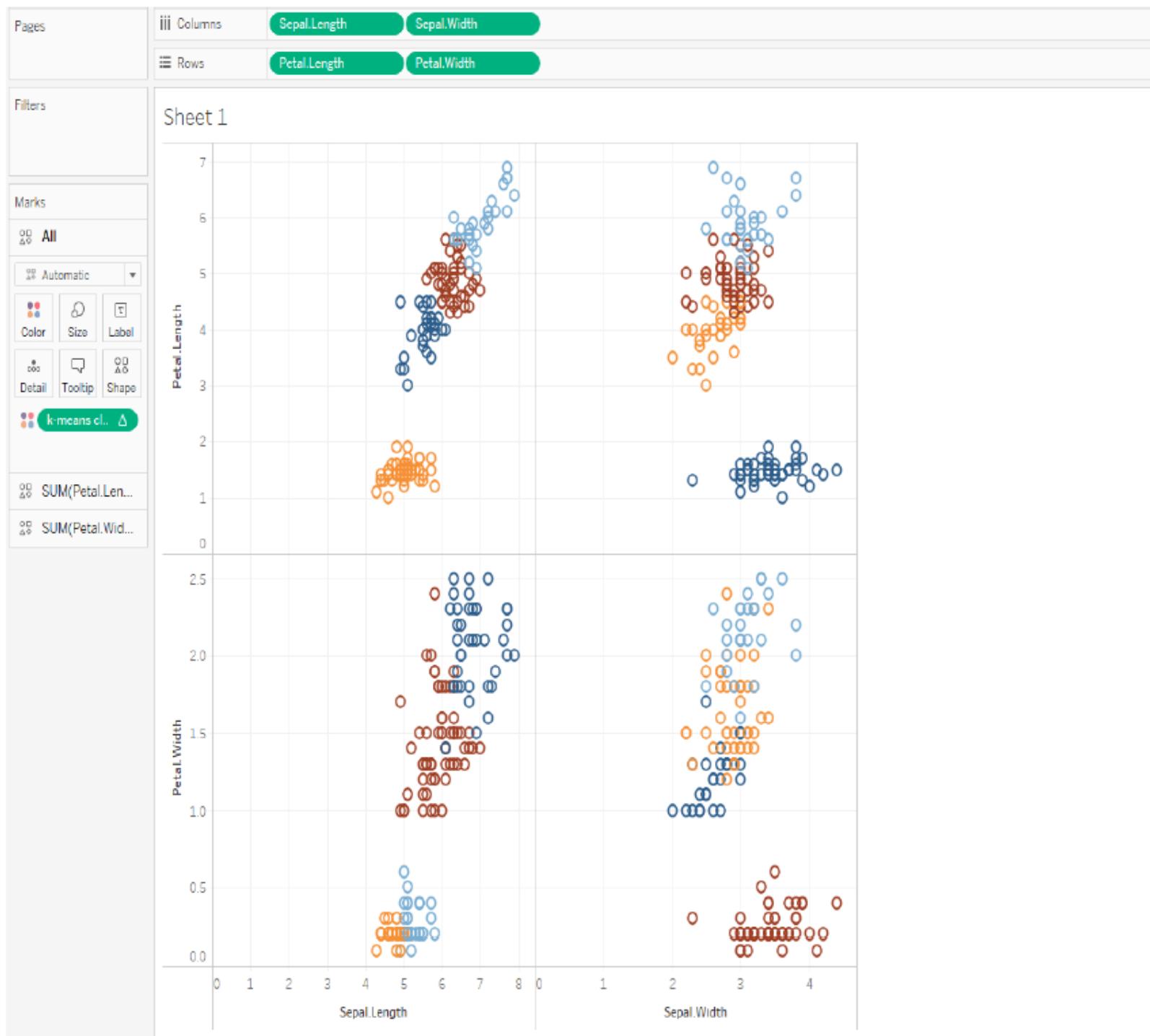
```
write.csv(iris, "C:/Users/admin/Desktop/20bce1216/iris.csv")
```

```
library(Rserve)
```

```
Rserve()
```



Clustering Output:

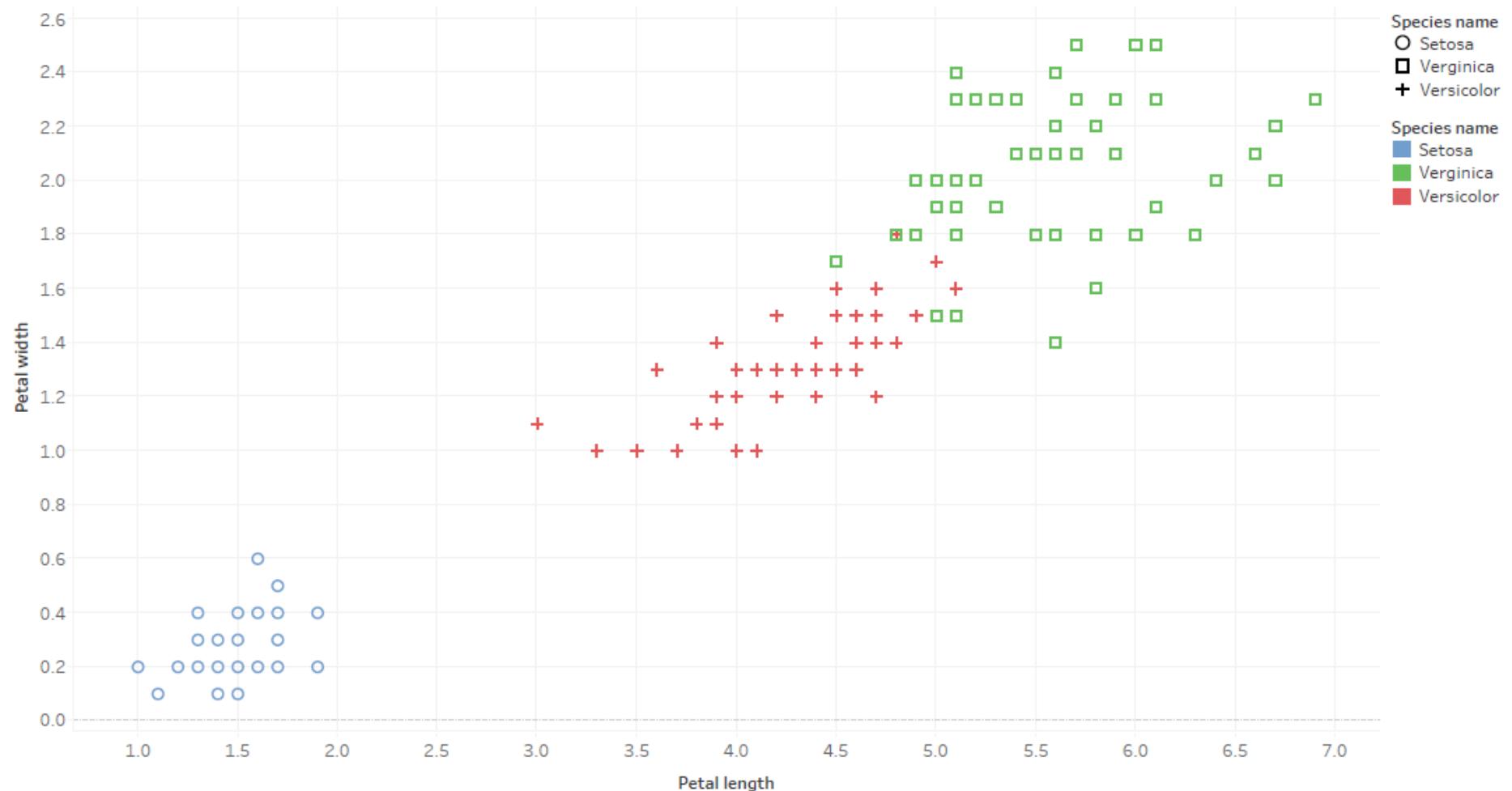


Dashboard:



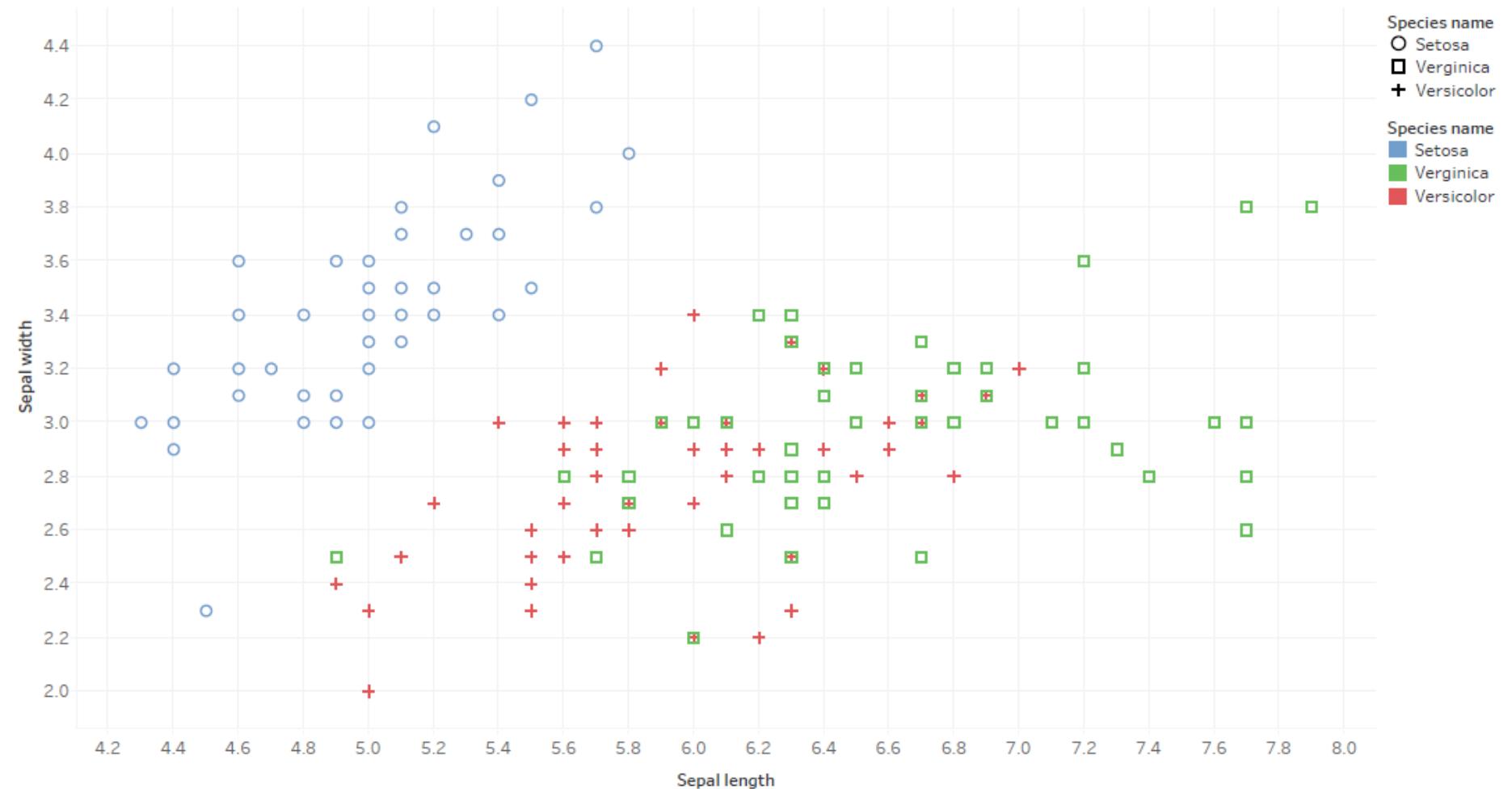
```
install.packages("ggRxtra")  
  
library(ggExtra)  
  
plot<-  
ggplot(iris,aes(x=Petal.Length,y=Species))+geom_point()+theme(legend.position  
="none")  
  
plot  
  
#ggMarginal  
  
plot1<-ggMarginal(plot,type="histogram")  
  
plot2 <-ggMarginal(plot, type="boxplot")
```

Petal Width vs Petal Length



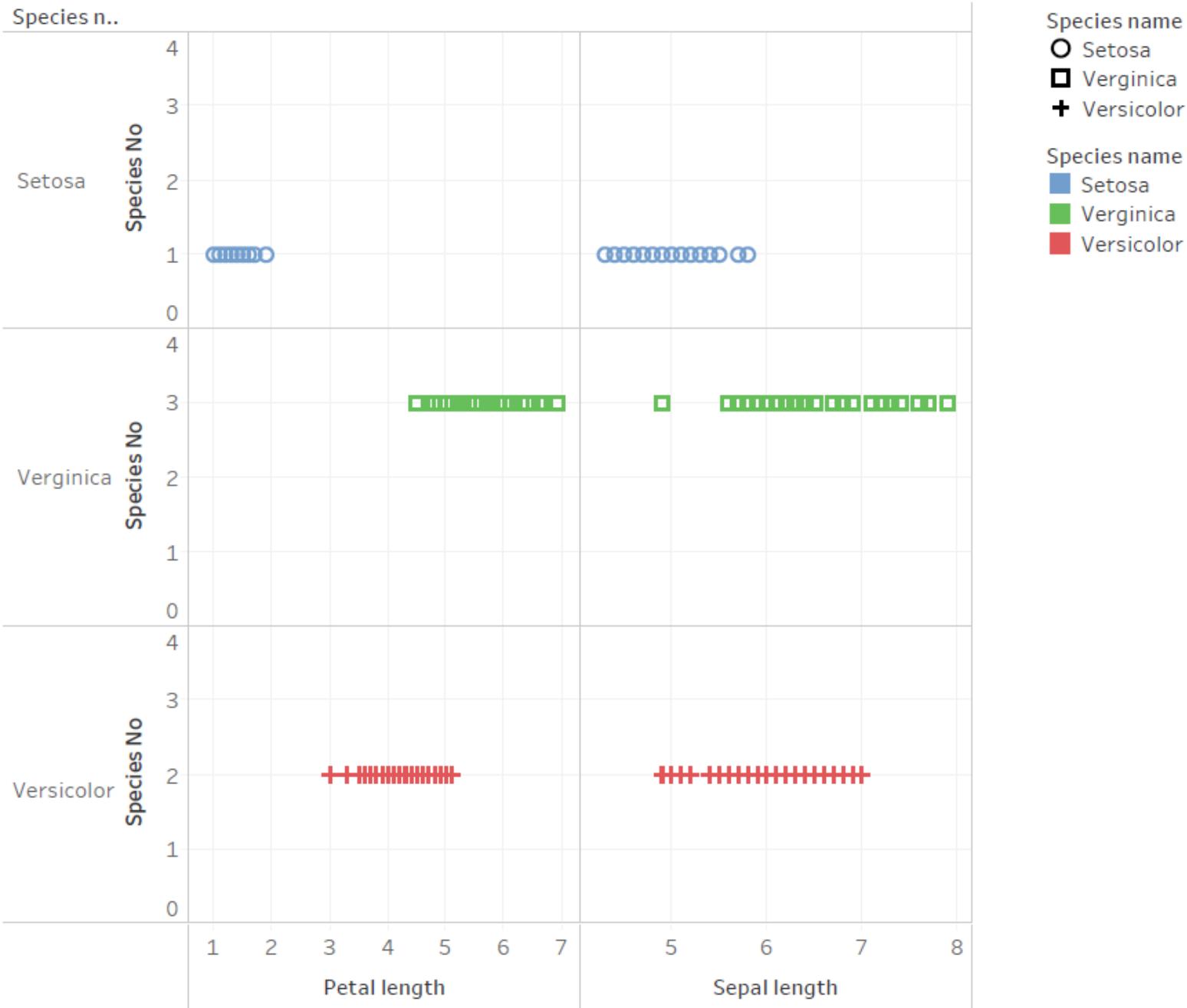
Petal length vs. Petal width. Color shows details about Species name. Shape shows details about Species name.

Sepal Length vs Sepal Width



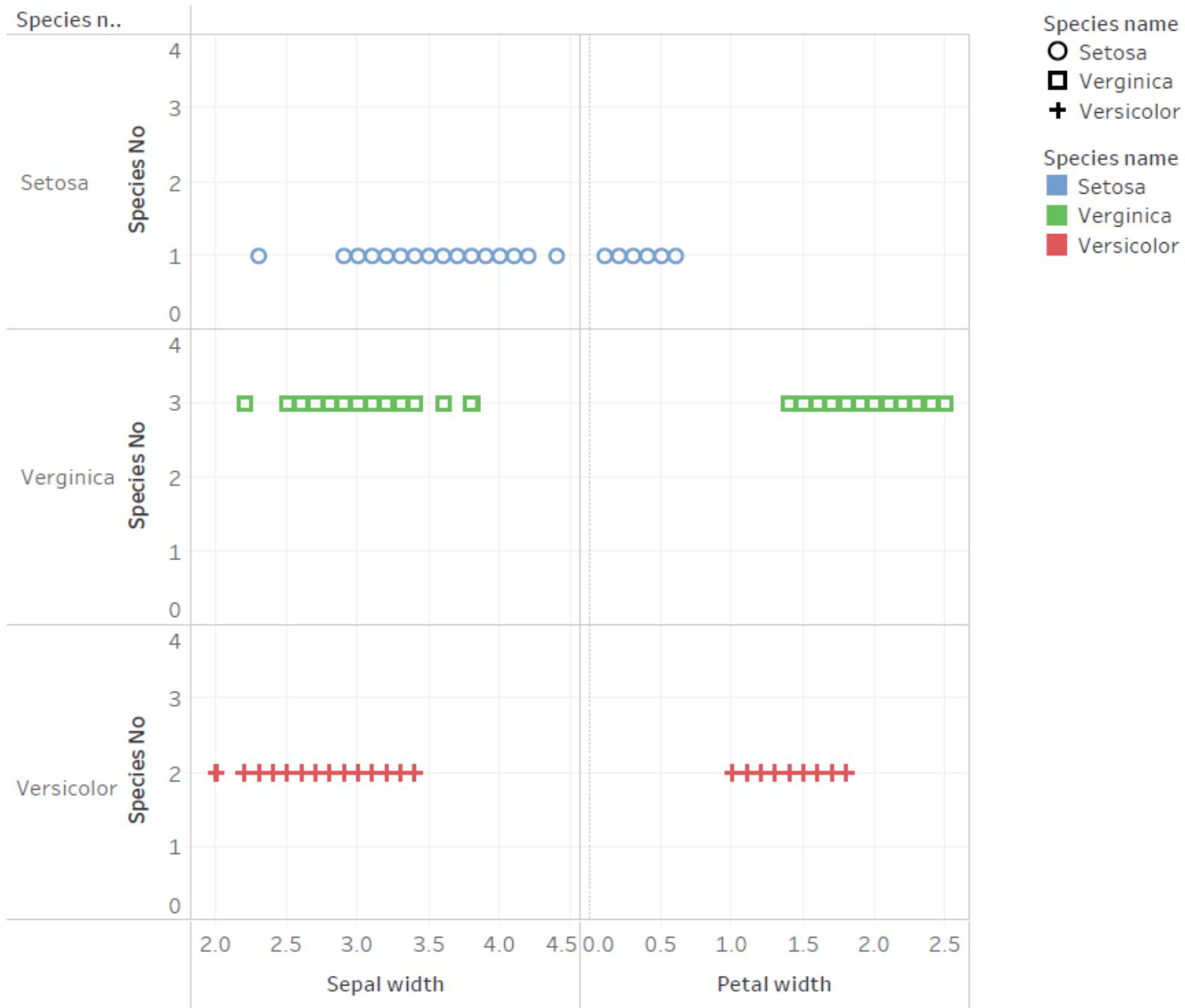
Sepal length vs. Sepal width. Color shows details about Species name. Shape shows details about Species name.

Species vs all features



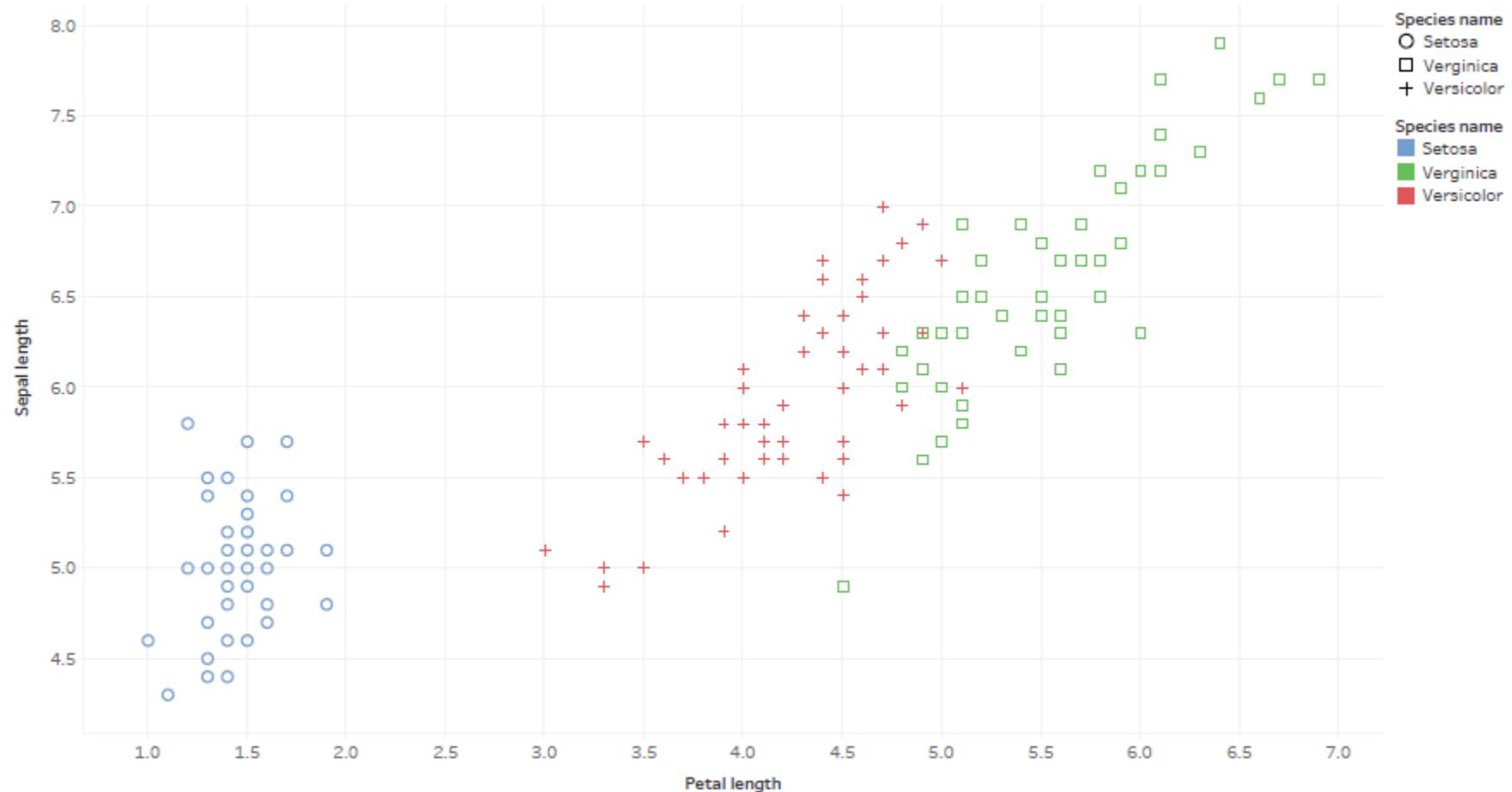
Petal length, Sepal length, Sepal width and Petal width vs. Species No broken down by Species name.
Color shows details about Species name. Shape shows details about Species name. Details are shown
for Species name.

Species vs all features



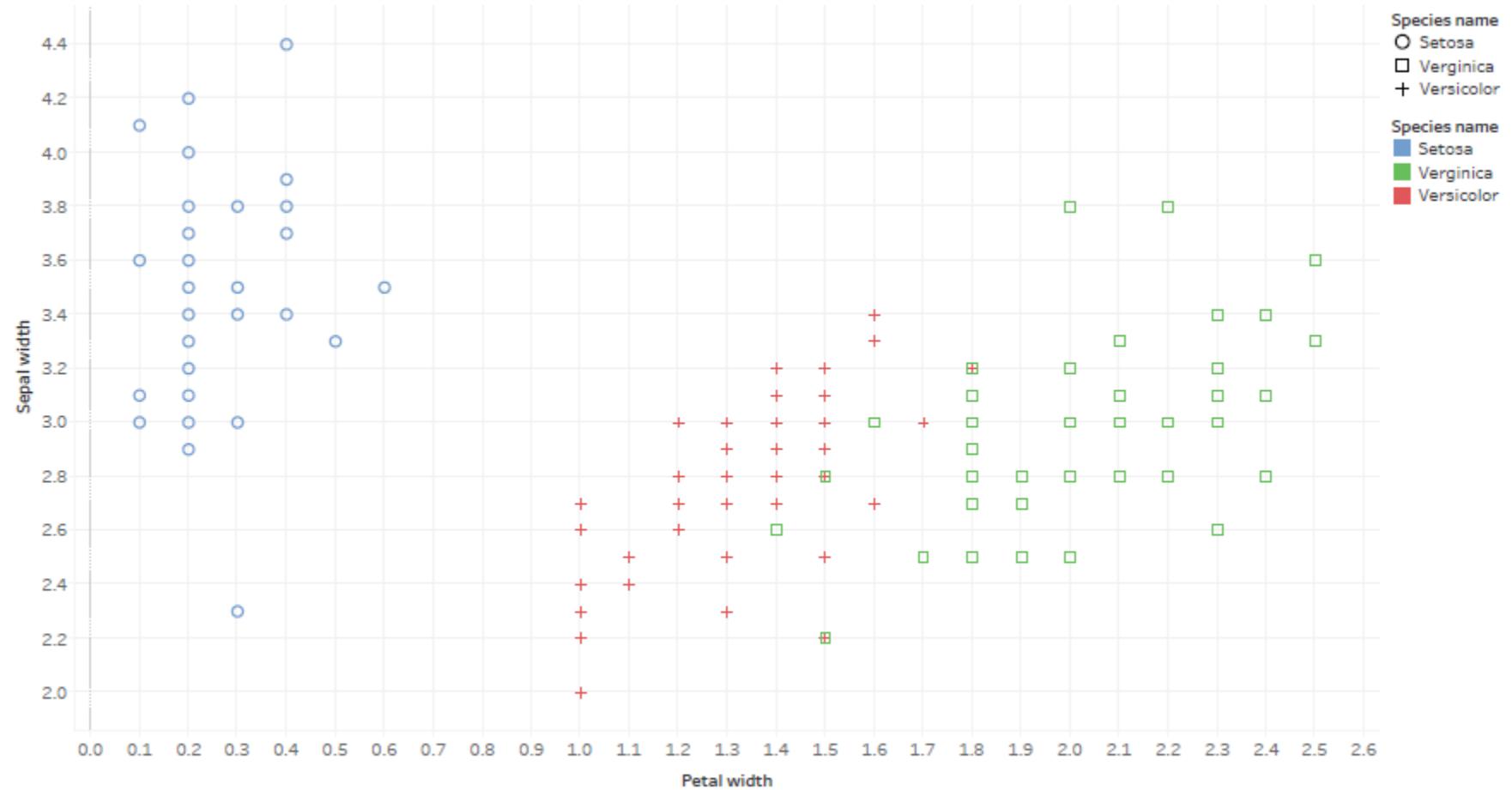
Petal length, Sepal length, Sepal width and Petal width vs. Species No broken down by Species name.
Color shows details about Species name. Shape shows details about Species name. Details are shown for Species name.

Petal Length vs Sepal Length



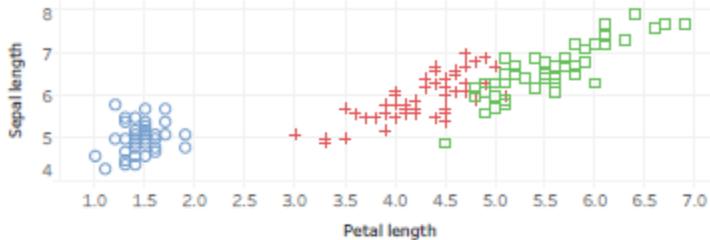
Petal length vs. Sepal length. Color shows details about Species name. Shape shows details about Species name.

Petal Width vs Sepal Width

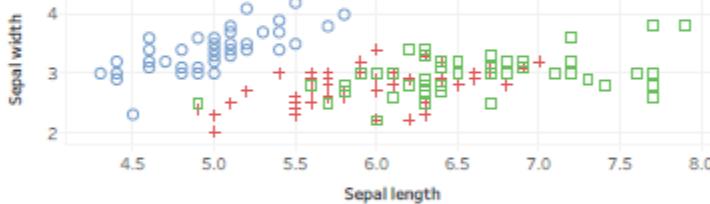


Petal width vs. Sepal width. Color shows details about Species name. Shape shows details about Species name.

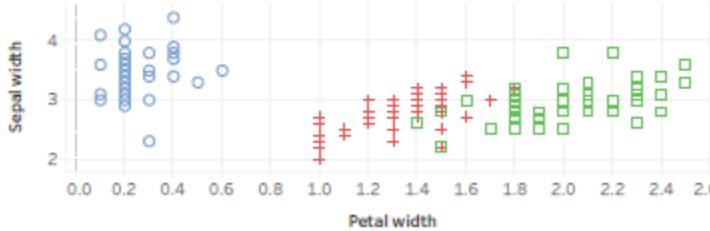
Petal Length vs Sepal Length



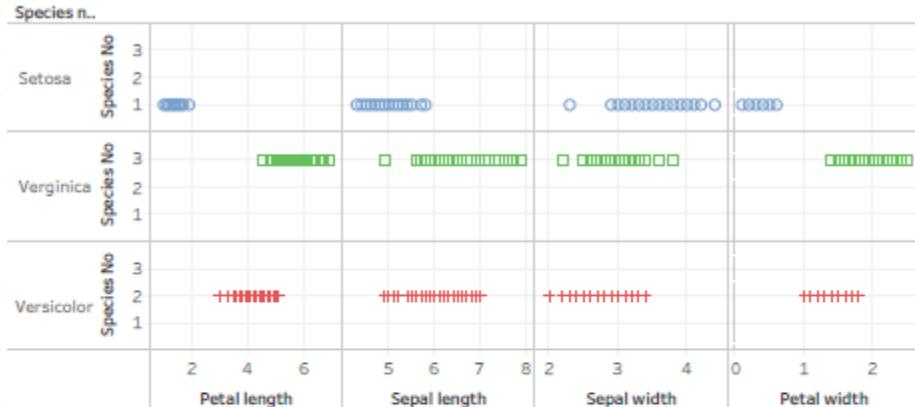
Sepal Length vs Sepal Width



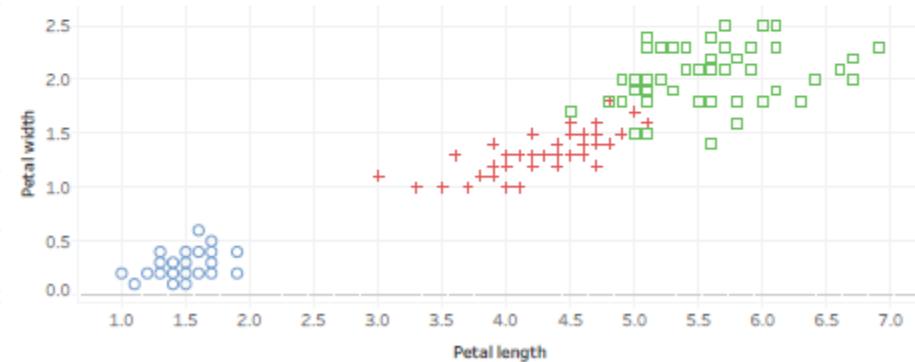
Petal Width vs Sepal Width



Species vs all features



Petal Width vs Petal Length



Case Study 2:

Visualize the frequency of bad credit rate obtained by different age group using histograms for a given “Loan dataset” in R.

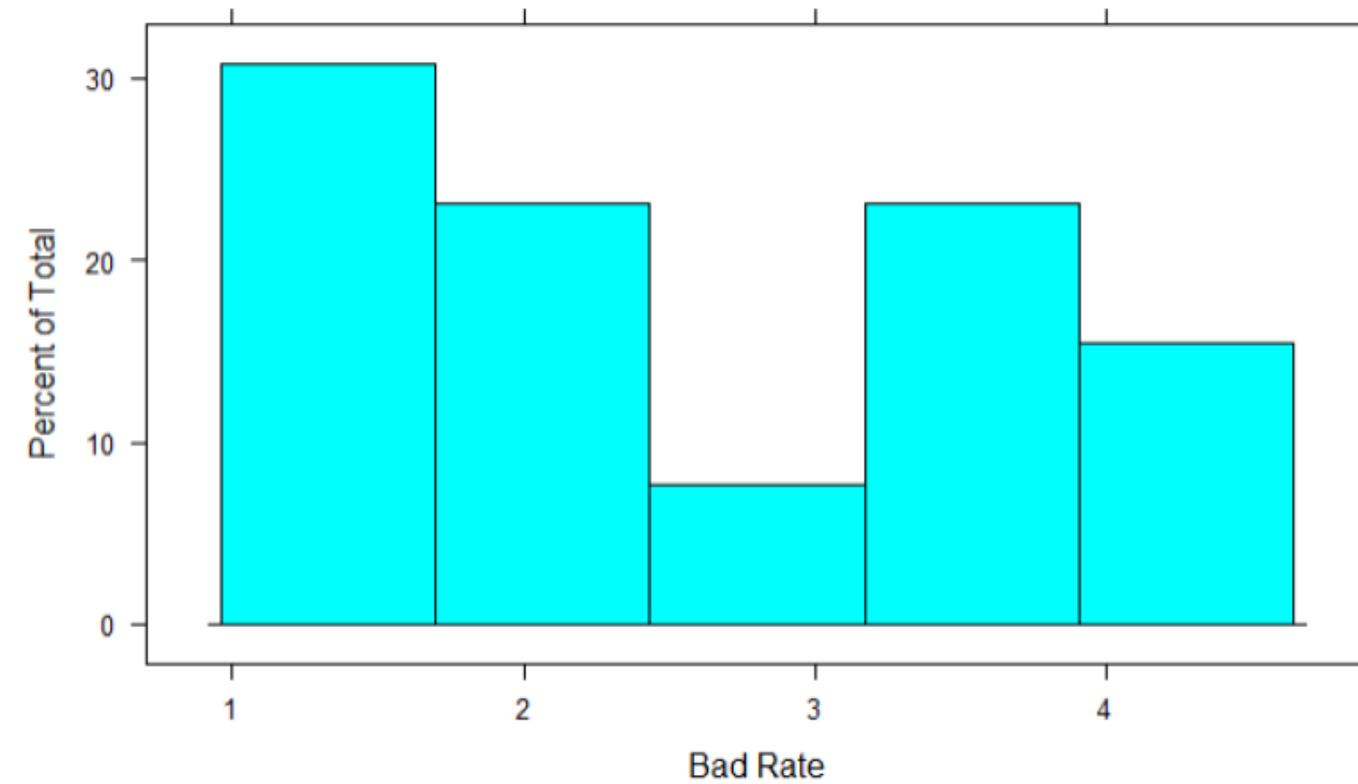
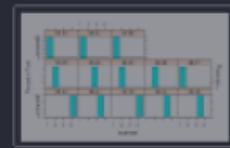
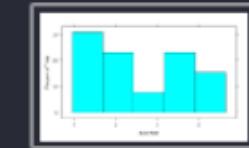
```
#Visualize the frequency of bad credit rate obtained by different age group using histograms for a given “Loan dataset” in R.
```

```
df<-read.csv("LoanData.csv")
```

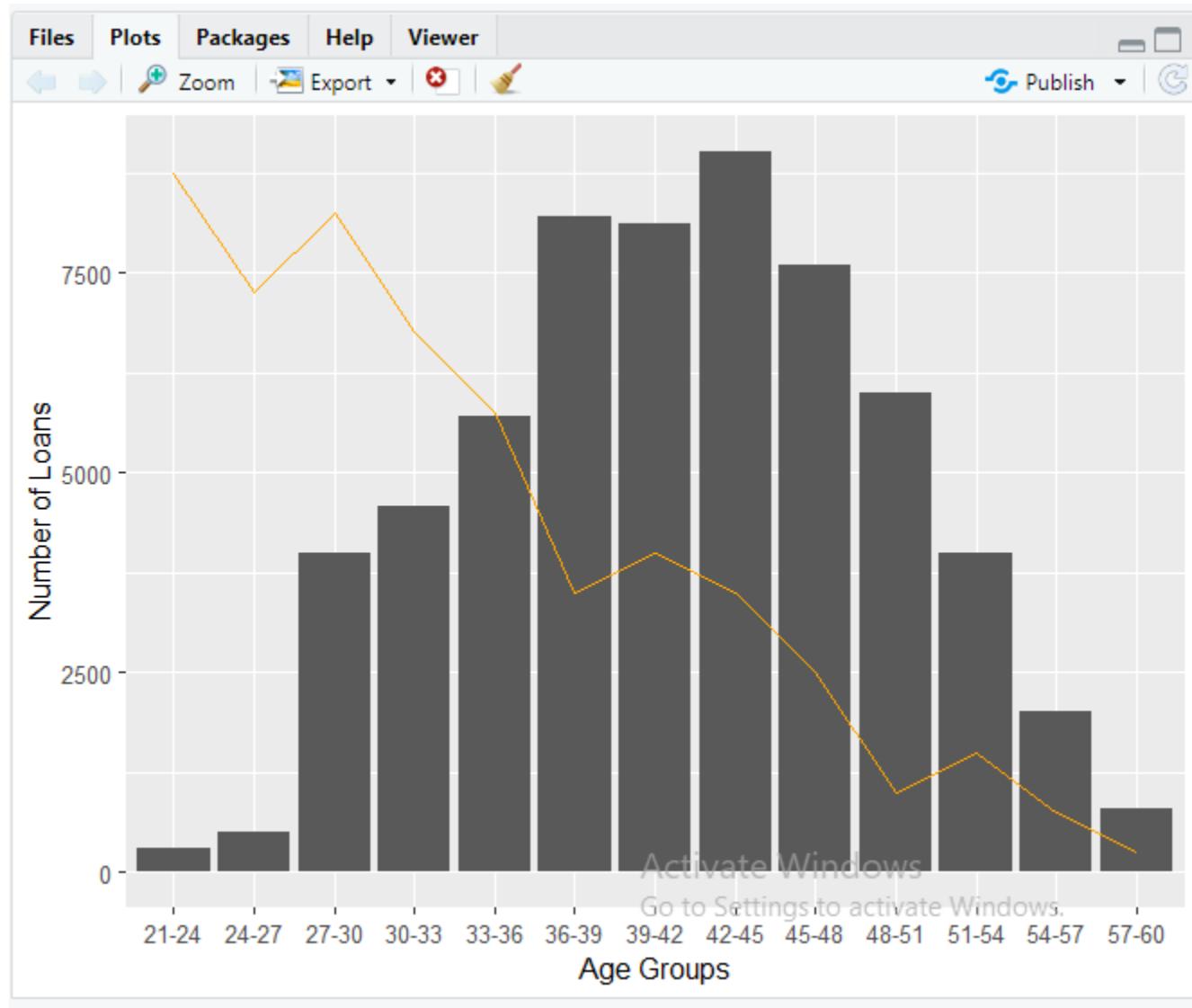
```
df
```

```
> #Visualize the frequency of bad credit rate obtained by different age group using histograms for a given "Loan dataset" in R.  
>  
> df<-read.csv("LoanData.csv")  
> df  
Age.Group Number.of.Loans Bad.Loans Good.Loans Bad.Rate  
1 21-24 310 14 296 4.5  
2 24-27 511 20 491 3.9  
3 27-30 4000 172 3828 4.3  
4 30-33 4568 169 4399 3.7  
5 33-36 5698 188 5510 3.3  
6 36-39 8209 197 8012 2.4  
7 39-42 8117 211 7906 2.6  
8 42-45 9000 216 8784 2.4  
9 45-48 7600 152 7448 2.0  
10 48-51 6000 84 5916 1.4  
11 51-54 4000 64 3936 1.6  
12 54-57 2000 26 1974 1.3  
13 57-60 788 9 779 1.1  
  
library(ggplot2)  
  
ggplot(data=df, aes(x=Age.Group, y=Bad.Rate)) +geom_bar(stat="identity") + labs(x = "Age Group", y = "Bad Credit Rate")
```

```
15 ````{r}
16 #view(df)
17 library(lattice)
18
19 df$age=factor(df$Age.Group)
20
21 histogram(~Bad.Rate, df, xlab="Bad Rate")
22
23 histogram(~Bad.Rate|age, df, xlab="Bad Rate", title="Histogram")
24 ````
```

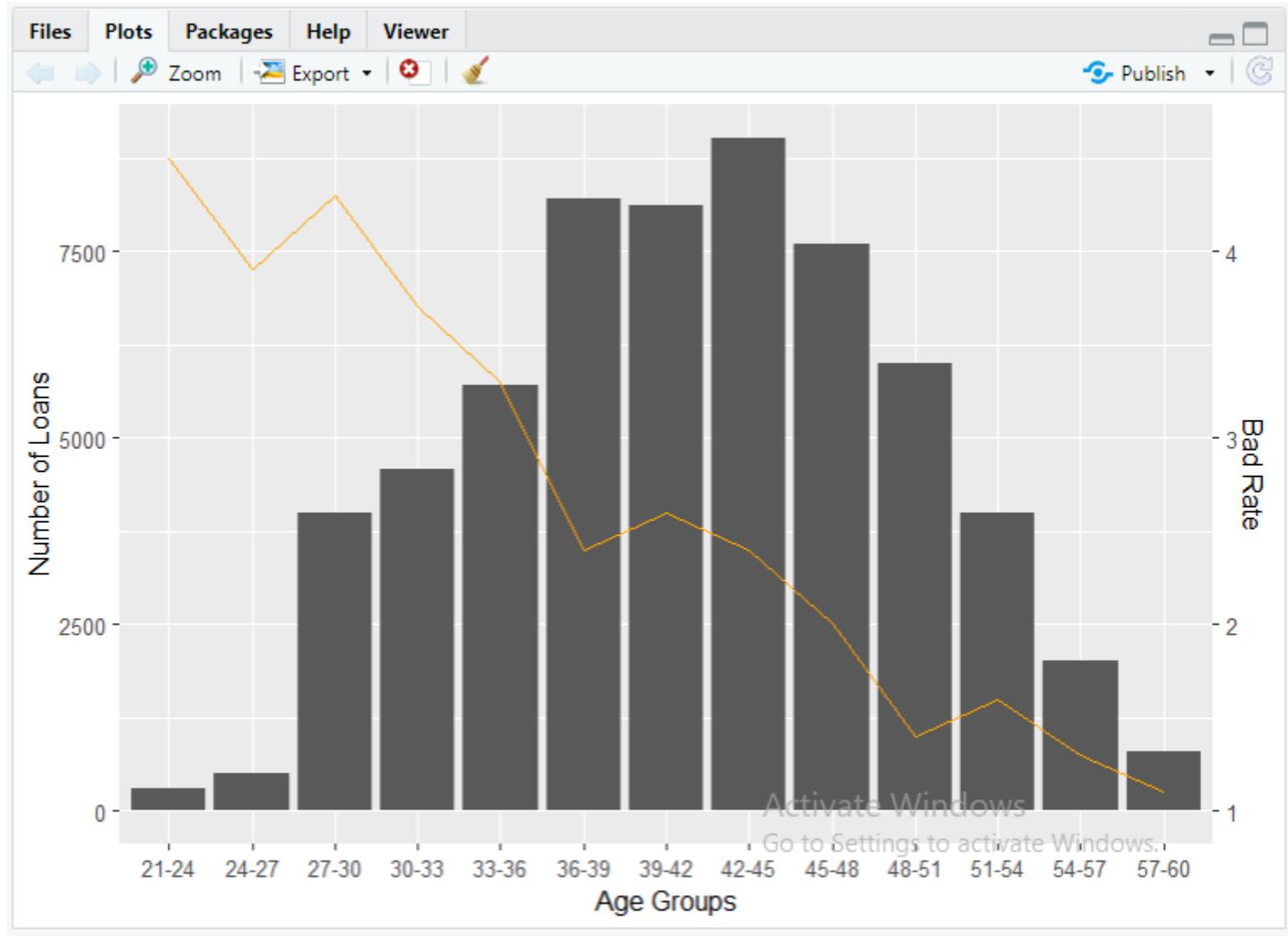


```
ggplot(df, aes(x=Age.Group))+ geom_bar(aes(y=Number.of.Loans),stat="identity") + geom_line(aes(y=(Bad.Rate-1)*2500, group = 1),stat="identity",color="orange")+
  labs(x="Age Groups",y="Number of Loans")
```

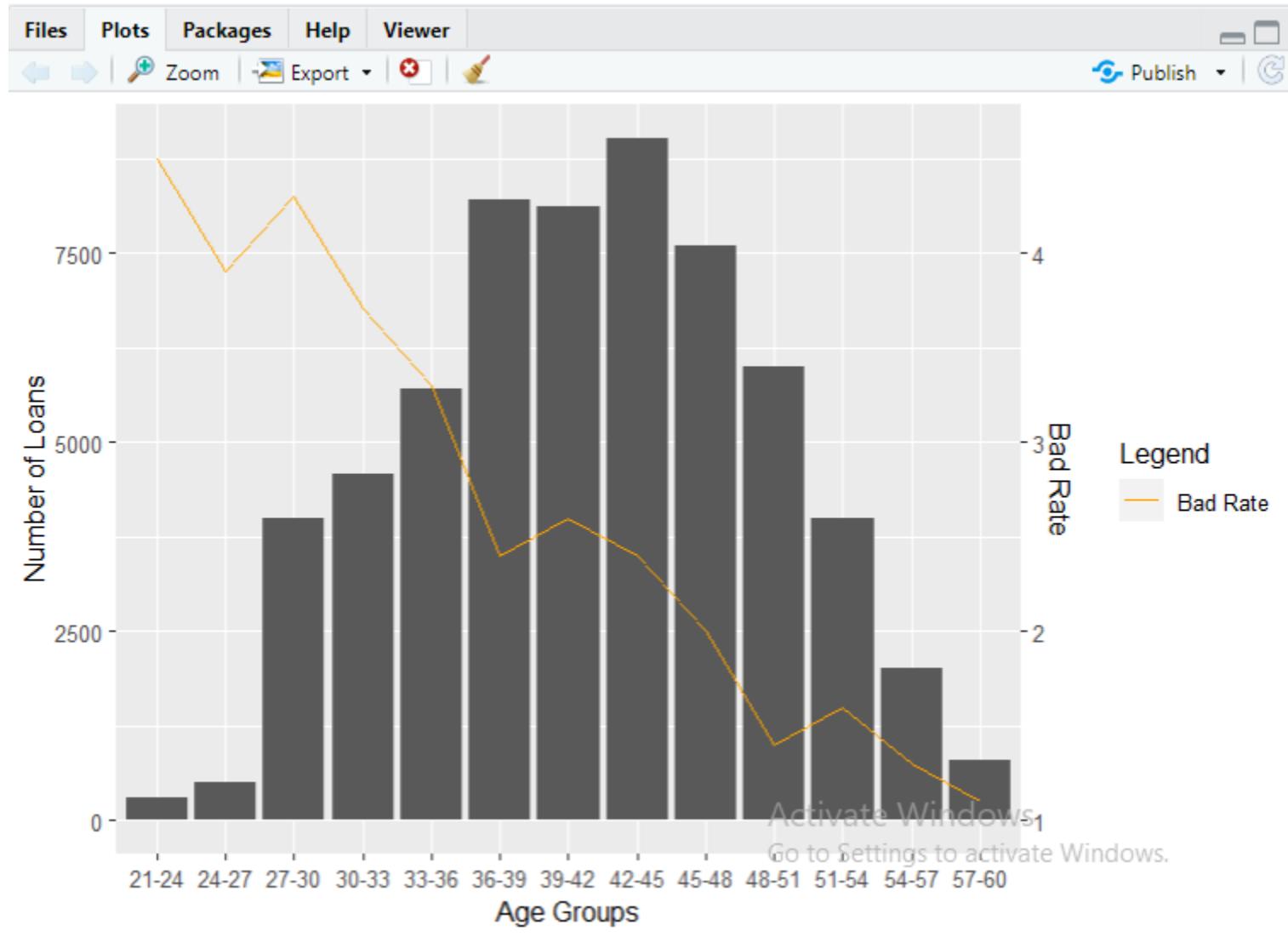


```
ggplot(df, aes(x=Age.Group))+
  geom_bar(aes(y=Number.of.Loans),stat="identity") +
  geom_line(aes(y=(Bad.Rate-1)*2500, group = 1),stat="identity", color="orange")+
  labs(x="Age Groups",y="Number of Loans")+
```

```
scale_y_continuous(sec.axis=sec_axis(~.*0.0004+1,name="Bad Rate"))
```



```
ggplot(df, aes(x=Age.Group))+  
  geom_bar(aes(y=Number.of.Loans),stat="identity") +  
  geom_line(aes(y=(Bad.Rate-1)*2500, group = 1, color = "Bad Rate"),stat="identity",  
            show.legend=TRUE)+  
  labs(x="Age Groups",y="Number of Loans") +  
  scale_color_manual(name = "Legend", values = c("Bad Rate" = "orange")) +  
  scale_y_continuous(sec.axis=sec_axis(~.*0.0004+1,name="Bad Rate"))
```



Show the different inference (with 5 different sheets) of Loan dataset using Tableau. Apply the Tableau with R integration to identify the bad loan and good loan.

Integrating R with Tableau

Console Terminal × Jobs ×

R 4.2.0 · ~/dFatLab/ ↗

```
> library(Rserve)
> Rserve()
Starting Rserve...
"C:\Users\admin\AppData\Local\R\WIN-LI~1\4.2\Rserve\libs\x64\Rserve.exe"
> |
```

Manage Analytics Extensions Connection X



Edit RServe Connection

Require SSL

Hostname Port

Sign in with username and password

Username Password

Test Connection Disconnect

Close Save

lm

X

Results are computed along Table (across).

```
SCRIPT_REAL("
y<- .arg1
x<- .arg2
fit<-lm(y~x)
fit$fitted.values
",
SUM([Bad Loans]),SUM([Good Loans])
)
```

The calculation is valid.

2 Dependencies ▾

Default Table Calculation

Apply

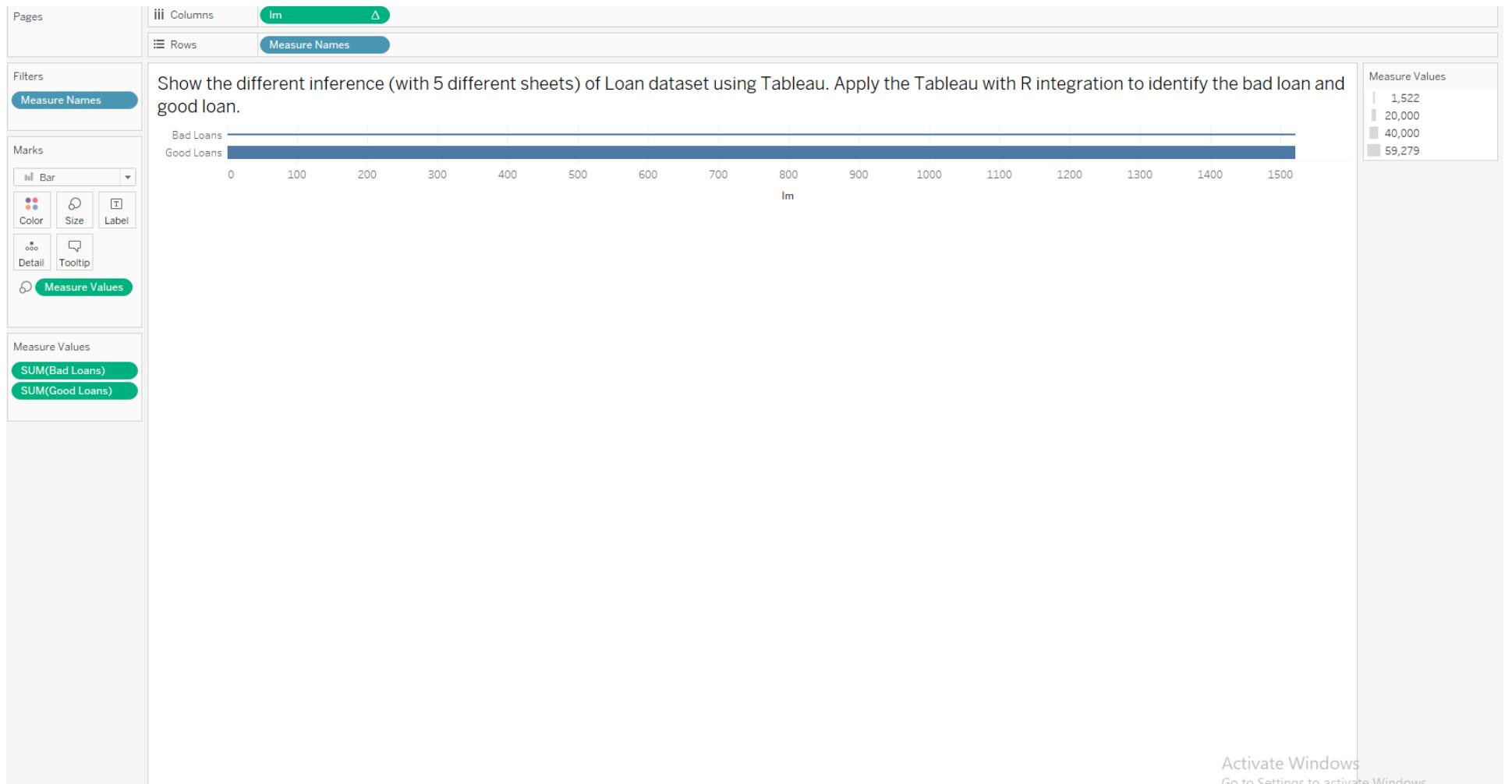
OK

All	▼
Search	
ABS	
ACOS	
AND	
AREA	
ASCII	
ASIN	
ATAN	
ATAN2	
ATTR	
AVG	
BUFFER	
CASE	
CEILING	

ABS (number)

Returns the absolute value of the given number.

Example: ABS (-7) = 7



X

All	▼
Search	
ABS	
ACOS	
AND	
AREA	
ASCII	
ASIN	
ATAN	
ATAN2	
ATTR	
Avg	
BUFFER	
CASE	
CEILING	

ABS (number)

Returns the absolute value of the given number.

Example: ABS (-7) = 7

Results are computed along Table (across).

```
SCRIPT_INT('
set.seed(42);
result <- kmeans(data.frame(.arg1,.arg2), 2);
result$cluster;',
SUM([Bad Loans]), SUM([Good Loans]))
```

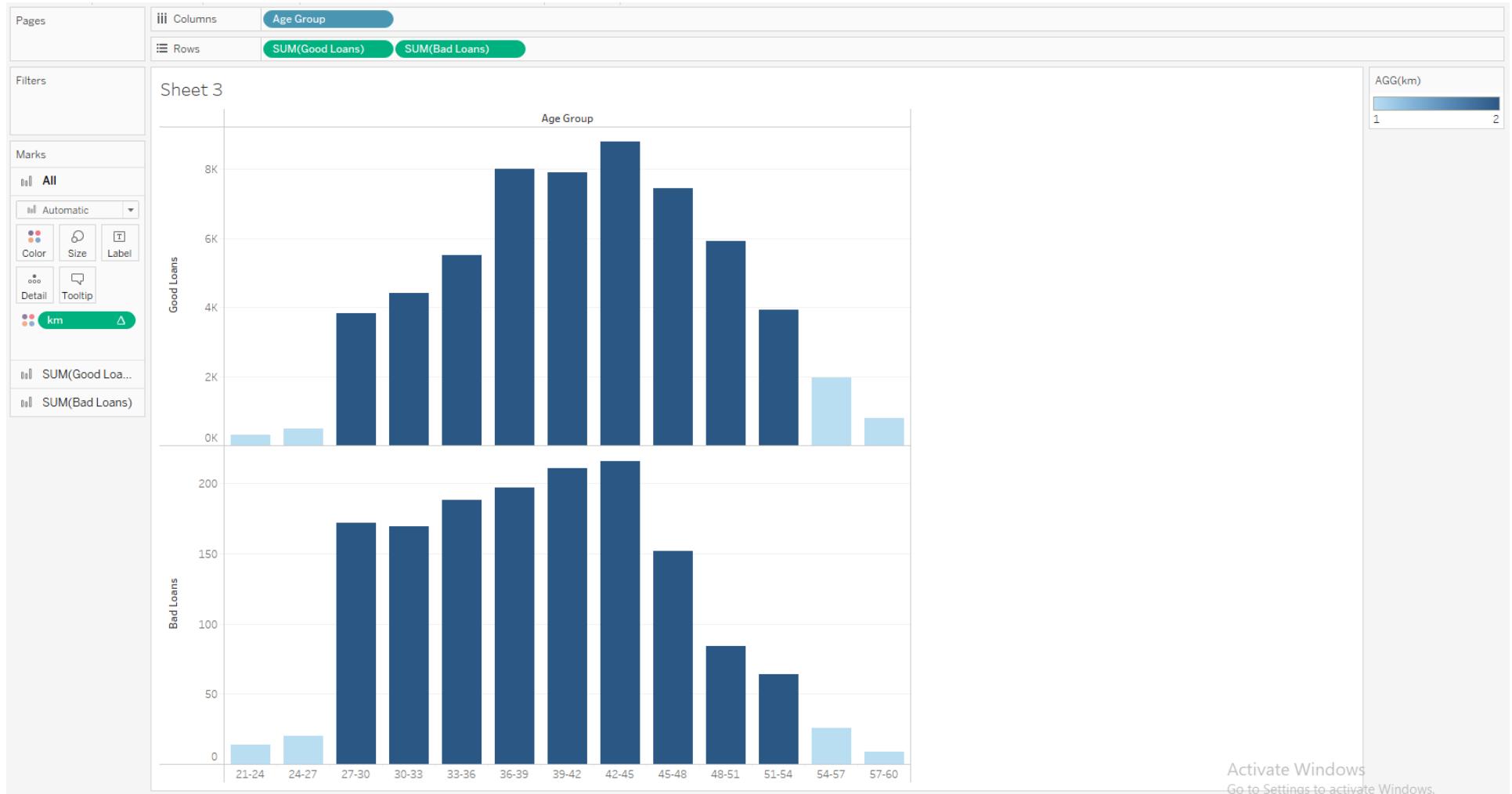
Default Table Calculation

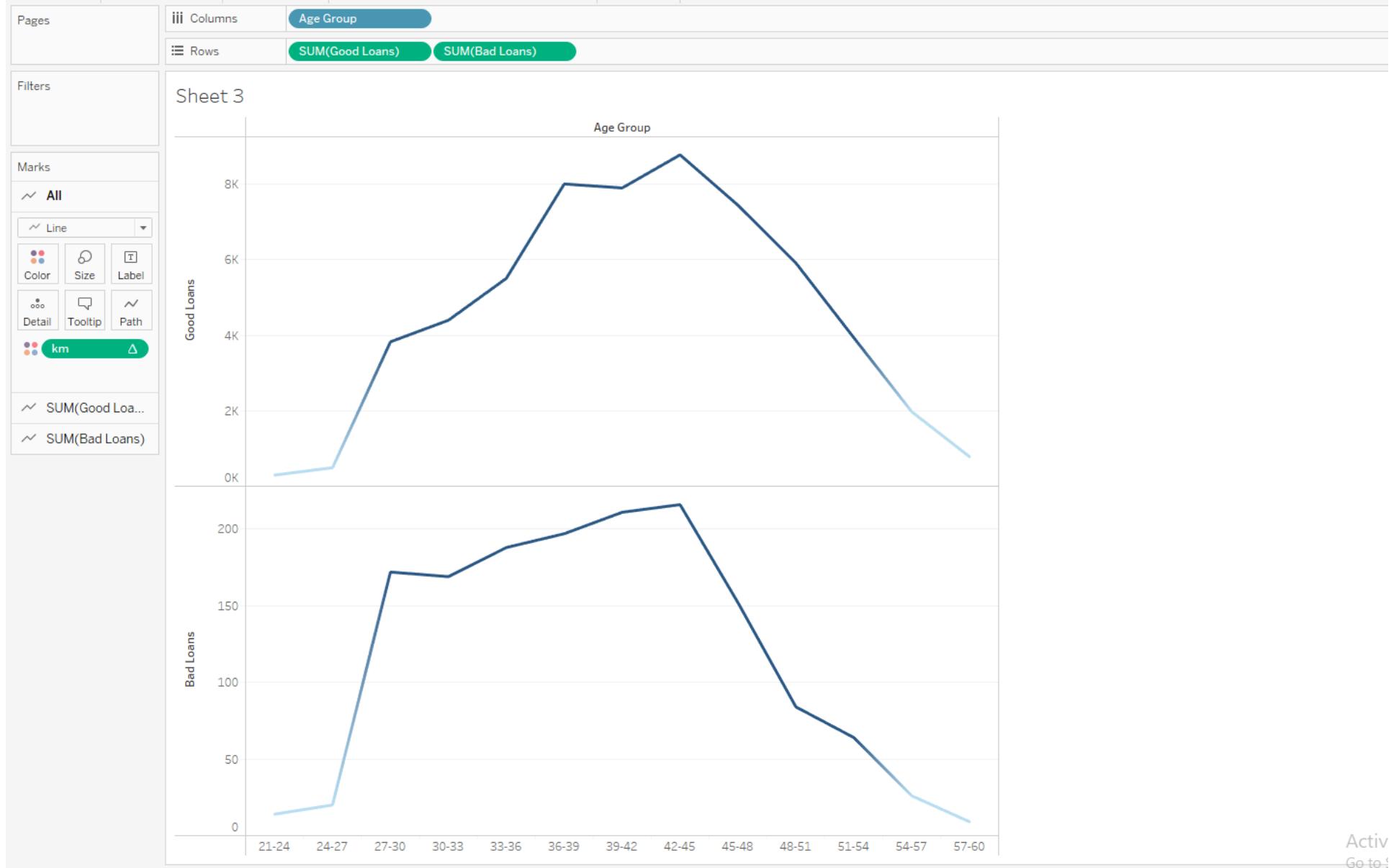
The calculation is valid.

1 Dependency ▾

Apply

OK





Activ
Go to:

kmeans

X

Results are computed along Table (across).

```
frame(.arg1,.arg2,.arg3,.arg4), 3);  
[Good Loans]),SUM([Bad Rate]),SUM([Number of Loans]))
```

The calculation is valid.

2 Dependencies ▾

Apply

OK

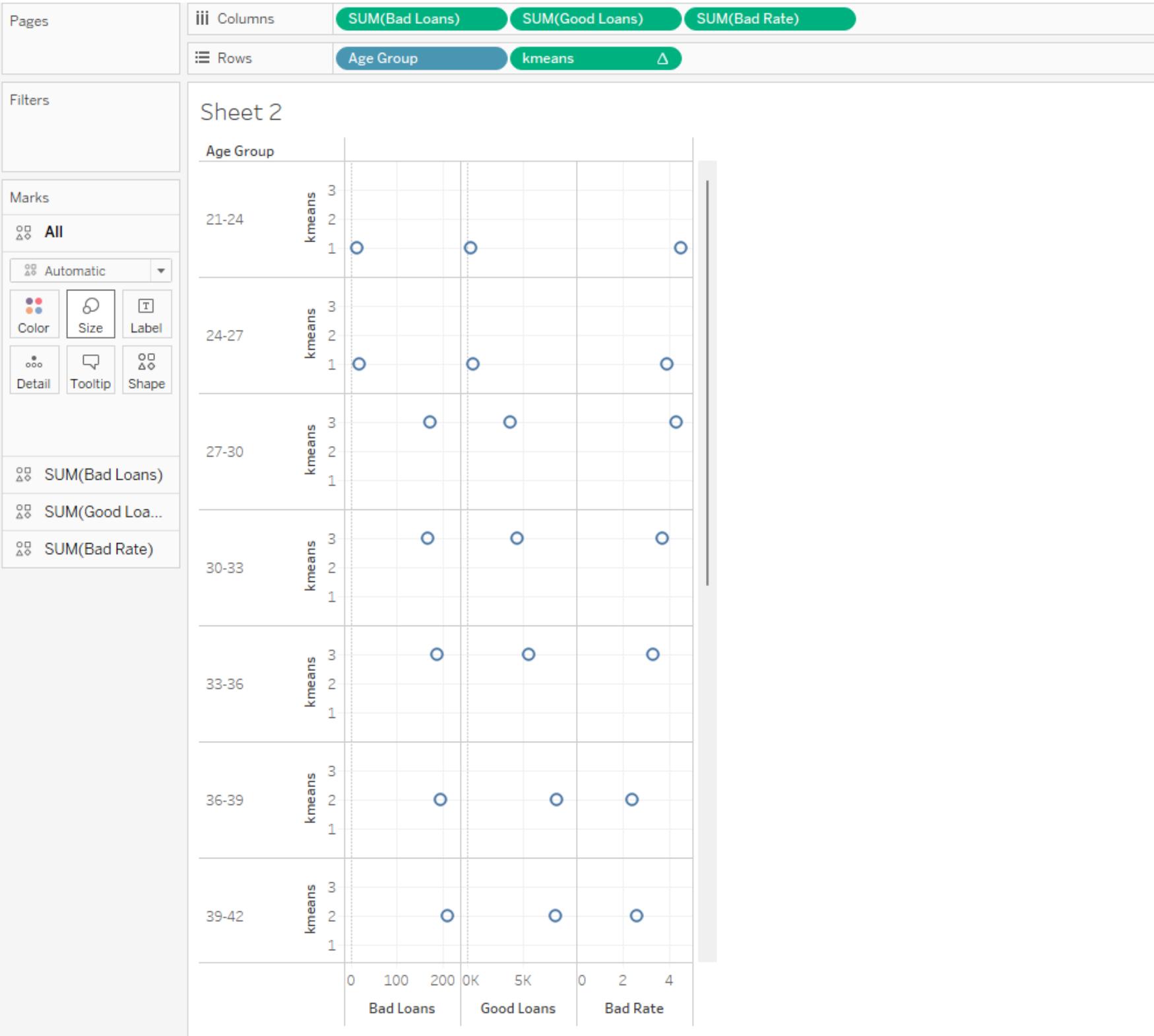
Default Table Calculation

All	▼
Search	
ABS	
ACOS	
AND	
AREA	
ASCII	
ASIN	
ATAN	
ATAN2	
ATTR	
AVG	
BUFFER	
CASE	
CEILING	

ABS (number)

Returns the absolute value of the given number.

Example: ABS (-7) = 7



Pages

iii Columns SUM(Good Loans) SUM(Bad Loans)

ii Rows Age Group SUM(Bad Rate) SUM(Number of Loa..)

Filters

Marks

All

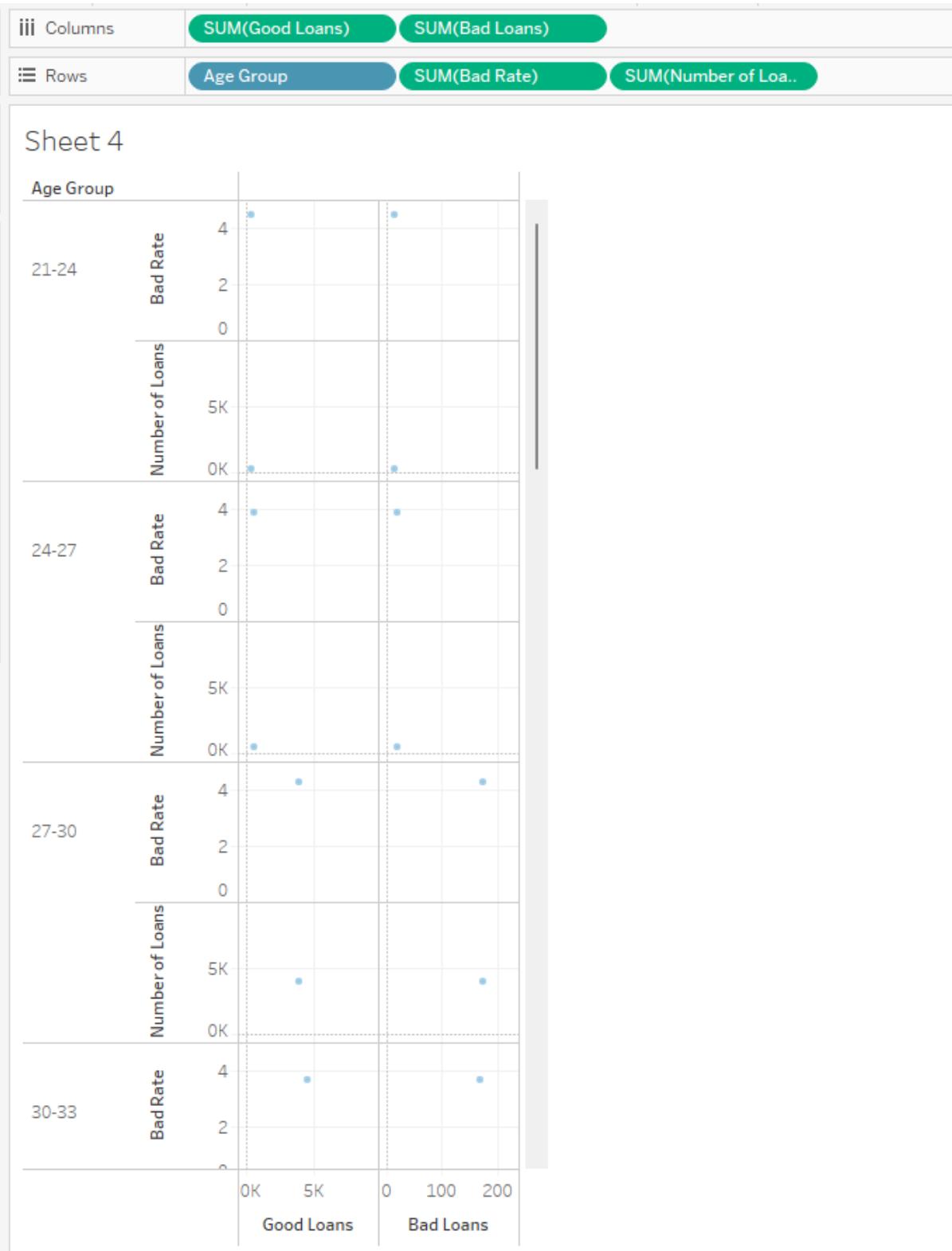
Density

Color Size Label

Detail Tooltip

Im △

SUM(Good Lo...) SUM(Bad Loans)



distinguish

Results are computed along Table (across).

```
SCRIPT_REAL("y<-arg1x1<-arg2x2<-arg2fit<-lm(y~x1+x2)fit$fitted.values",SUM([Bad Loans]),SUM([Number of Loans]),SUM([Bad Rate]))}
```

The calculation is valid.

Default Table Calculation

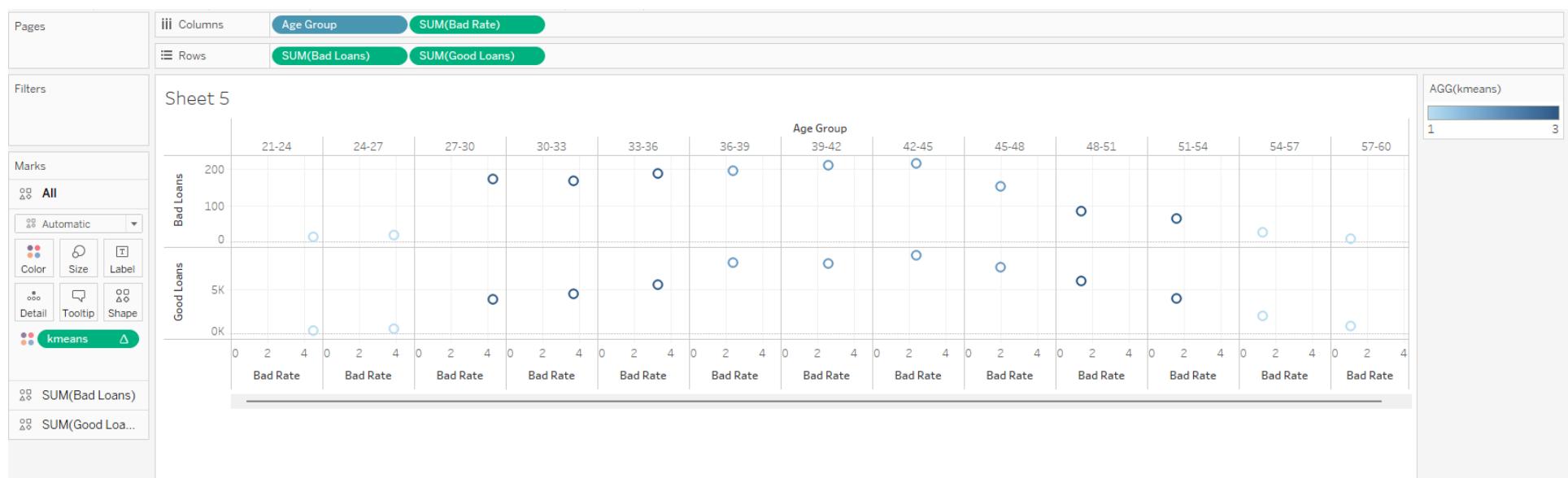
OK

All
Search
ABS
ACOS
AND
AREA
ASCII
ASIN
ATAN
ATAN2
ATTR
AVG
BUFFER
CASE
DEVMAP

ABS (number)

Returns the absolute value of the given number.

Example: $\text{ABS}(-7) = 7$





Case Study 3:

Create Time series data visualizations using suitable statistical model and explore the “Chicago_batteries” dataset using R. Create forecasting and Trend line charts that highlight something interesting inference in “Chicago_batteries dataset” using Tableau.

Rstudio CODE:

```
#Create Time series data visualizations using suitable statistical model and explore the “Chicago_batteries” dataset using R. Create forecasting and Trend line charts that highlight something interesting inference in “Chicago_batteries dataset” using Tableau.
```

```
data <- read.csv("C:/Users/admin/Downloads/chicago_batteries_2002_to_2013.csv")
library(tseries) library(forecast) library(TTR) library(dplyr) library(lattice)
xyplot(data$ozone~data$daylight_hours|factor(data$month), data, xlab="daylight_hours, ylab="ozone, group="day")
datats<-ts(data)
plot.ts(datats, main="plot of timeseries data", col="blue") na.omit(datats)
datalogts<-log(datats)
plot.ts(datalogts, main="plot of log of timeseries data", col="blue")
```

```
summary(data) str(data) library(TTR) library(dplyr) cycle
```

```
xyplot(cycle~data$month) acf(datats) pacf((datats))
```

```
train<-data[1:2500,] test<-data[2501:4379,]
```

```
fit<-arima(datats, order = c(0L, 0L, 0L), seasonal = list(order = c(0L, 0L, 0L), period = NA), xreg = NULL, include.mean = TRUE, transform.pars = TRUE, fixed = NULL, init = NULL)
```

```
pred<-predict(fit,train) plot(pred
```

```
plot(train)
```

Exploring the data:

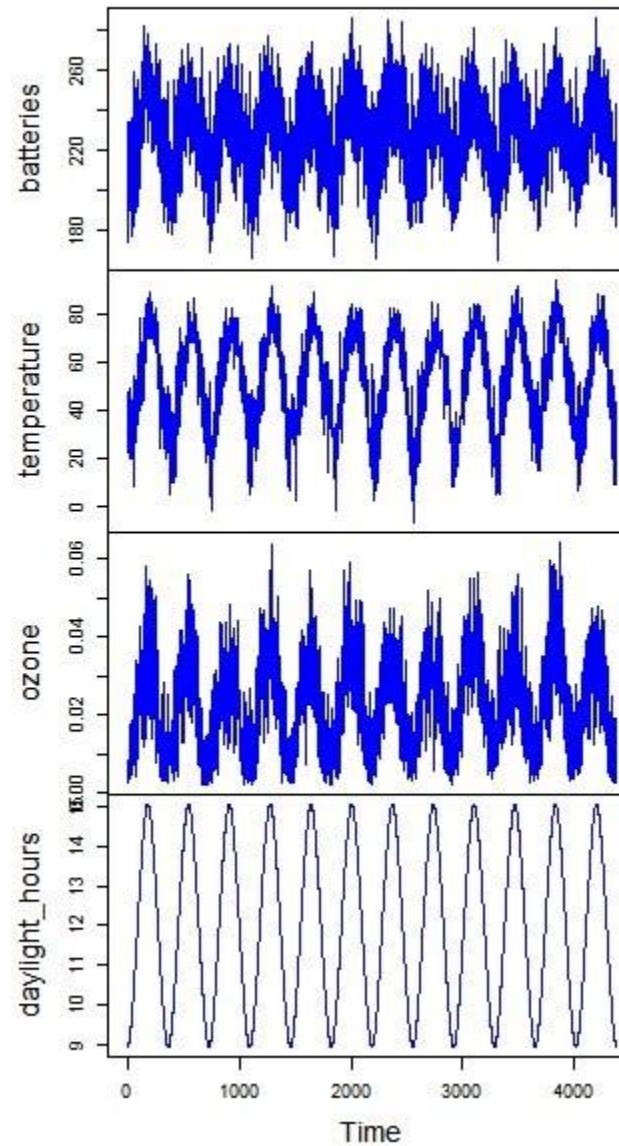
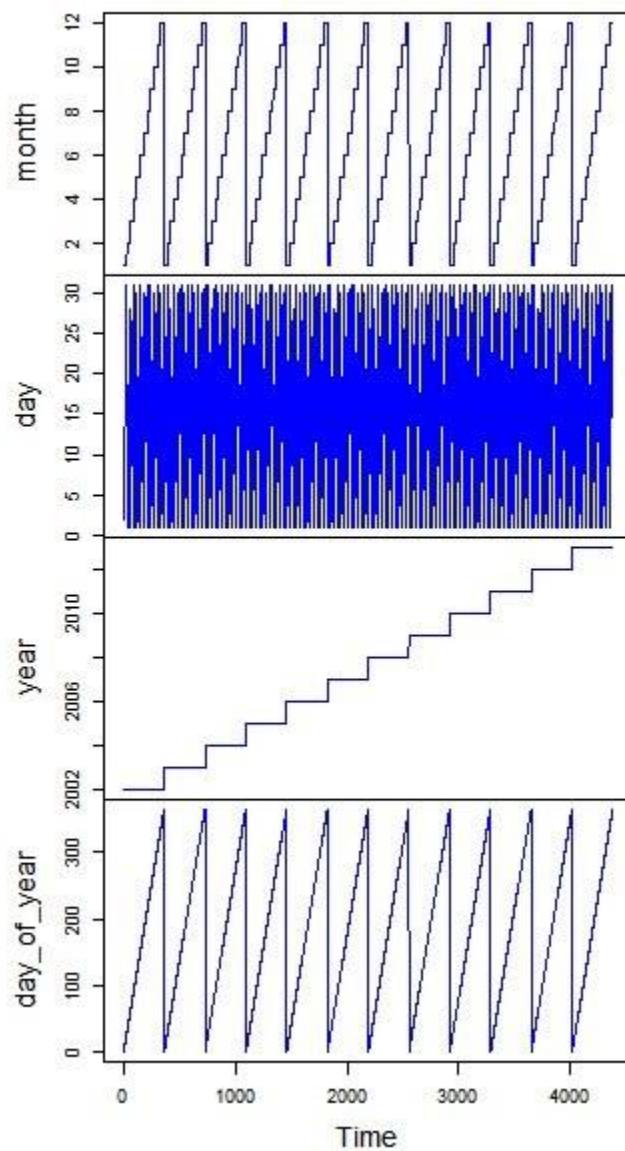
```
> head(datats)
Time Series:
Start = 1
End = 6
Frequency = 1
  month day year day_of_year batteries temperature      ozone daylight_hours
1     1   2 2002           2  173.8245        24 0.0027920    8.955492
2     1   3 2002           3  183.0151        22 0.0039580    8.958159
3     1   4 2002           4  207.3607        29 0.0067705    8.961890
4     1   5 2002           5  192.1771        32 0.0031670    8.966683
5     1   6 2002           6  185.6178        32 0.0083335    8.972532
6     1   7 2002           7  195.6608        22 0.0044585    8.979435
> tail(datats)
Time Series:
Start = 4374
End = 4379
Frequency = 1
  month day year day_of_year batteries temperature      ozone daylight_hours
4374  12  26 2013         360  191.4680        26 0.016917    8.968046
4375  12  27 2013         361  200.6741        35 0.014833    8.962989
4376  12  28 2013         362  220.9803        43 0.018958    8.958992
4377  12  29 2013         363  217.4040        26 0.019208    8.956059
4378  12  30 2013         364  208.0243        9 0.010708    8.954191
4379  12  31 2013         365  243.0088        9 0.010316    8.953390
```

```
> data <- read.csv("C:/Users/admin/Downloads/chicago_batteries_2002_to_2013.csv")
> datats<-ts(data)
> datats
Time Series:
Start = 1
End = 4379
Frequency = 1
   month day year day_of_year batteries temperature      ozone daylight_hours
1       1   2 2002          2    173.8245        24 0.002792000 8.955492
2       1   3 2002          3    183.0151        22 0.003958000 8.958159
3       1   4 2002          4    207.3607        29 0.006770500 8.961890
4       1   5 2002          5    192.1771        32 0.003167000 8.966683
5       1   6 2002          6    185.6178        32 0.008333500 8.972532
6       1   7 2002          7    195.6608        22 0.004458500 8.979435
7       1   8 2002          8    201.3720        32 0.004292000 8.987385
8       1   9 2002          9    226.6568        47 0.002666500 8.996376
9       1  10 2002         10    232.2680        41 0.002854000 9.006401
10      1  11 2002         11    233.8388        38 0.006333500 9.017453
11      1  12 2002         12    207.3651        37 0.005458000 9.029524
12      1  13 2002         13    199.0590        34 0.005645500 9.042603
13      1  14 2002         14    204.2526        37 0.003354500 9.056682
14      1  15 2002         15    223.4759        32 0.005687500 9.071751
15      1  16 2002         16    207.4463        32 0.004375000 9.087797
16      1  17 2002         17    198.1164        21 0.005000000 9.104809
17      1  18 2002         18    227.5494        20 0.004916500 9.122776
18      1  19 2002         19    213.2389        24 0.007729500 9.141684
19      1  20 2002         20    192.2885        30 0.010937500 9.161520
20      1  21 2002         21    197.2001        32 0.006312500 9.182270
21      1  22 2002         22    221.8237        44 0.011479000 9.203920
22      1  23 2002         23    211.9438        44 0.008937500 9.226456
```

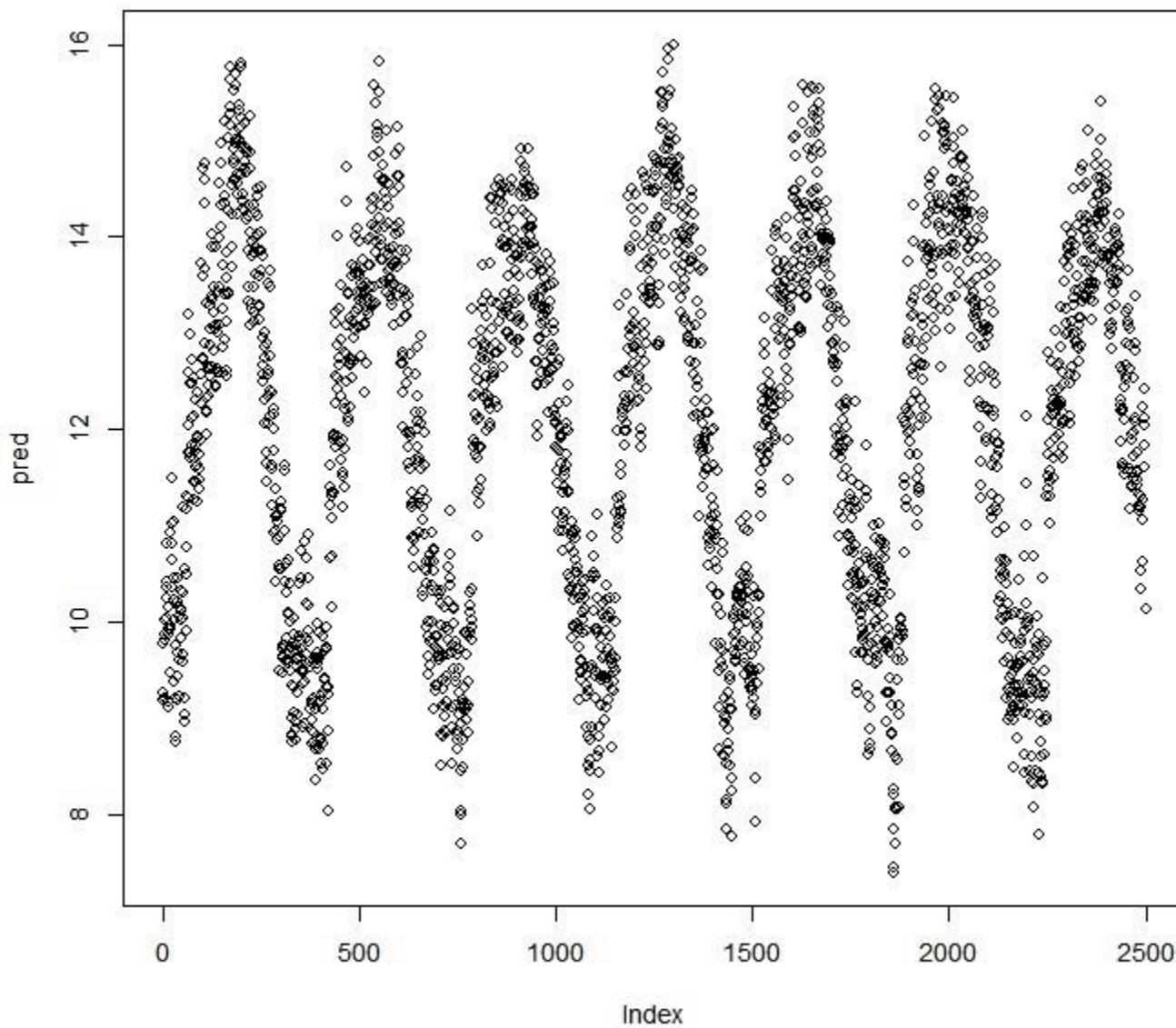
```
datats<-ts(data)
```

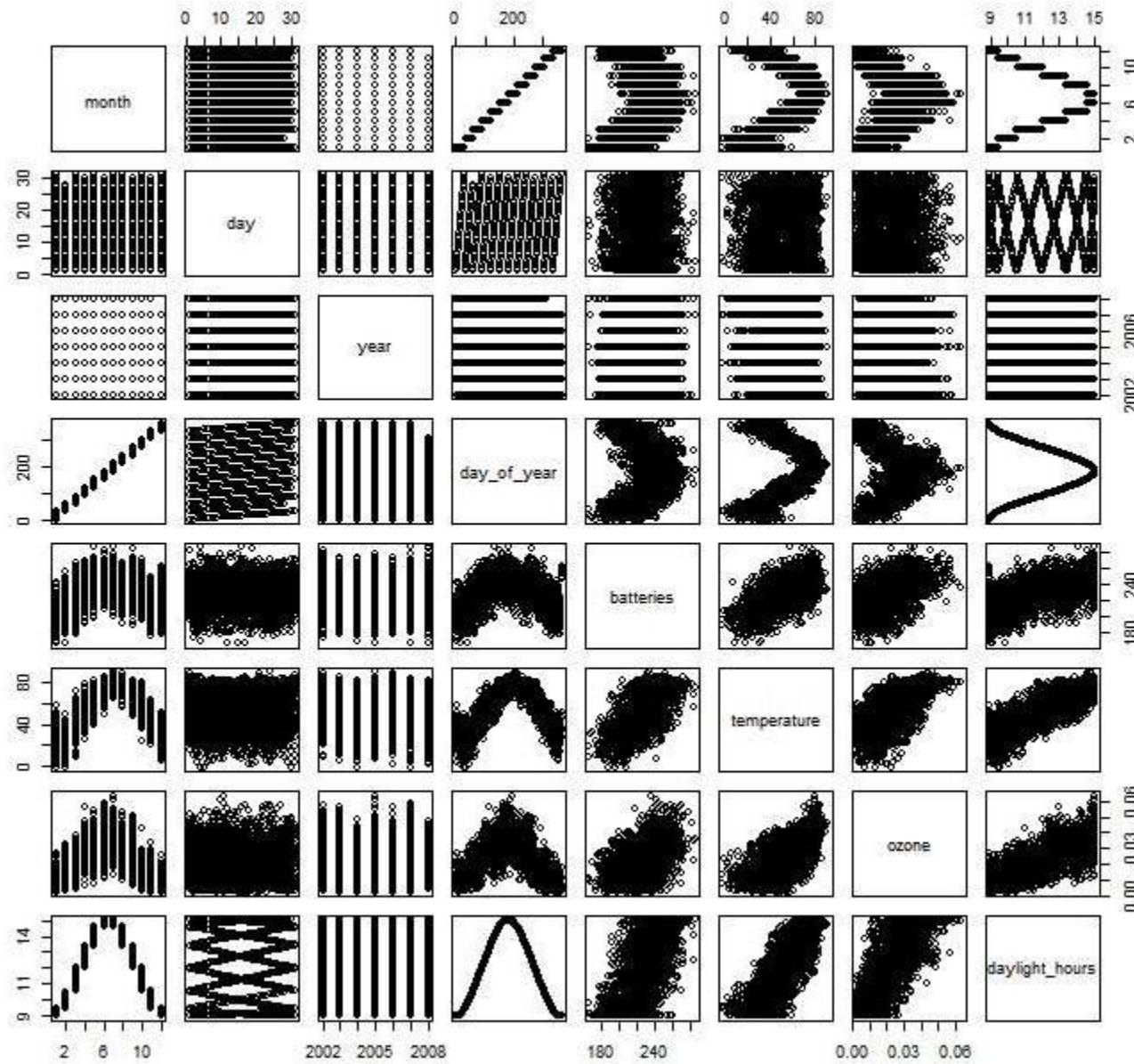
```
plot.ts(datats, main="plot of timeseries data", col="blue")
```

plot of timeseries data



`pred<-predict(fit,train) plot(pred)`





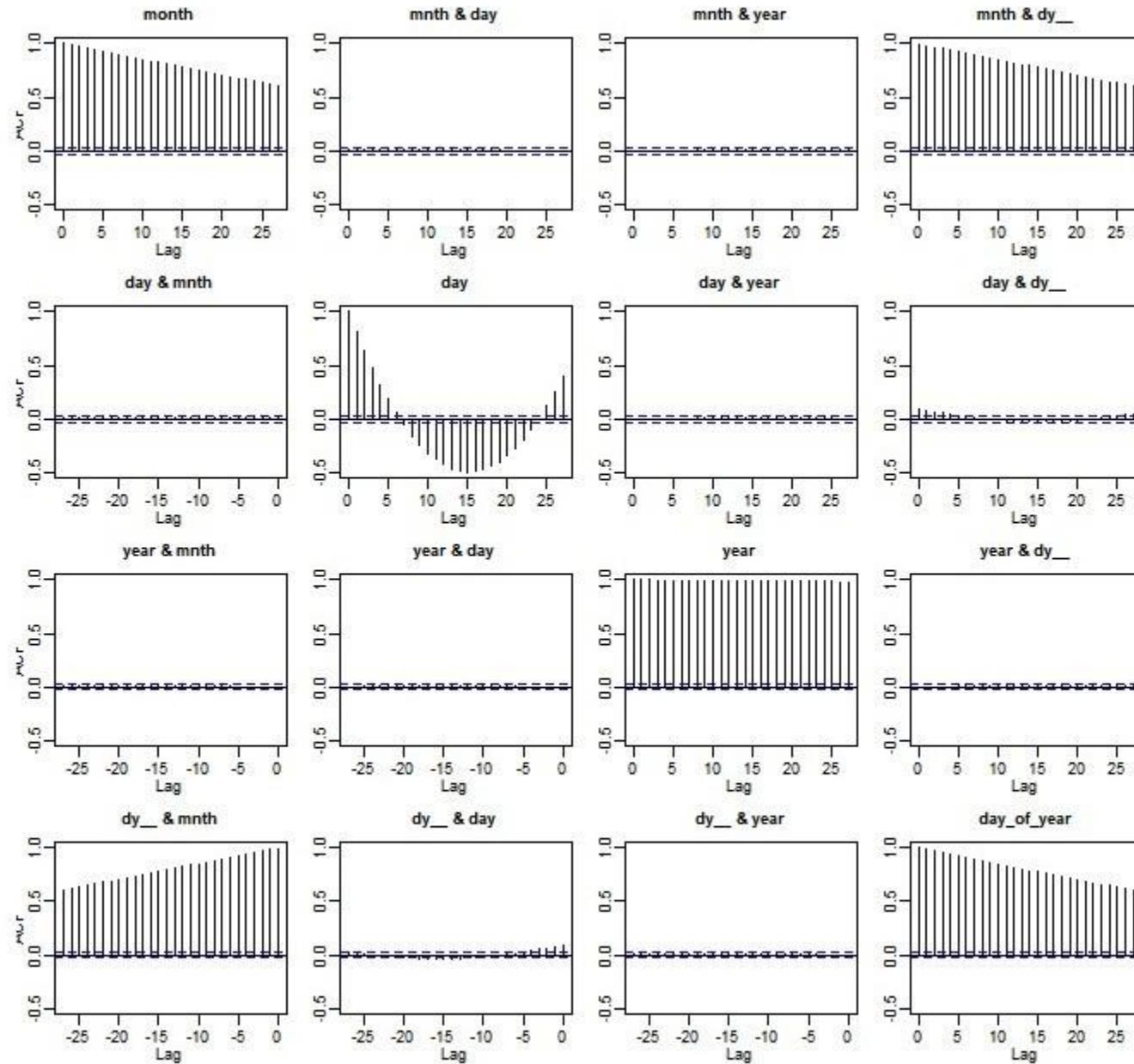
```

> pred<-predict(fit,train)
> pred
      1       2       3       4       5       6       7       8       9       10      11      12      13      14      15
9.274140 9.192822 9.774430 9.822346 10.013658 9.216642 9.868566 10.806028 10.422846 10.358885 10.240828 10.044681 10.156595 9.930026 9.868032
      16      17      18      19      20      21      22      23      24      25      26      27      28      29      30
9.163586 9.114114 9.472580 9.972596 9.929877 10.929007 10.824424 10.366912 10.649002 11.015051 11.486990 11.050483 10.453617 10.238972 10.020295
      31      32      33      34      35      36      37      38      39      40      41      42      43      44      45
9.382174 9.219109 9.754457 8.765845 8.817209 9.184579 9.435596 9.674364 10.451599 10.217690 9.600454 10.148229 9.226463 10.058352 10.022709
      46      47      48      49      50      51      52      53      54      55      56      57      58      59      60
10.136636 9.837581 10.110793 10.325767 10.305473 9.591912 9.661217 10.035720 11.189454 10.548329 9.213819 8.960510 9.038987 11.311263 11.182533
      61      62      63      64      65      66      67      68      69      70      71      72      73      74      75
10.503428 9.917122 10.771001 12.042060 11.752175 13.190976 12.584069 11.286795 11.696955 11.757955 12.476671 12.989563 12.720637 12.483676 11.717939
      76      77      78      79      80      81      82      83      84      85      86      87      88      89      90
12.127392 11.830371 12.132520 11.451537 11.263953 12.530198 12.310328 11.599547 11.455632 11.786115 11.838999 11.908059 12.381274 11.930798 11.259145
      91      92      93      94      95      96      97      98      99      100     101     102     103     104     105
11.626097 11.593066 11.467839 11.383275 11.739093 11.895864 12.264256 11.888466 12.575909 13.731436 12.720254 12.743192 13.596687 14.697816 14.757564
      106     107     108     109     110     111     112     113     114     115     116     117     118     119     120
14.353625 14.600296 13.666901 12.894177 12.181086 11.954097 12.321863 13.316933 12.198754 12.465372 12.349696 12.632690 12.653452 12.629305 13.046310
      121     122     123     124     125     126     127     128     129     130     131     132     133     134     135
12.871226 12.547906 13.296188 14.028058 13.895252 13.38027 12.651522 13.354593 13.484961 13.218331 13.054244 12.816566 13.037258 13.896040 13.086853
      136     137     138     139     140     141     142     143     144     145     146     147     148     149     150
12.465398 12.444007 13.011193 12.627412 12.588834 13.425257 13.986055 13.081985 12.581544 13.476552 14.553310 14.098431 14.334849 14.760282 14.758568
      151     152     153     154     155     156     157     158     159     160     161     162     163     164     165
14.972773 13.708498 13.248082 12.852536 12.595001 12.929122 13.400595 14.421369 15.195637 14.346543 13.839471 13.114328 12.545998 12.618880 13.568479
      166     167     168     169     170     171     172     173     174     175     176     177     178     179     180
13.422491 13.401978 14.263587 15.034034 15.274549 15.156682 15.356959 15.757114 15.641214 14.237613 14.528607 13.897721 13.688276 14.591629 14.923448
      181     182     183     184     185     186     187     188     189     190     191     192     193     194     195
14.771240 14.802053 15.511489 15.007350 14.598311 14.979907 15.583382 15.693318 14.555714 14.433663 14.242471 14.658310 14.996531 15.016193 15.112425
      196     197     198     199     200     201     202     203     204     205     206     207     208     209     210
15.368355 15.313239 15.759512 14.651887 15.232001 15.791858 15.144159 14.358948 14.449684 14.293420 14.912012 14.790768 14.700406 14.270440 14.926206
      211     212     213     214     215     216     217     218     219     220     221     222     223     224     225
15.184176 14.908958 14.183931 14.665875 14.878541 14.335163 13.333261 13.475385 13.865477 14.221916 14.878511 15.258572 14.715164 14.053238 13.076176
      226     227     228     229     230     231     232     233     234     235     236     237     238     239     240
13.803199 13.251805 13.954253 13.302651 13.220789 13.115790 14.319177 13.831067 13.487214 13.602297 13.996238 13.890838 14.255725 13.892349 14.223501
      241     242     243     244     245     246     247     248     249     250     251     252     253     254     255
14.507210 14.421560 14.023680 13.865234 13.142147 13.261378 13.292177 13.596234 14.049984 14.519094 13.858292 13.651621 12.668482 12.489768 12.948519
      256     257     258     259     260     261     262     263     264     265     266     267     268     269     270
13.805209 12.570855 12.311908 12.857384 13.001698 12.748273 12.054842 12.900350 11.644247 11.760168 11.455165 12.056293 12.605671 12.336607 12.567101
      271     272     273     274     275     276     277     278     279     280     281     282     283     284     285
13.515477 13.654247 13.483119 12.766495 12.145753 12.403722 11.631063 12.374781 11.203036 11.598129 11.781477 11.996915 12.205570 12.149617 10.921693
      286     287     288     289     290     291     292     293     294     295     296     297     298     299     300
10.954497 11.373130 11.103758 10.425984 11.126173 11.042545 10.858681 11.095721 10.854631 10.898472 9.494783 10.591074 10.556236 10.898773 11.249101
      301     302     303     304     305     306     307     308     309     310     311     312     313     314     315
11.145912 11.177858 10.555919 9.472880 9.757977 9.645526 9.720057 9.629614 9.754294 10.424900 11.576778 11.630631 10.951410 9.599590 9.828477
      316     317     318     319     320     321     322     323     324     325     326     327     328     329     330
10.308459 10.610895 10.644558 10.076168 9.602924 9.565005 10.097714 10.392474 10.002174 9.494008 9.730641 9.344978 8.826537 8.807267 8.762881
      331     332     333     334     335     336     337     338     339     340     341     342     343     344     345
8.994128 10.044436 9.695261 9.330726 9.874402 9.067225 8.958395 8.881358 8.766021 9.749546 9.264294 9.033783 9.631044 9.692938 9.601453
      346     347     348     349     350     351     352     353     354     355     356     357     358     359     360
9.847061 9.639705 10.401440 9.793563 9.488336 10.740200 10.459924 9.815932 9.713708 9.511913 9.356760 9.479989 9.500085 8.904043 8.943077
      361     362     363     364     365     366     367     368     369     370     371     372     373     374     375
9.370850 9.593795 10.828393 9.836587 10.662556 10.188704 9.659087 9.569363 9.795124 9.901318 10.178996 10.904542 10.465803 9.579755 9.099351
      376     377     378     379     380     381     382     383     384     385     386     387     388     389     390
9.532371 9.536521 8.873893 8.733526 9.186072 9.144375 8.983213 9.212485 9.217149 8.851653 8.763031 8.351221 8.715628 9.622038 9.103833
      391     392     393     394     395     396     397     398     399     400     401     402     403     404     405
8.676343 9.510602 9.622816 9.687278 9.990770 9.569962 9.623180 9.884821 9.146642 8.687587 9.235030 8.528634 9.174512 8.779333 8.785921
      406     407     408     409     410     411     412     413     414     415     416     417     418     419     420
9.094224 8.473614 8.743453 9.273527 9.849498 9.695603 9.406582 9.393950 9.414559 9.938145 9.728115 9.738143 9.320007 8.525846 8.040771

```

#Plotting values of dataset

acf(datats)



[1,1]

Chapter 5

Tableau Visualization

TABLEAU

Calculation1 X

Results are computed along Table (across).

```
SCRIPT_INT(
'set.seed(42);
train<-data[1:2500,]
test<-data[2501:4379,]
fit<-lm(_.arg1~_.arg2)

fit
',
SUM([Daylight Hours]), SUM([Ozone]))
```

▶

The calculation is valid.

Default Table Calculation

Create forecasting and Trend line charts that highlight something interesting inference in “Chicago_batteries dataset” using Tableau.

r-tableau integration:

Manage Analytics Extensions Connection

X



Action Completed

X



Successfully connected to the analytics extension.

OK

Sign in with username and password

Username

Password

Calculation2

X

Results are computed along Table (across).

```
SCRIPT_INT(
'set.seed(42);
train<-data[1:2500,]
test<-data[2501:4379,]
fit<-arima(datats, order = c(0L, 0L, 0L),
           seasonal = list(order = c(0L, 0L, 0L), period = NA),
           xreg = NULL, include.mean = TRUE,
           transform.pars = TRUE,
           fixed = NULL, init = NULL)

fit
',
SUM([Daylight Hours]), SUM([Ozone]))
```



[Default Table Calculation](#)

Apply

OK

The calculation is valid.

Fig: Highlighting trend for Ozone for each year from 2003 to 2013 and forecasting the prediction for 2014



Inference: For each year, the level of ozone rises by a factor of 0.173016 with a pi value of 0.719736 and R-squared value of 0.0009635. The forecast prediction of ozone for the year 2014 is 9.497, an increase from the year 2013. There is a linear increase in ozone from 2003.

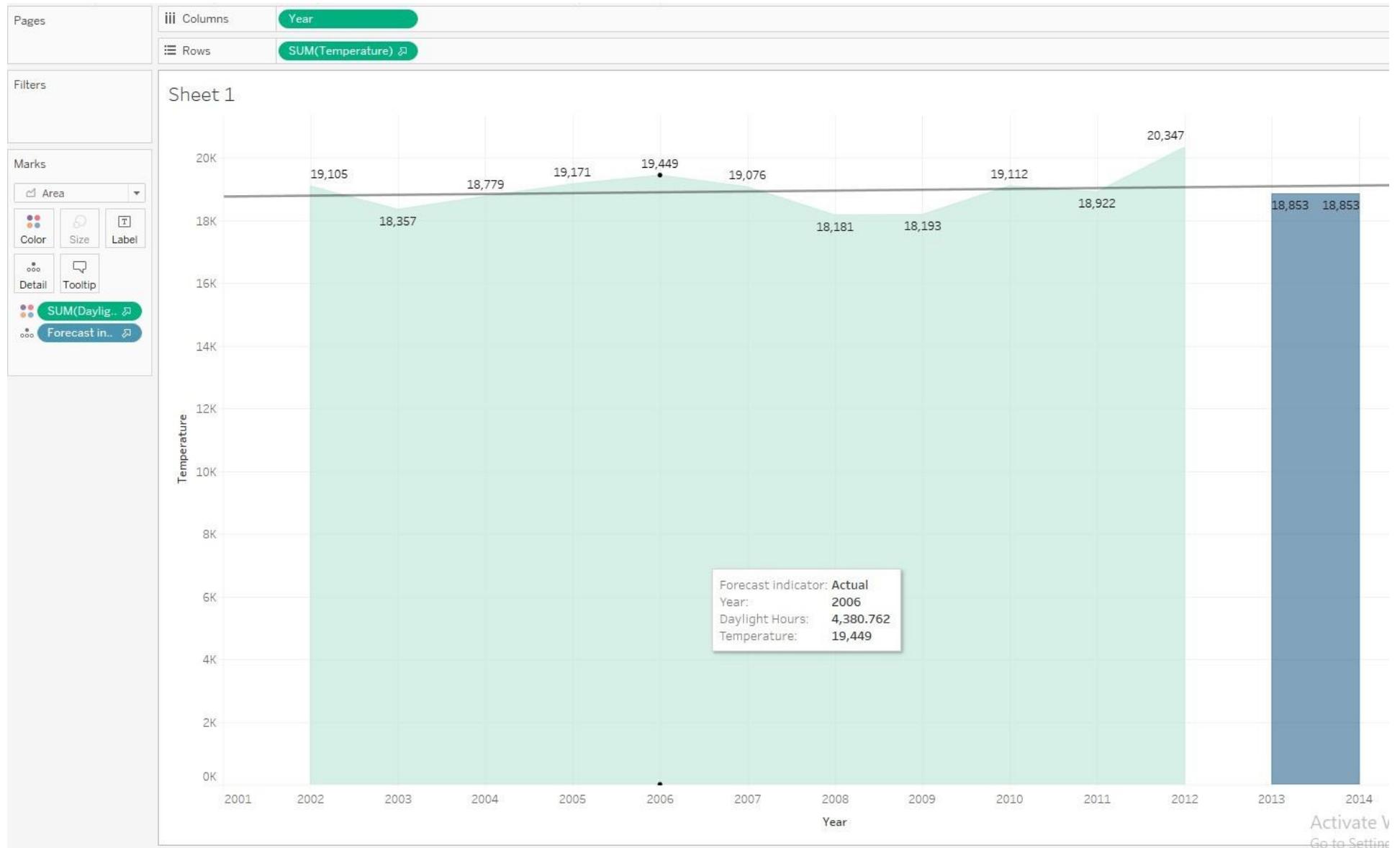
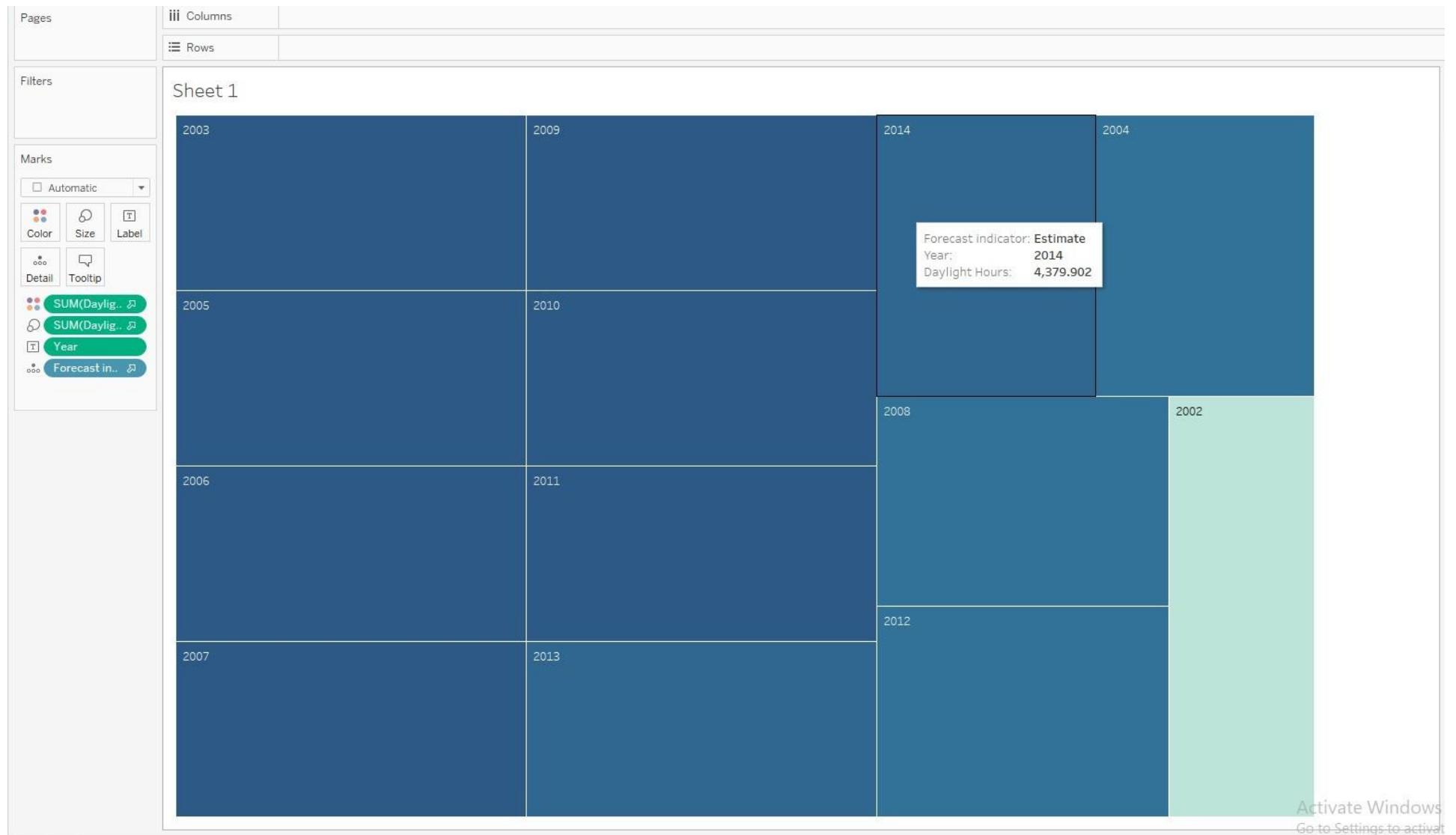


Fig: Highlighting trend for Temperature for each year from 2003 to 2013 and forecasting the prediction for 2014. Daylight hour is also used as a deciding factor for comparison.

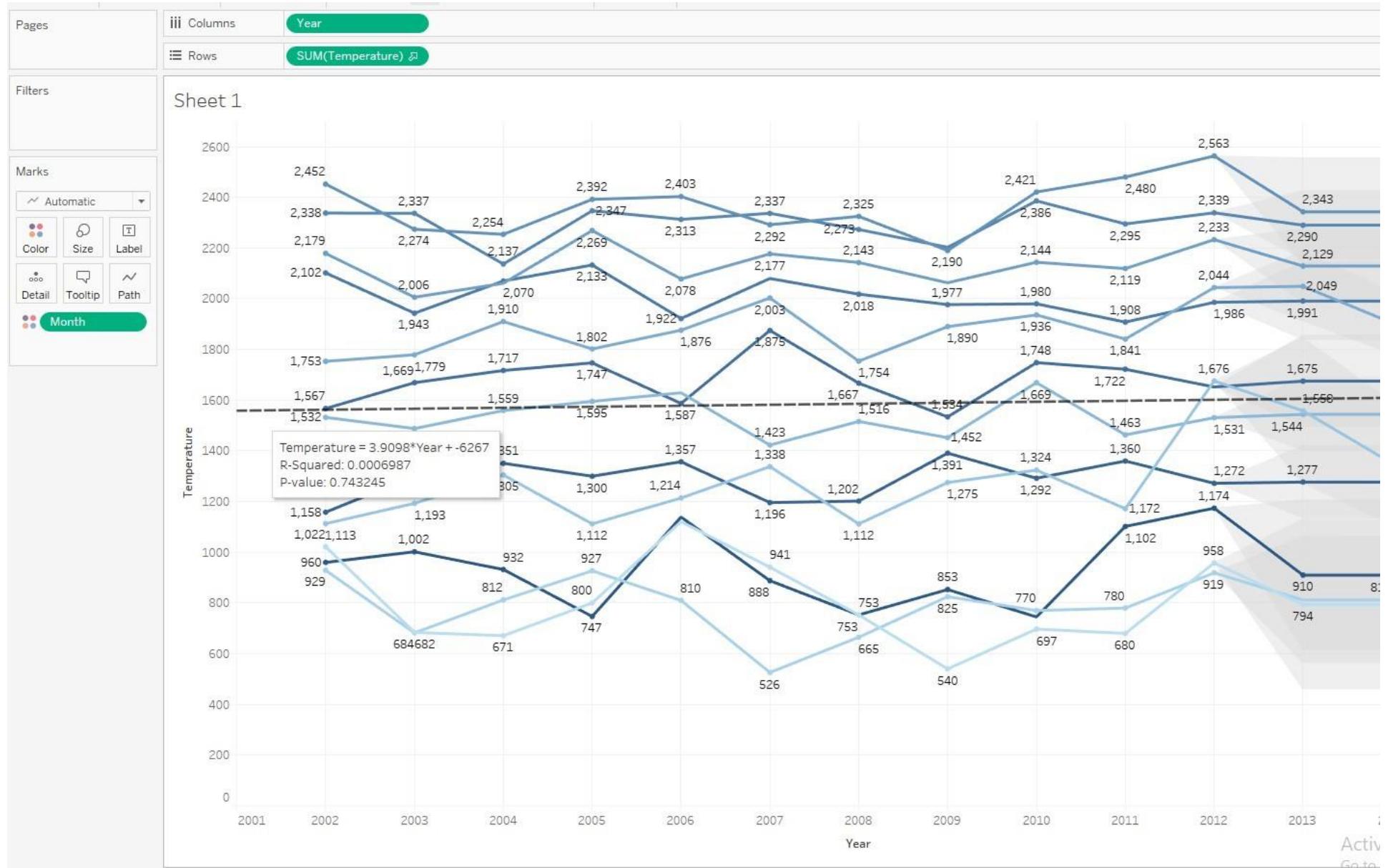
Inference: For each year, Both temperature and daylight hours has seen a continuous increase followed by a dip with the year 2012, temperature hitting an all time high of 20,347. For the year 2014, the temperature is predicted to significantly decrease to 18,853. Due to not so much changes in the temperature and daylight hours, the trend line is constant throughout the graph.

Fig: Forecast indicator using a heatmap to analyse change and estimation in temperature



Inference: The brighter the color gets, the higher the daylight hour for that year

Fig: A trend and forecast analysis chart for temperature for each month of the year along with the trend line



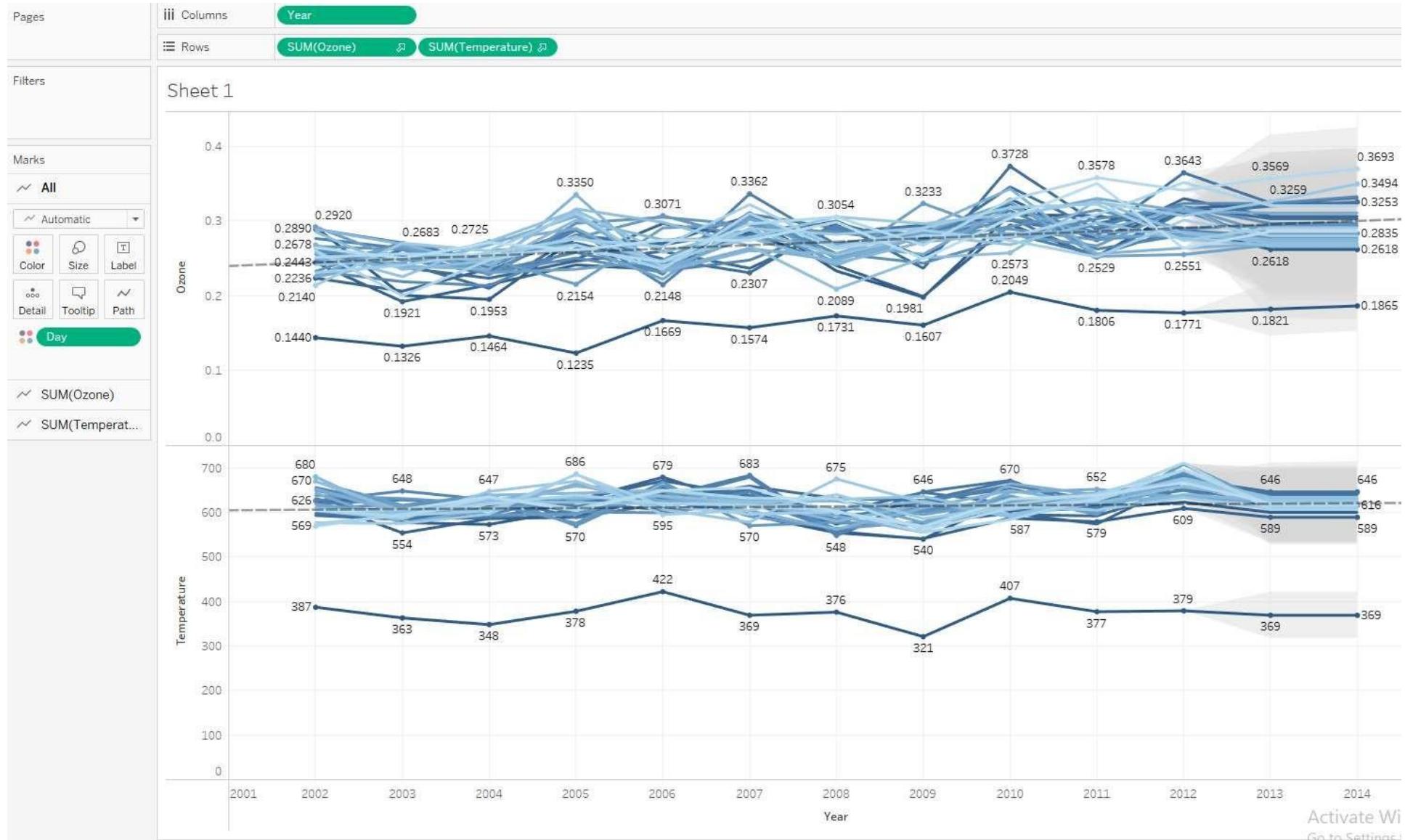
Inference: There is an increase in temperature starkly during certain months of the year particularly middle months of the year

Fig: Comparing temperature and ozone for each year month wise



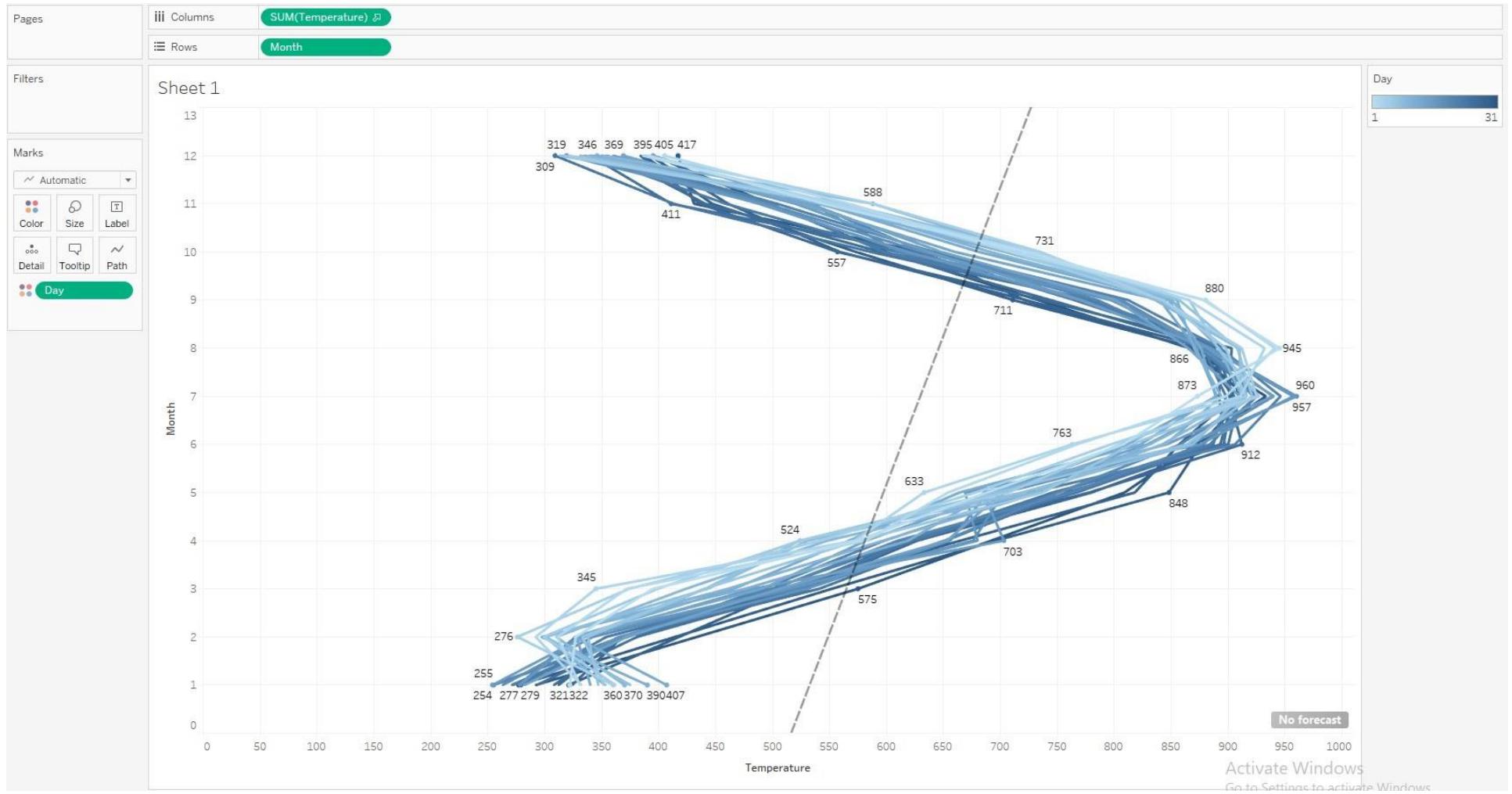
Inference: Both ozone and temperature show random increase and decrease month wise however there is a constant increase in ozone when done year wise without monthly interference.

Fig: Comparing temperature and ozone for year day wise



Inference: The Day 31 is distinctly having a lower temperature and ozone level for all the years starting from 2004 to 2013 .Both ozone and temperature show random increase and decrease day wise however there is a constant increase in ozone when done year wise without monthly interference.

Fig: Comparison of temperature and month and finding trends and forecast in them.



Case study 4:

- 1) Visualize the scatter plot, (points should not overlap), Scatterplot With Encircling, Jitter Plot for “Heart dataset” with respect to chol – cholesterol. Use different color and shape for each plot. Implement this using R.
- 2) Explore the visual insight by Creating 5 Sheets with one Dashboard in tableau using the above dataset. Apply Tableau python integration to show the impact of heart attack based of cholesterol.

1) R CODE w OUTPUT:

```
library(corrplot)
```

```
df1 = read.csv(file="heart.csv")
```

```
head(df1)
```

```
df1 = df1[complete.cases(df1),]
```

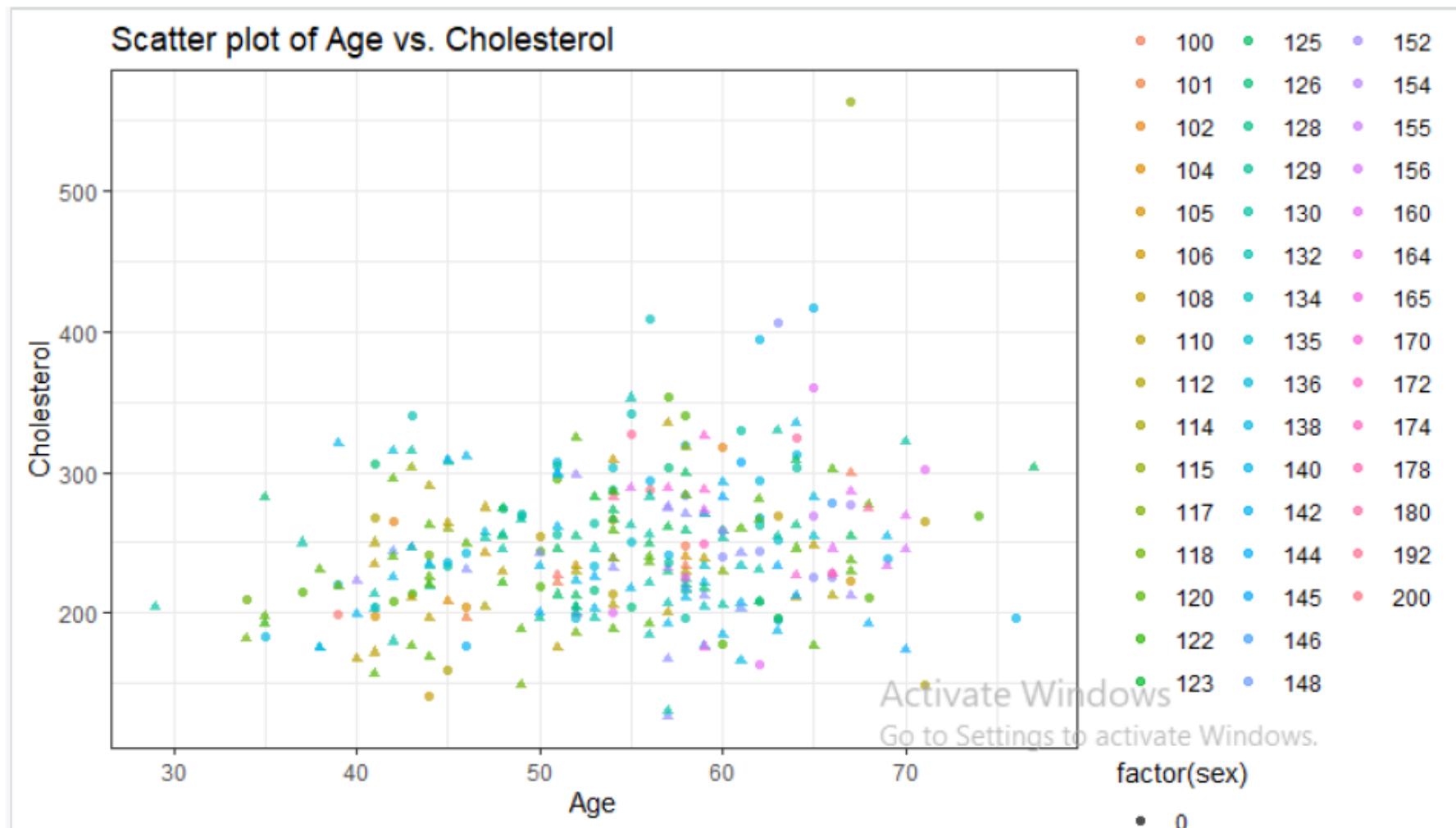
	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	s1p	caa	thall	output
1	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
2	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
3	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
4	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
5	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
6	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1

```
# scatterplot
```

```
library(ggplot2)
```

```
ggplot(df1, aes(x = age, y = chol, color = factor(trtbps), shape = factor(sex))) +
```

```
geom_point(alpha = 0.7) + labs(x = "Age", y = "Cholesterol", title = "Scatter plot of Age vs.  
Cholesterol") + theme_bw()
```



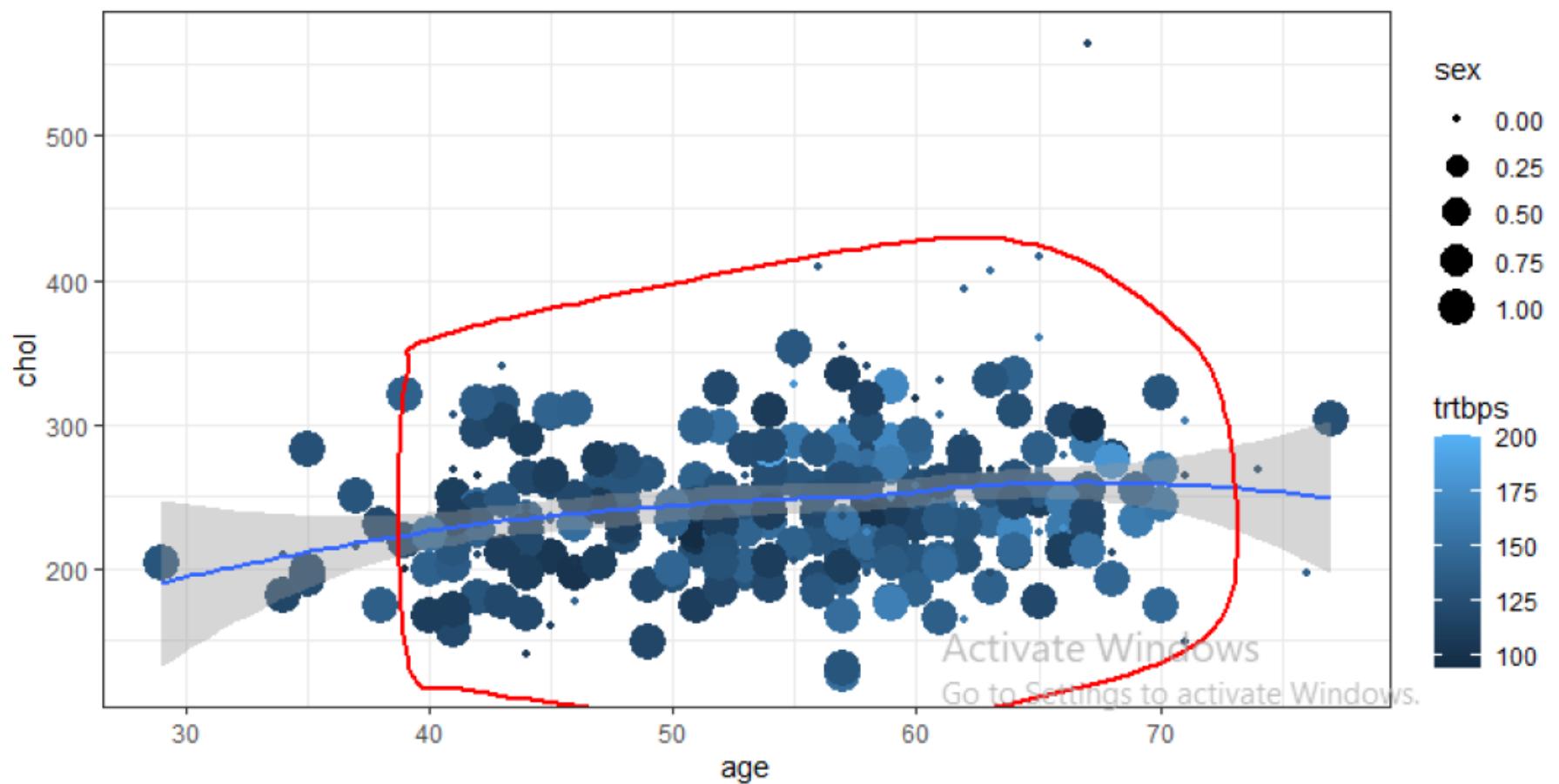
```
# Scatterplot with encircling
library(ggalt)

# Scatterplot with encircling
library(ggalt)

condition = df1[df1$age > 40 & df1$age <= 70 & df1$chol > 0 & df1$chol < 400, ]
ggplot(df1, aes(x=age, y=chol)) + geom_point(aes(col=trtbps, size=sex)) +
geom_smooth(method="loess", sex) + geom_encircle(aes(x=age, y=chol), data=condition,
color="red", size=2, expand=0.08) + labs(subtitle="Age Vs Cholesterol", y="chol", x="age",
title="Scatterplot + Encircle")
```

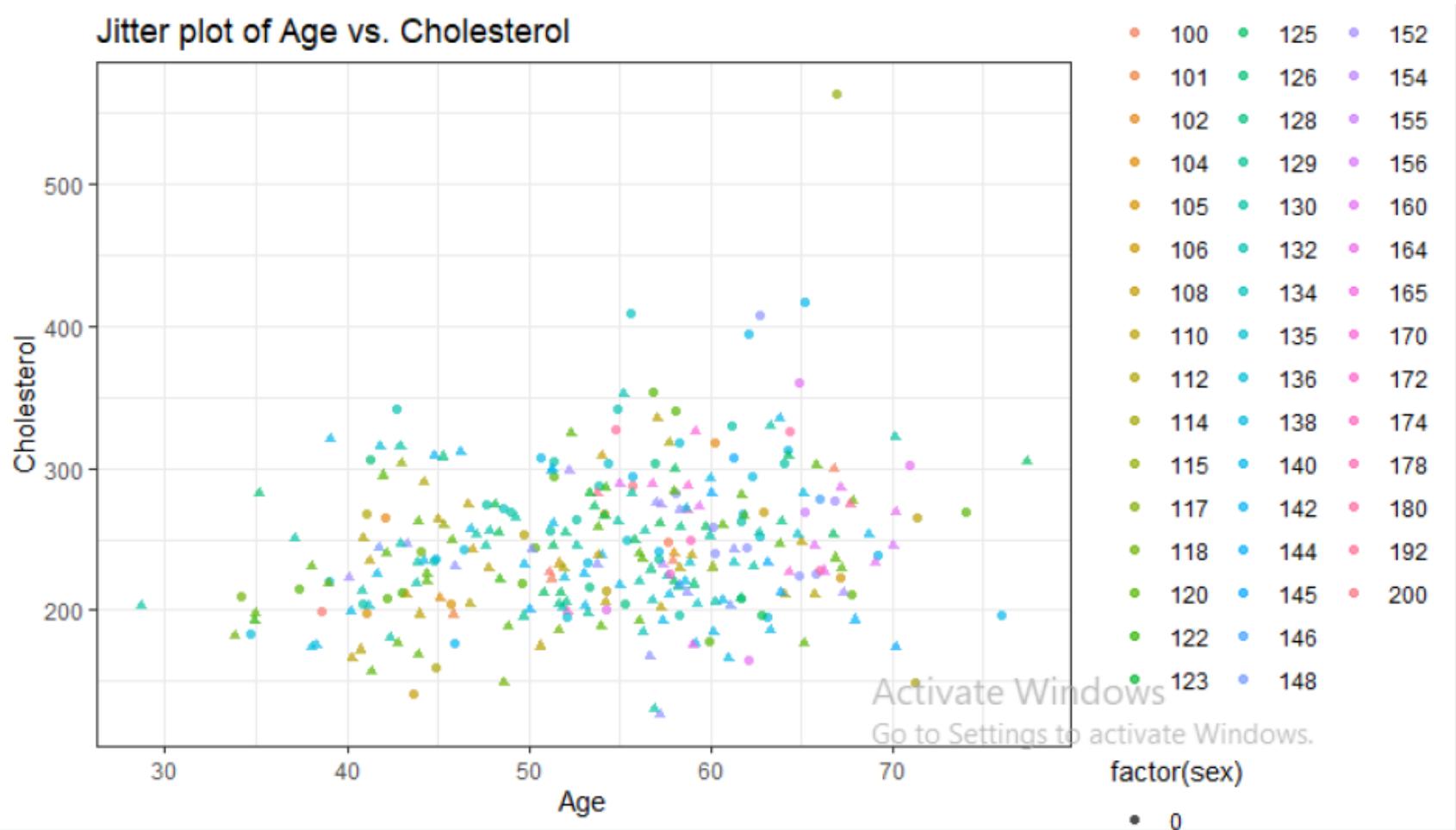
Scatterplot + Encircle

Age Vs Cholesterol



```
#jitterplot
```

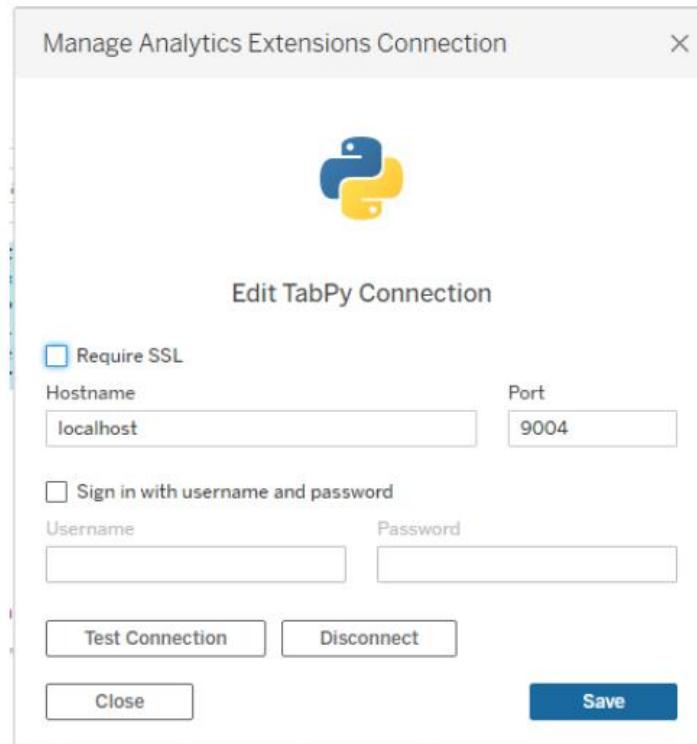
```
ggplot(df1, aes(x = age, y = chol, color = factor(trtbps), shape = factor(sex))) +geom_jitter(alpha = 0.7) +  
  labs(x = "Age", y = "Cholesterol", title = "Jitter plot of Age vs. Cholesterol") +  
  theme_bw()
```



2) TABLEAU WITH TABPY:

Create the tabpy connection

```
(C:\Users\admin\Anaconda\envs\Tableau-Python-Server) C:\Users\admin>tabpy
2023-04-12,15:27:25 [INFO] (app.py:app:244): Parsing config file c:\users\admin\anaconda\envs\tableau-python-server\lib\site-packages\tabpy\tabpy_server\app\..\common\default.conf
2023-04-12,15:27:25 [INFO] (app.py:app:436): Loading state from state file c:\users\admin\anaconda\envs\tableau-python-server\lib\site-packages\tabpy\tabpy_server\state.ini
2023-04-12,15:27:25 [INFO] (app.py:app:333): Password file is not specified: Authentication is not enabled
2023-04-12,15:27:25 [INFO] (app.py:app:347): Call context logging is disabled
2023-04-12,15:27:25 [INFO] (app.py:app:125): Initializing TabPy...
2023-04-12,15:27:25 [INFO] (callbacks.py:callbacks:43): Initializing TabPy Server...
2023-04-12,15:27:25 [INFO] (app.py:app:129): Done initializing TabPy.
2023-04-12,15:27:25 [INFO] (app.py:app:83): Setting max request size to 104857600 bytes
2023-04-12,15:27:25 [INFO] (callbacks.py:callbacks:64): Initializing models...
2023-04-12,15:27:25 [INFO] (app.py:app:107): Web service listening on port 9004
```

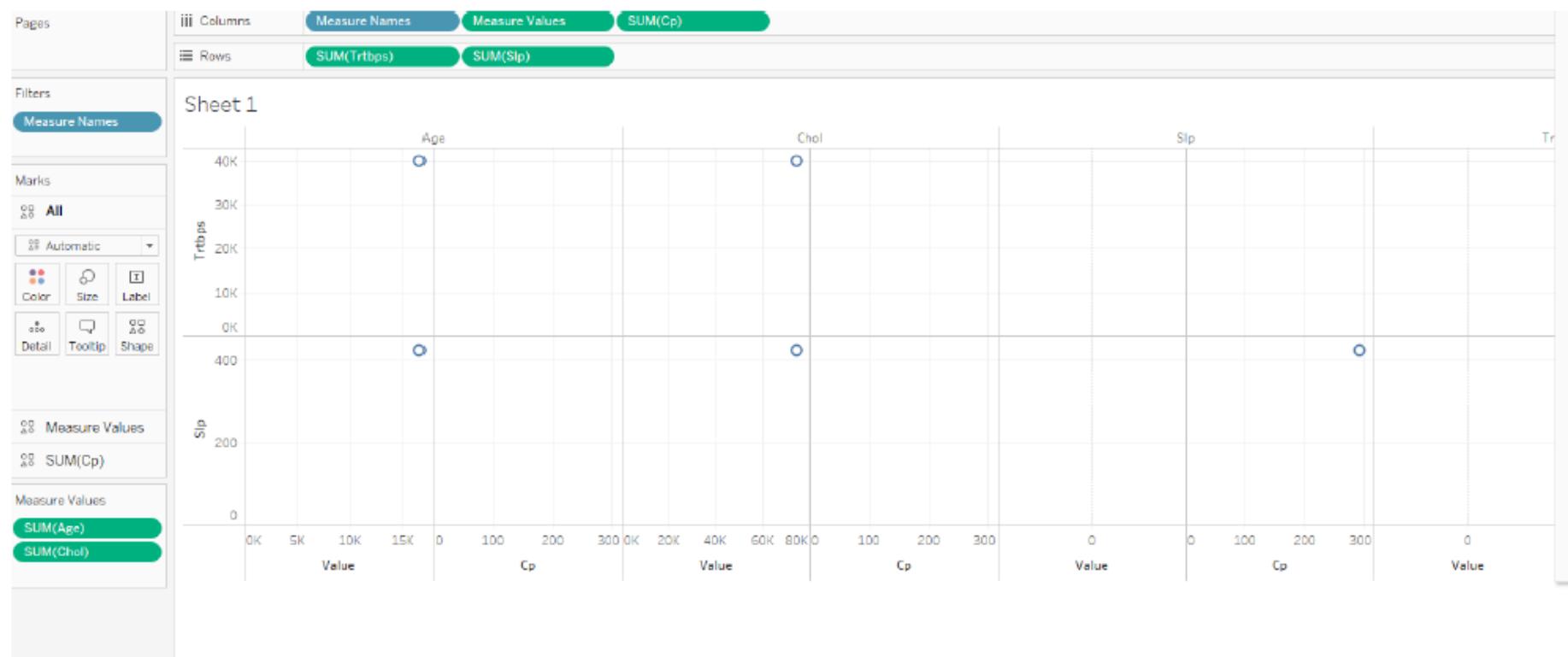


Data Source

heart.csv ▾ 15 fields 303 rows

Name heart.csv				# heart.csv							
Type	Field Name	Physical Table	Remote Field Name	Age	Sex	Cp	Trtbps	Chol	Fbs	Restecg	Thalachh
#	Age	heart.csv	age	63	1	3	145	233	1	0	150
#	Sex	heart.csv	sex	37	1	2	130	250	0	1	187
#	Cp	heart.csv	cp	41	0	1	130	204	0	0	172
#	Trtbps	heart.csv	trtbps	56	1	1	120	236	0	1	178
#	Chol	heart.csv	chol	57	0	0	120	354	0	1	163
#	Fbs	heart.csv	fbs	57	1	0	140	192	0	1	148
				56	0	1	140	294	0	0	153
				44	1	1	120	263	0	1	173
				52	1	2	172	199	1	1	162
				57	1	2	150	168	0	1	174

Sheet 1:



Dashboard

Dashboard Story Analysis Map Format Server Window

New Dashboard

```
#PYTHON SCRIPT
SCRIPT_REAL(
import tabpy
import pandas as pd
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data frame
df = pd.DataFrame(_arg1)

# Calculate the correlation between cholesterol and heart attack
corr, pvalue = pearsonr(df["chol"], df["target"])

# Create a scatter plot of cholesterol vs. heart attack
sns.scatterplot(data=df, x="chol", y="target", hue="sex", style="thal")

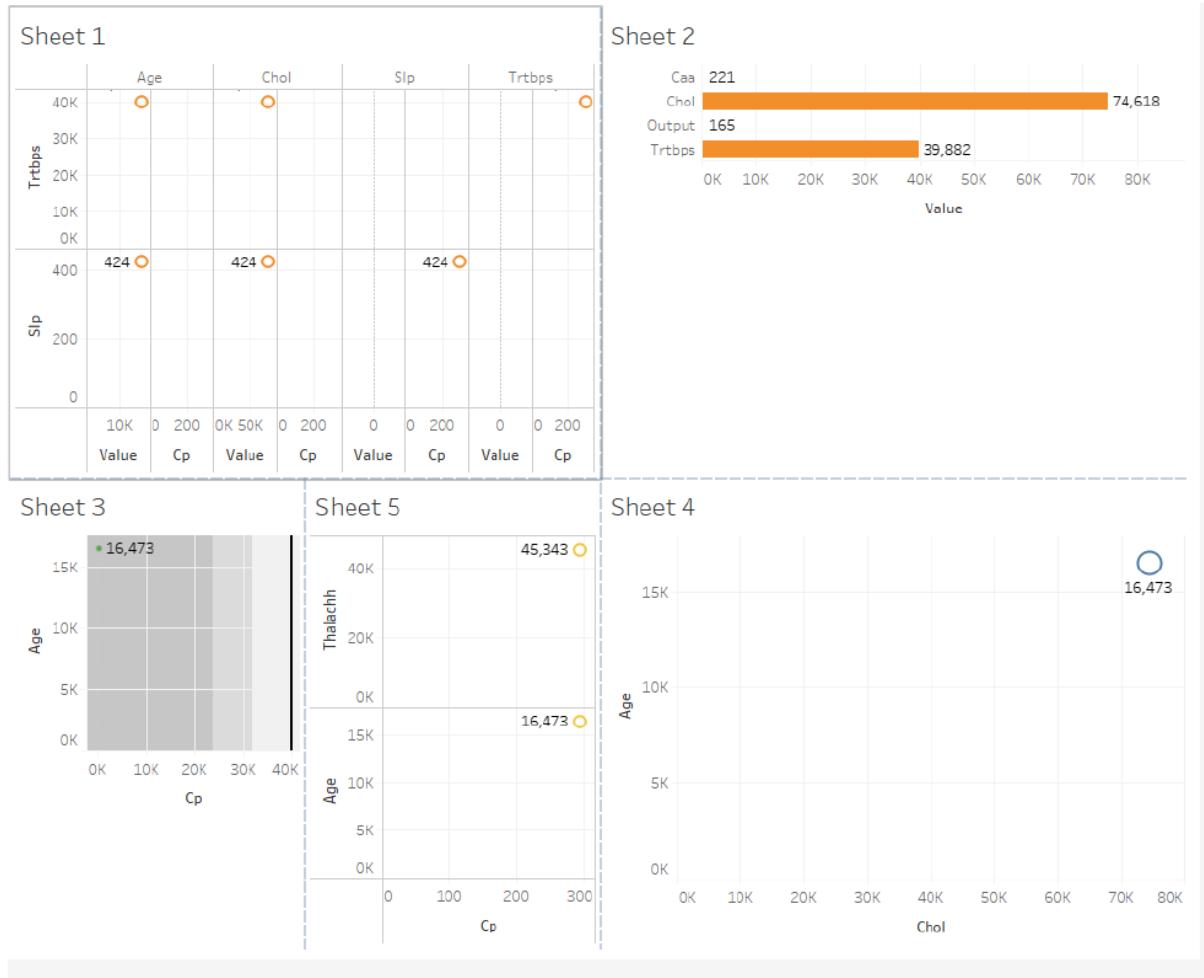
# Add the correlation coefficient
plt.annotate(f"Corr: {corr:.2f} (p = {pvalue:.2f})", xy=(250, 0.8), fontsize=12)

# Save the plot as a PNG file
plt.savefig("heart.png")
```

```

return {"image_path": "heart.png"}  

', ATTR([Sex]), ATTR([thal]), SUM([Chol]), SUM([target]))|
```



Case study 5:

Implement using R – statistical modeling and Visualize a Scatterplot with Encircling, Jitter Plot, Bubble plot, Correlogram, Diverging Lollipop Chart for given “Auto dataset”.

Apply Tableau R integration to understand the impact of horsepower with mpg and show the 5 different inferences.

R PROGRAMMING

#STATISTICAL PLOTTING

```
library(ggplot2)  
library(ggc当地  
library(ggExtra)  
library(GGally)  
library(scales)  
library(ggalt)  
  
DATA <- read.csv("D:/Auto.csv",header = TRUE)  
head(DATA)  
summary(DATA)  
str(DATA)
```

```
#SCATTERPLOT WITH ENCIRCLING
```

```
DT_CHECK <- DATA[DATA$mpg > 25,]
```

```
ggplot(DATA,aes(x = mpg,y = horsepower,col = year)) + geom_point(aes(x = mpg,y = horsepower)) +
```

```
    geom_encircle(aes(x = mpg,y = horsepower),DT_CHECK,type = "l",col =  
"red",stat = "identity",fill_palette = NULL)
```

```
ggplot(DATA,aes(x = mpg,y = horsepower,col = year)) + geom_point(aes(x = mpg,y = horsepower)) +
```

```
    geom_encircle(aes(x = mpg,y = horsepower),DT_CHECK,type = "l",col =  
"red",stat = "identity",fill_palette = NULL)
```

```
#JITTERPLOT
```

```
ggplot(DATA,aes(x = mpg,y = horsepower,fill = year,col = year)) +  
geom_smooth(aes(x = mpg,y = cylinders)) #+ geom_smooth(DATA,aes( mpg,  
horsepower),stat = "smooth")
```

```
ggplot(DATA,aes(x = mpg,y = horsepower,fill = year,col = year)) +  
geom_jitter(aes(x = mpg,y = cylinders)) + geom_smooth(aes(x = mpg,y =  
cylinders))
```

```
#BUBBLE PLOT
```

```
ggplot(DATA,aes(x = mpg,y = horsepower,fill = year,col = origin)) +  
  geom_point(aes(x = mpg,y = horsepower),size=DATA$cylinders)  
 +geom_smooth(aes(x = mpg ,y = horsepower))
```

```
#CORRELOGRAM
```

```
DATA_C <- DATA
```

```
DATA_C[,4] <- as.numeric(DATA_C[,4])
```

```
DATA_C[,8] <- as.numeric(DATA_C[,8])
```

```
DATA_C[,9] <- DATA_C[,-9]
```

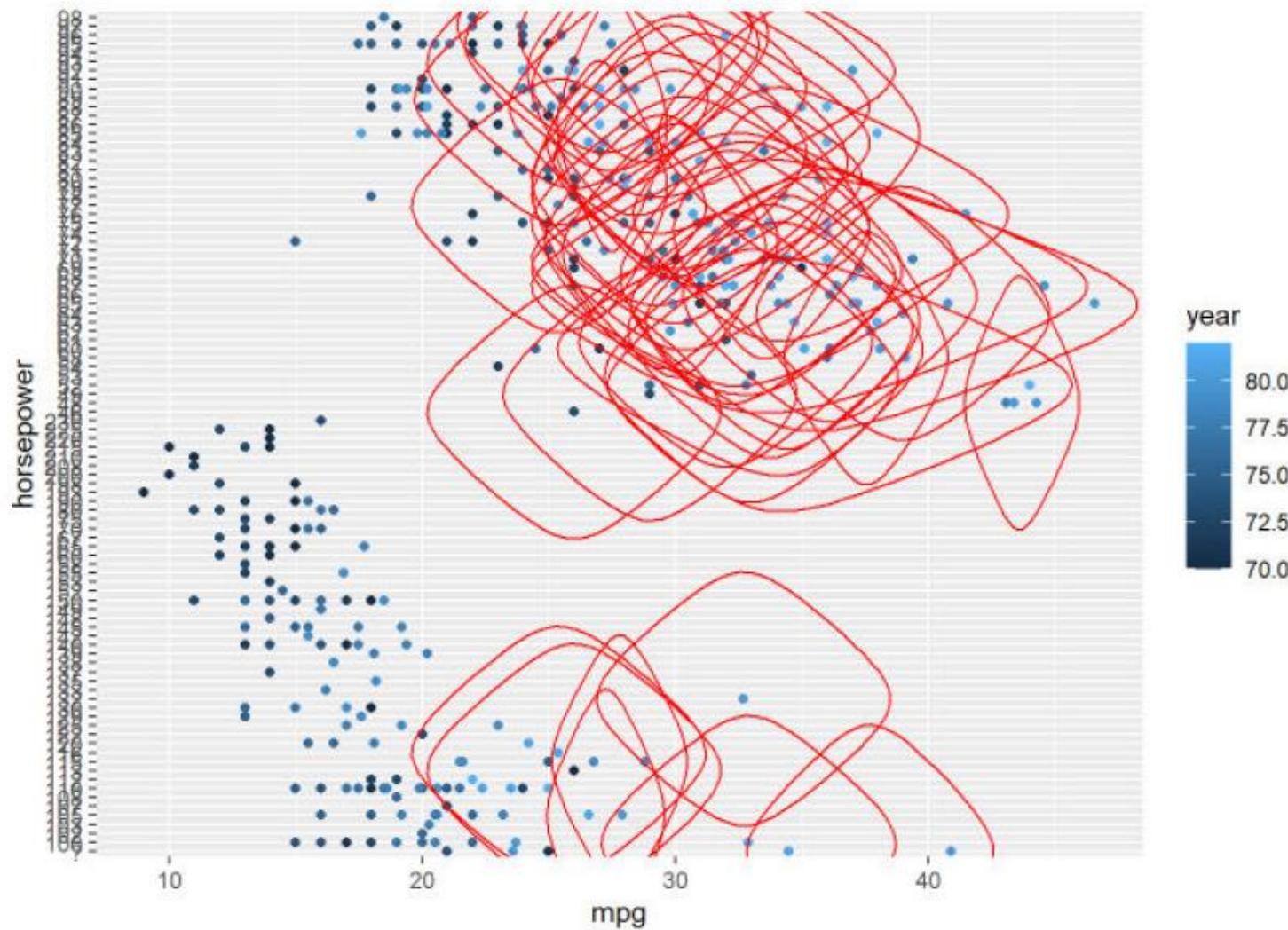
```
DATA_C[is.na(DATA_C)] <- 0
```

```
CORR <- cor(DATA_C)
```

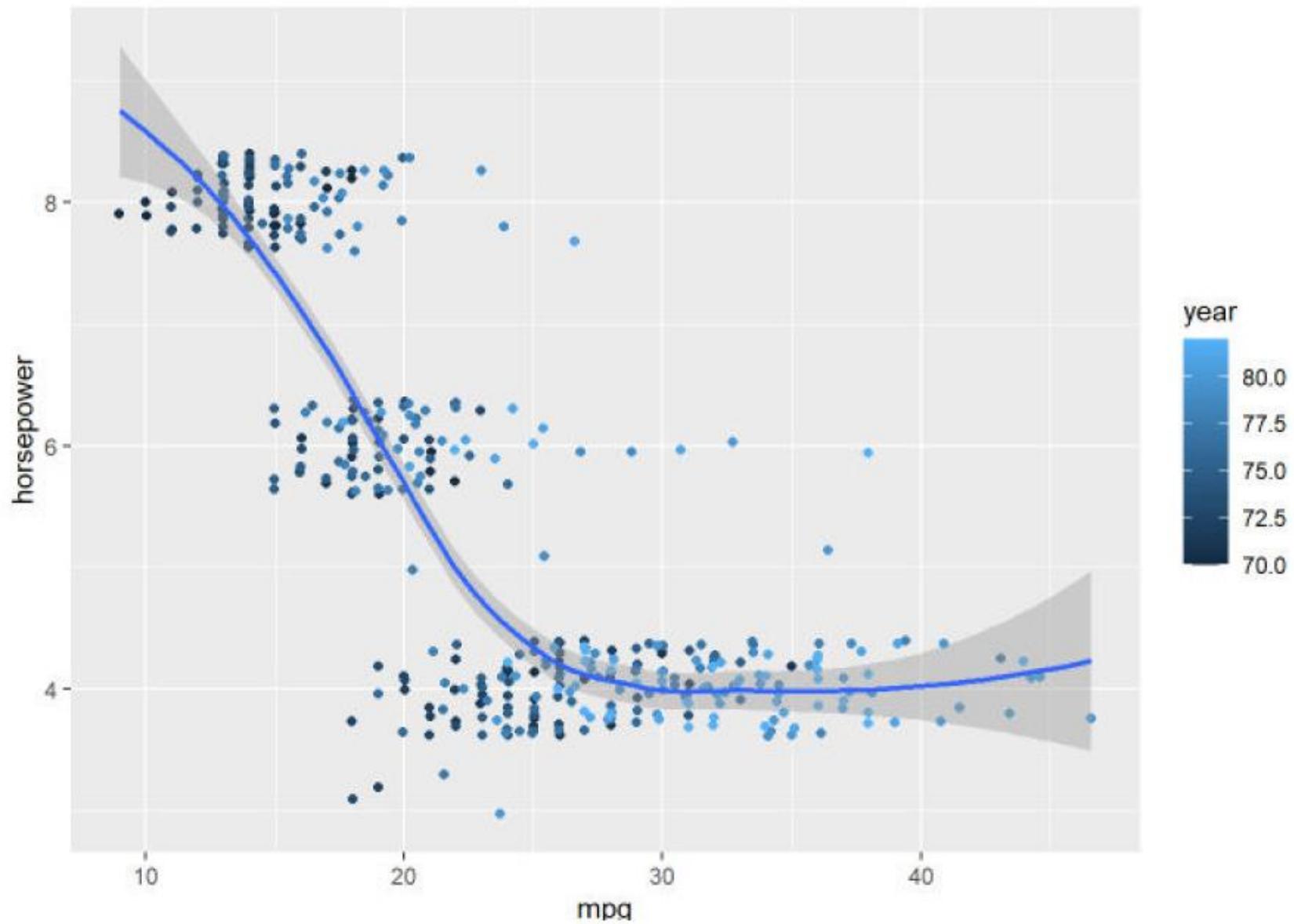
```
ggcorrplot(CORR,method = "circle", type="lower",title = "CORRELOGRAM OF  
DATA",show.legend = TRUE)
```

OUTPUT

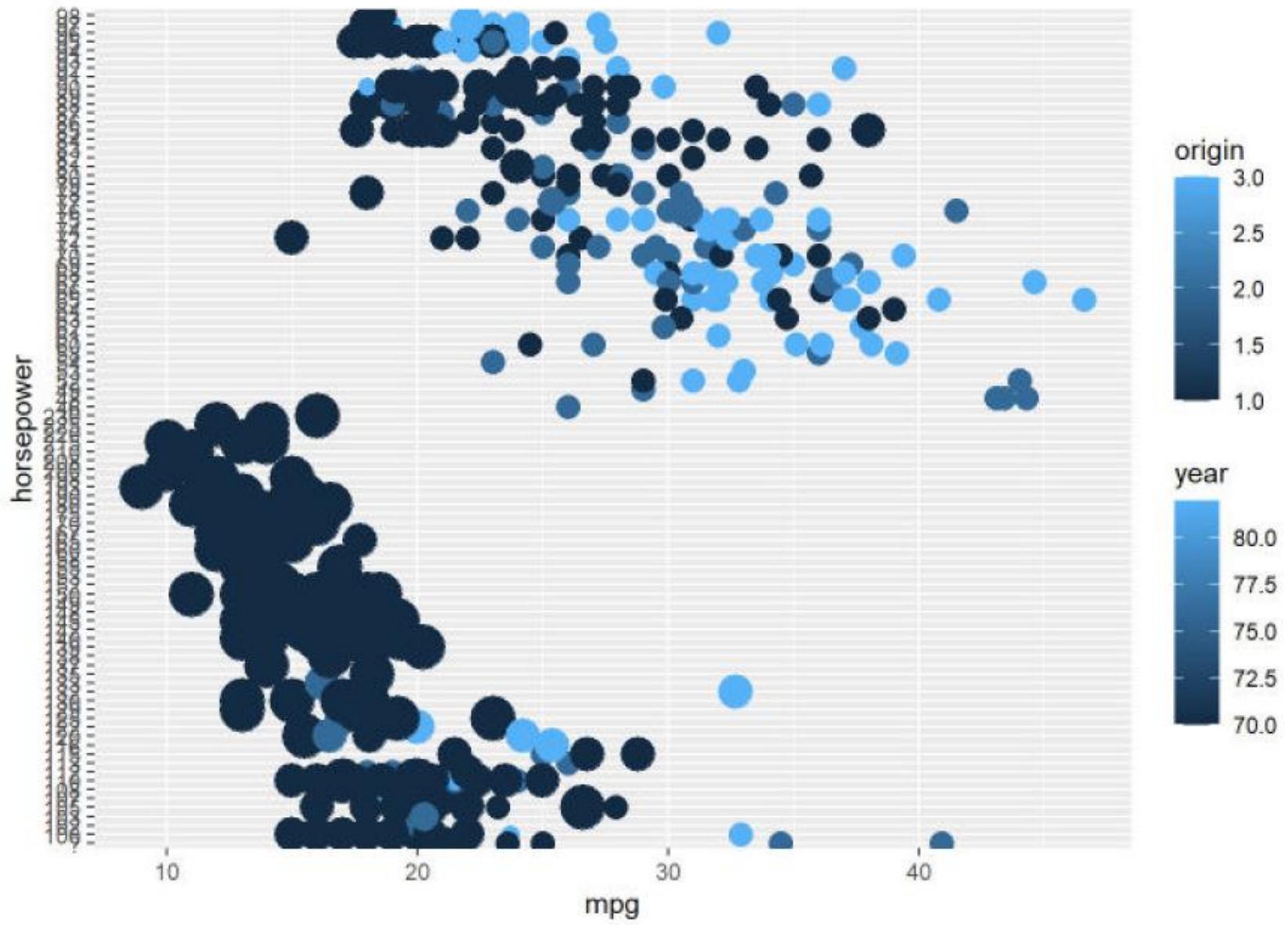
INFERENCE - Encircled those which have mpg > 25



INFERENCE - Jitter plot shows that points of scatterplot of horse power and mpg where higher to lower through cylinders throughout the year than the further years.



BUBBLE PLOT - Shows the footprint and size of horsepower and mpg with origins as classification



CORRELALOGRAM

INFERENCE - Shows the relationship between different attributes in the matrix.

CORRELOGRAM OF DATA

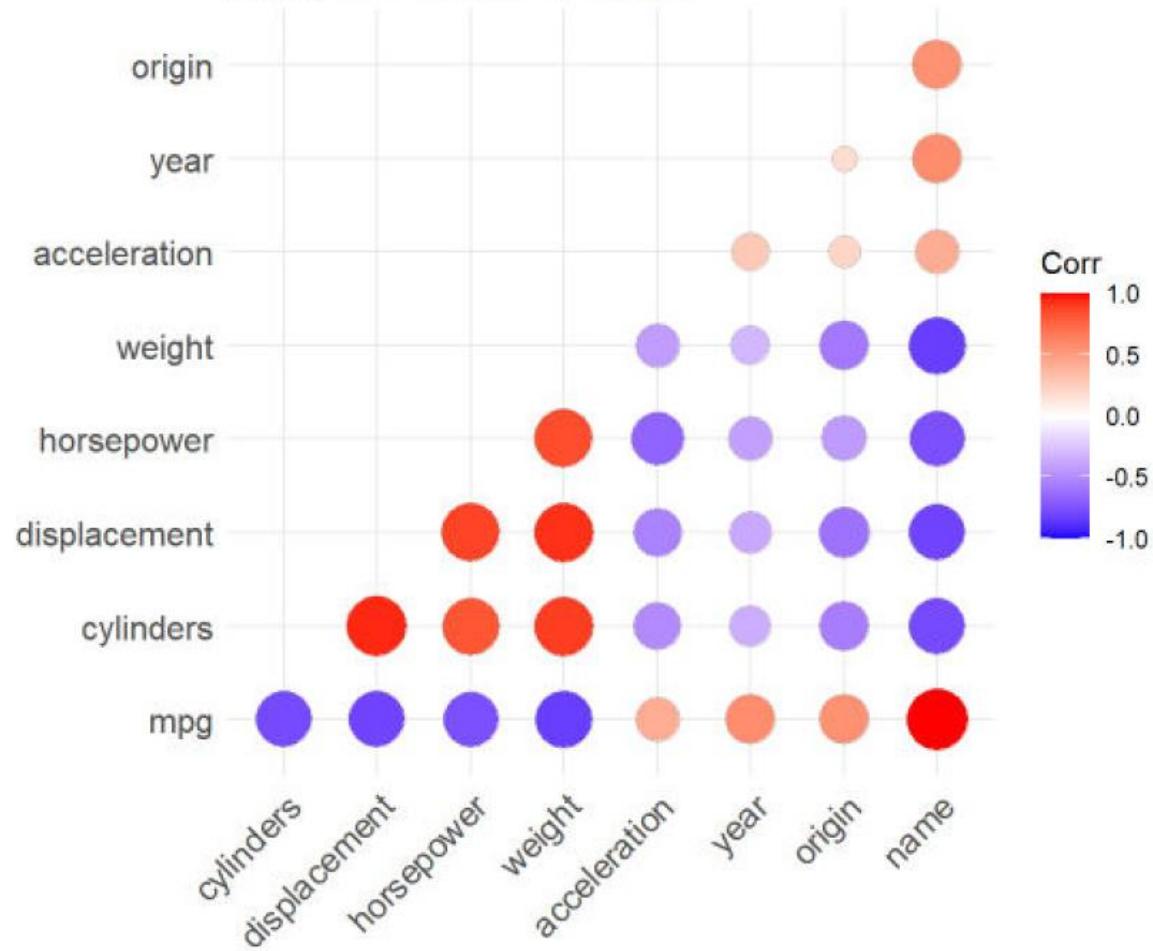
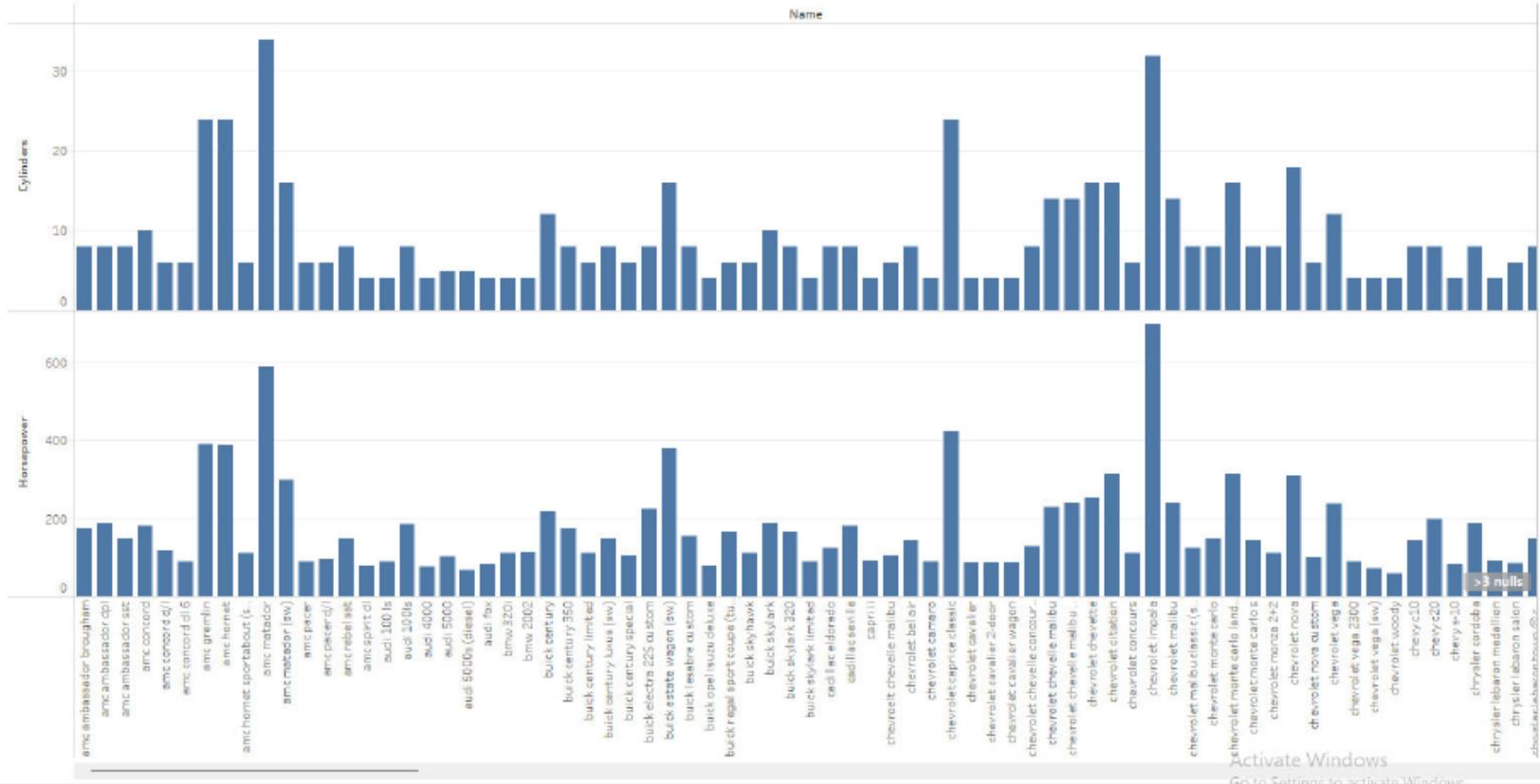


TABLEAU VISUALIZATIONS

INFERENCE 1 --> Shows Names against its horsepower and Cylinders

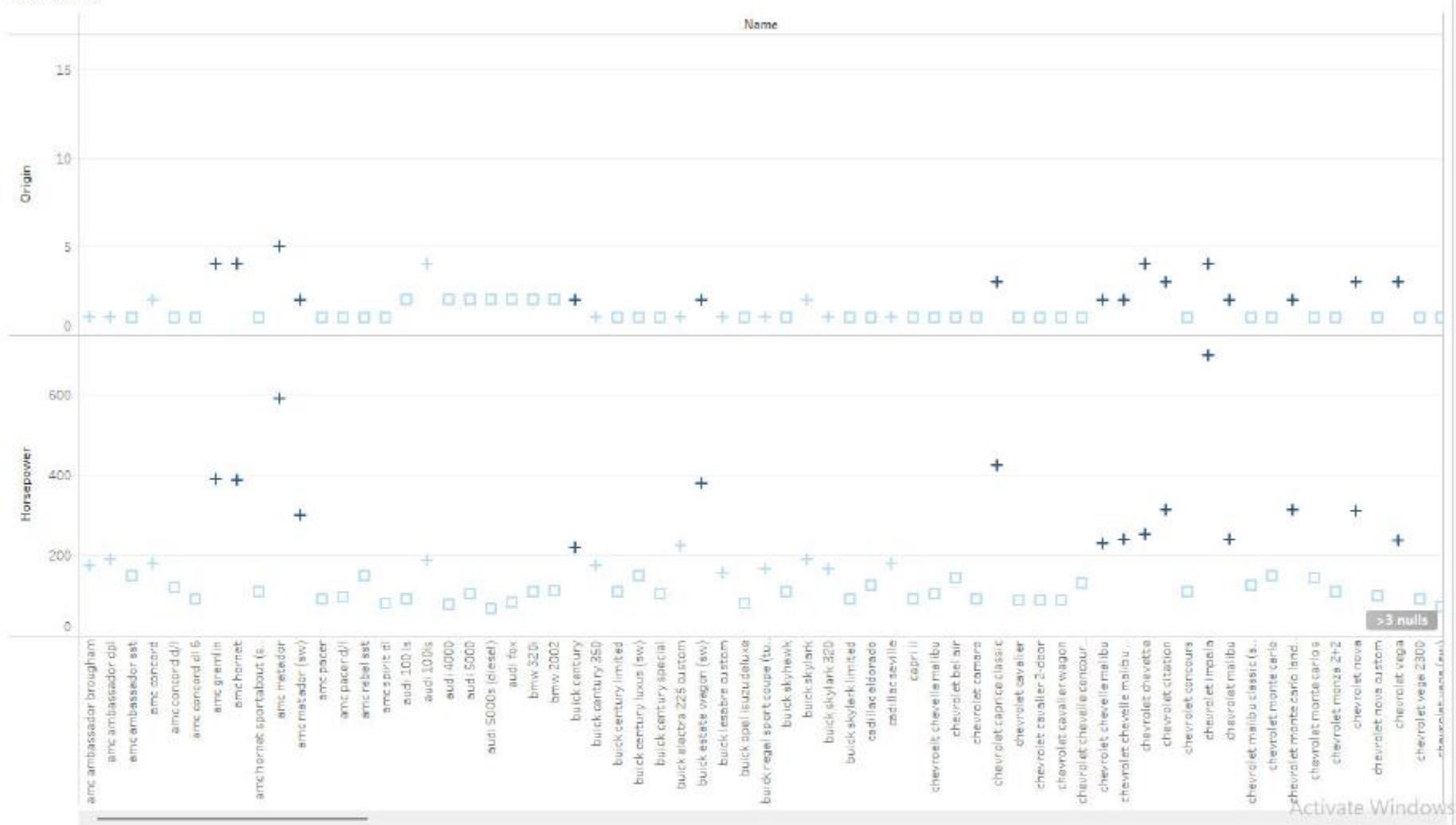
It shows that cheverlot and hornet have highest horsepower with highest cylinders

INFERENCE 1



It has visualization of horsepower and origin against names with clustering of origin and weights and horsepower being > 150.

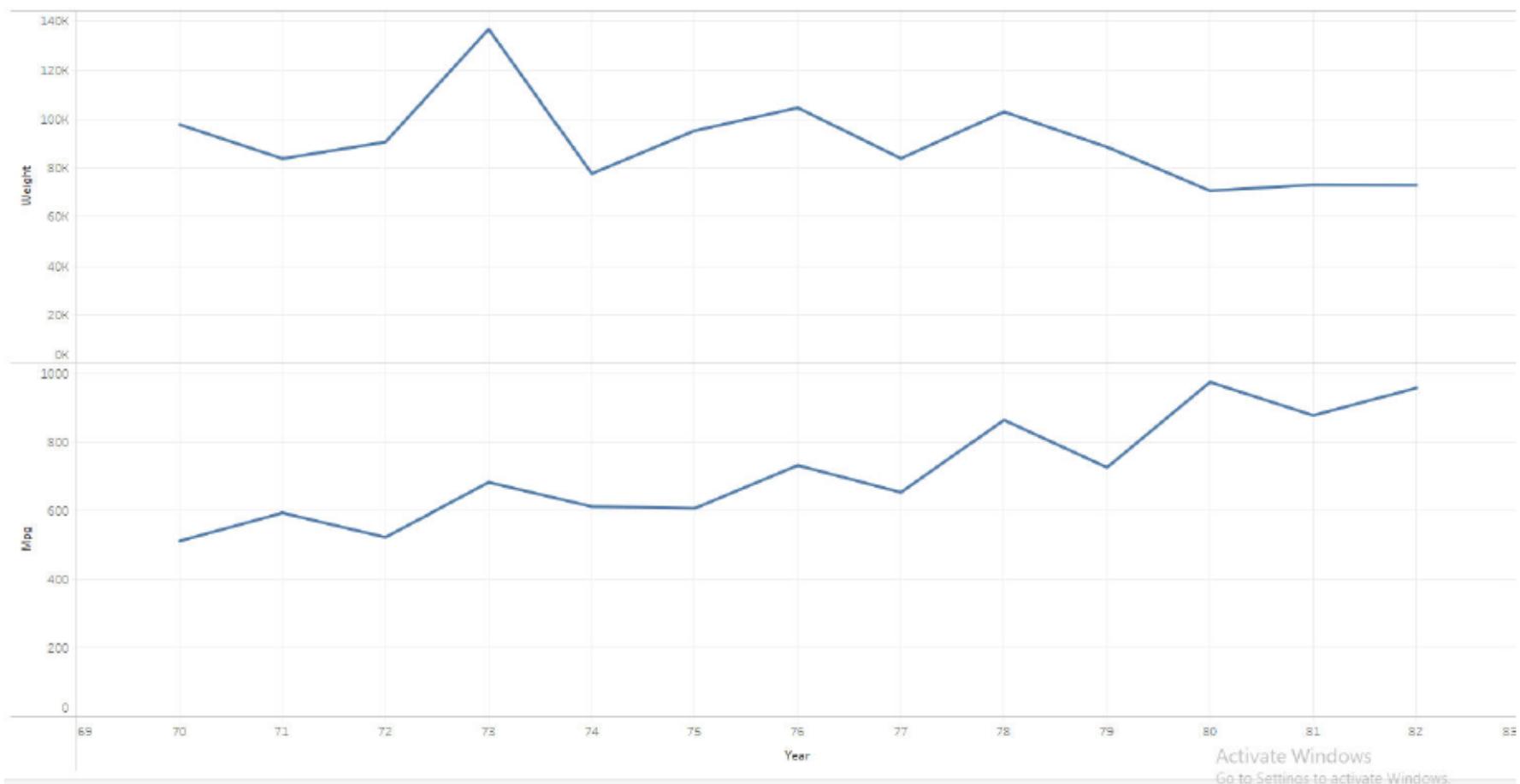
CLUSTER



INFERENCE

Time series analysis clearly shows that as weight has been decreasing at a slower rate the mpg is being increasing over the years.

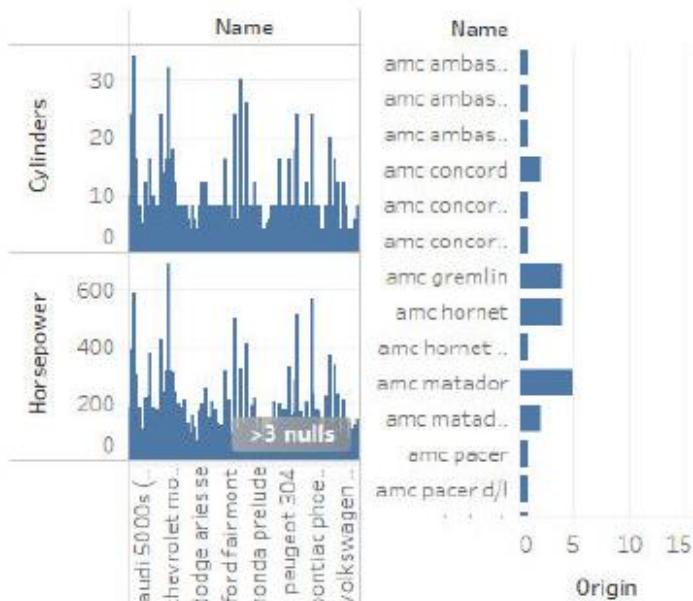
INFERENCE 3



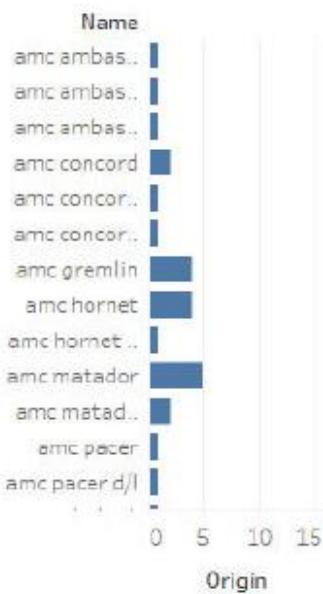
INFERENCE

This plot simply shows the scatterplot with horsepower which highlights horsepower > 150.

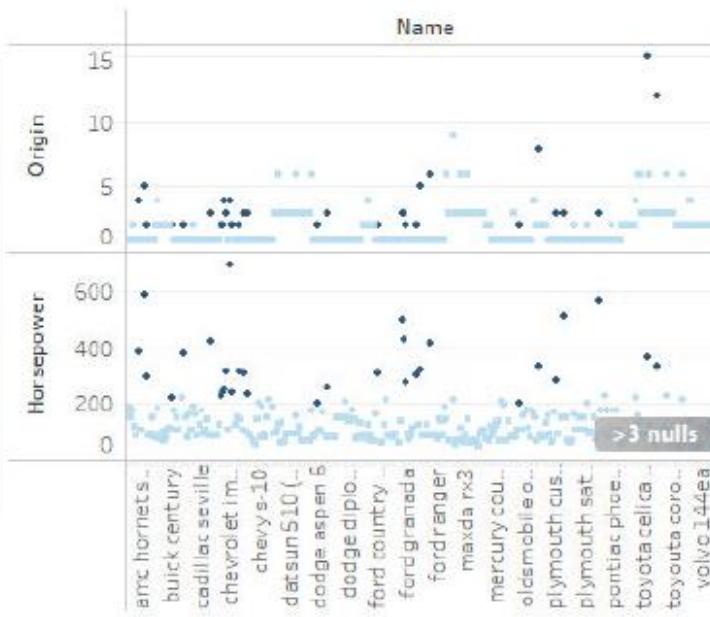
INFERENCE 1



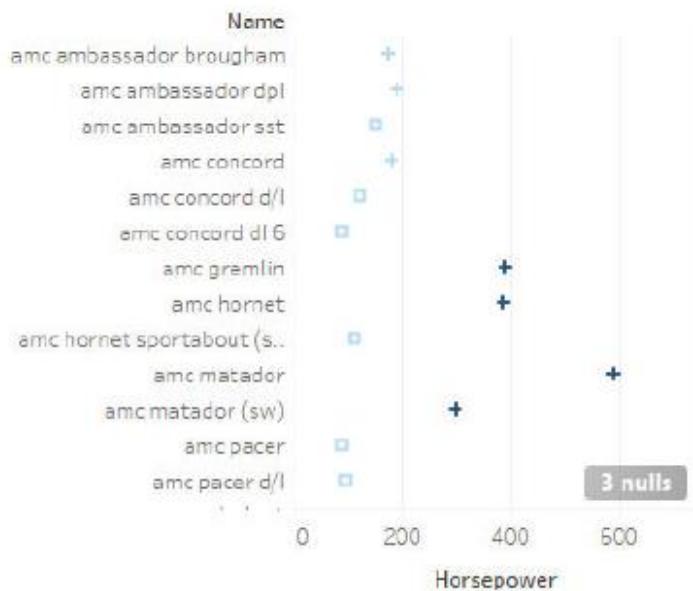
INFERENCE 5



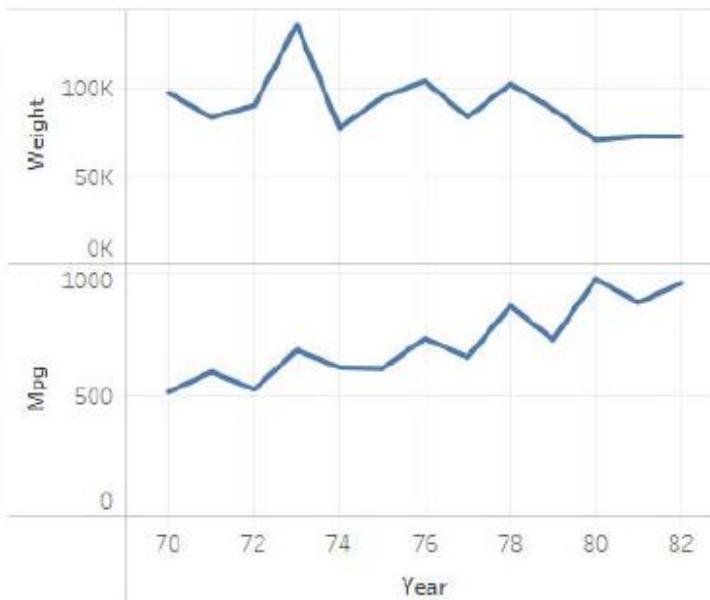
CLUSTER



INFERENCE 4



INFERENCE 3



Case Study 6:

Use default “mtcars” dataset to make a scatterplot of “weight” and “mpg”. Change axes labels and give your plot a title. Set the alpha of all points to 0.7. Set the color of the points to the “mpg” variable (continuous). Also, set this color parameter so that high mpgs are colored green and low mpgs are colored blue. Title the legend “MPG”. Set the size of the points to the “wt” variable (continuous). Also, to make the small points more readable, set the range of sizes to go from 3-10. Title this legend “Weight”. Implement this visualization using R.

```
```{r}
library(ggplot2)
library(data.table)
library(dplyr)
data <- mtcars
head(data)
dim(data)
```
```

Output:

The screenshot shows the RStudio interface. On the left, the R Console pane displays the loading of the 'ggplot2' package and the creation of a data frame named 'df'. The Data View pane on the right shows the 'mpg' dataset as a data frame with 6 rows and 11 columns. Below the Data View, a tooltip provides a detailed description of the data frame.

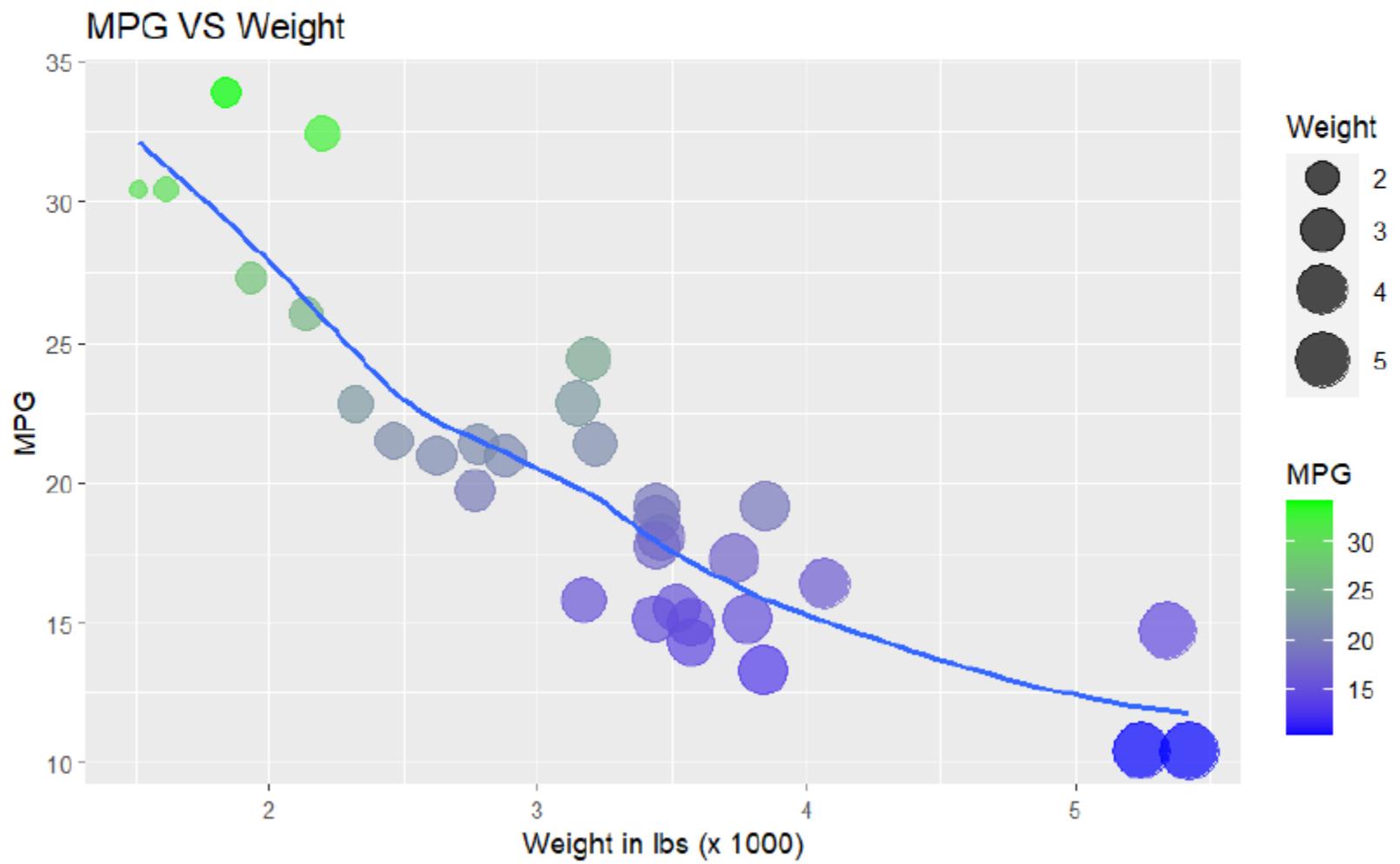
| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | ... |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|-----|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | |

6 rows | 1-10 of 11 columns

Code:

```
27 - ````{r}
28   gg <- ggplot(data, aes(wt, mpg)) +
29     geom_point(alpha=0.7, aes(size=wt, color=mpg)) +
30     geom_smooth(method="loess", se=F) +
31     scale_size(name="Weight", range=c(3,10)) +
32     scale_color_gradient(name="MPG", low="blue", high="green") +
33     labs(x="Weight in lbs (x 1000)", y="MPG", title="MPG VS Weight")
34   gg
35 - ````
```

Output:



Q2.

Predict the better insight by Creating 5 Sheets with one Dashboard for given dataset and explore the impact of “wt” weight on “mpg” – mileage per Galan by integrating python with tableau using the “mtcars” dataset. Apply the calculated field in your experiment to show the results.

TabPy Connection :

```
Command Prompt - tabpy
Microsoft Windows [Version 10.0.22000.1817]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>pip intsall tabpy
ERROR: unknown command "intsall" - maybe you meant "install"

C:\Users\admin>pip install tabpy
Collecting tabpy
  Downloading tabpy-2.4.0-py2.py3-none-any.whl (110 kB)
    ██████████ | 110 kB ...
Requirement already satisfied: pytest in c:\users\admin\anaconda\lib\site-packages (from tabpy) (3.0.5)
Collecting coveralls
  Downloading coveralls-3.3.1-py2.py3-none-any.whl (14 kB)
    ██████████ | 14 kB ...
Requirement already satisfied: pandas in c:\users\admin\anaconda\lib\site-packages (from tabpy) (0.19.2)
Collecting simplejson
  Downloading simplejson-3.19.1-cp36-cp36m-win_amd64.whl (76 kB)
    ██████████ | 76 kB ...
Requirement already satisfied: nltk in c:\users\admin\anaconda\lib\site-packages (from tabpy) (3.2.2)
Requirement already satisfied: jsonschema in c:\users\admin\anaconda\lib\site-packages (from tabpy) (2.5.1)
Collecting hypothesis
  Downloading hypothesis-6.31.6-py3-none-any.whl (377 kB)
    ██████████ | 377 kB ...
Collecting configparser
  Downloading configparser-5.2.0-py3-none-any.whl (19 kB)
Collecting twisted
  Downloading Twisted-21.2.0-py3-none-any.whl (3.1 MB)
    ██████████ | 3.1 MB ...
```

Command Prompt - tabpy

```
Found existing installation: idna 2.2
Uninstalling idna-2.2:
  Successfully uninstalled idna-2.2
Attempting uninstall: pytest
  Found existing installation: pytest 3.0.5
  Uninstalling pytest-3.0.5:
    Successfully uninstalled pytest-3.0.5
Successfully installed Automat-22.10.0 atomicwrites-1.4.1 attrs-22.2.0 configparser-5.2.0 constantly-15.1.0 coverage-6.2
coveralls-3.3.1 docopt-0.6.2 future-0.18.3 genson-1.2.2 hyperlink-21.0.0 hypothesis-6.31.6 idna-3.4 importlib-metadata-
4.8.3 incremental-22.10.0 iniconfig-1.1.1 mock-5.0.1 pluggy-1.0.0 py-1.11.0 pytest-7.0.1 pytest-cov-4.0.0 simplejson-3.1
9.1 sklearn-0.0.post1 sortedcontainers-2.4.0 tabpy-2.4.0 textblob-0.17.1 tomli-1.2.3 twisted-21.2.0 twisted-iocpsupport-
1.0.3 typing-extensions-4.1.1 urllib3-1.26.15 zipp-3.6.0 zope.interface-5.5.2

C:\Users\admin>tabpy
2023-04-12,14:46:49 [INFO] (app.py:app:244): Parsing config file c:\users\admin\anaconda\lib\site-packages\tabpy\tabpy_
server\app..\common\default.conf
2023-04-12,14:46:49 [INFO] (app.py:app:436): Loading state from state file c:\users\admin\anaconda\lib\site-packages\tab
py\tabpy_server\state.ini
2023-04-12,14:46:49 [INFO] (app.py:app:333): Password file is not specified: Authentication is not enabled
2023-04-12,14:46:49 [INFO] (app.py:app:347): Call context logging is disabled
2023-04-12,14:46:49 [INFO] (app.py:app:125): Initializing TabPy...
2023-04-12,14:46:49 [INFO] (callbacks.py:callbacks:43): Initializing TabPy Server...
2023-04-12,14:46:49 [INFO] (app.py:app:129): Done initializing TabPy.
2023-04-12,14:46:49 [INFO] (app.py:app:83): Setting max request size to 104857600 bytes
2023-04-12,14:46:49 [INFO] (callbacks.py:callbacks:64): Initializing models...
2023-04-12,14:46:49 [INFO] (app.py:app:107): Web service listening on port 9004
```

Manage Analytics Extensions Connection

×



Edit TabPy Connection

Require SSL

Hostname: localhost

Port: 9004

Action Completed

Successfully connected to the analytics extension.

OK

TEST CONNECTION DISCONNECT

Close Save

Python Script :

Calculation1

X

Results are computed along Table (across).

```
SCRIPT_REAL("import math
lst = []
for i in _arg1:
    if math.isnan(i):
        lst.append(0)
    else :
        lst.append(math.log(i))
return lst",
SUM([Wt])
)
```



The calculation is valid.

3 Dependencies ▾

Default Table Calculation

Apply

OK

Calculation1 (copy)

X

Results are computed along Table (across).

```
SCRIPT_REAL("import math
lst = []
for i in _arg1:
    if math.isnan(i):
        lst.append(0)
    else :
        lst.append(math.log(i))
return lst",
SUM([Mpg])
)
```

The calculation is valid.

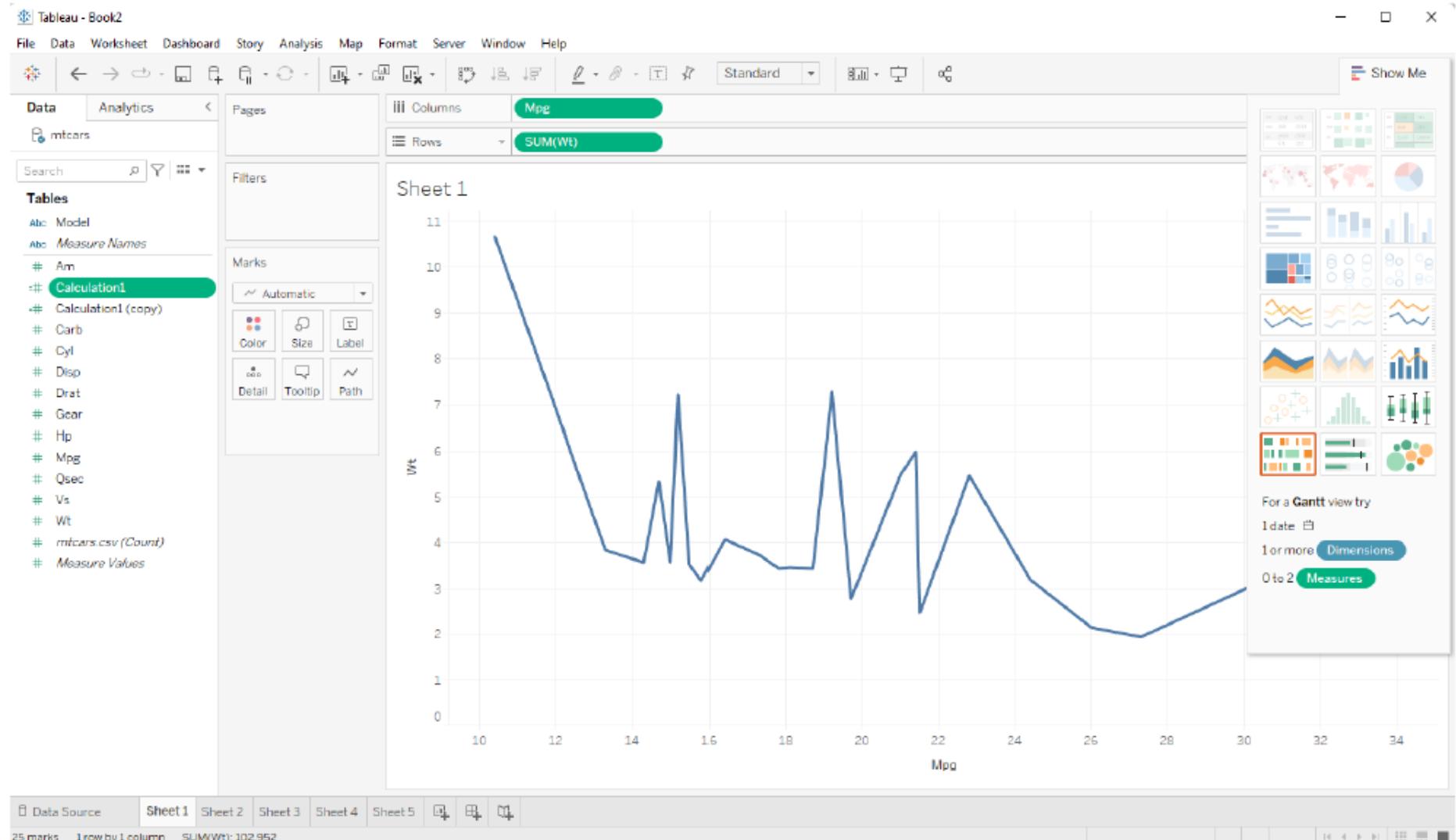
3 Dependencies ▾

Default Table Calculation

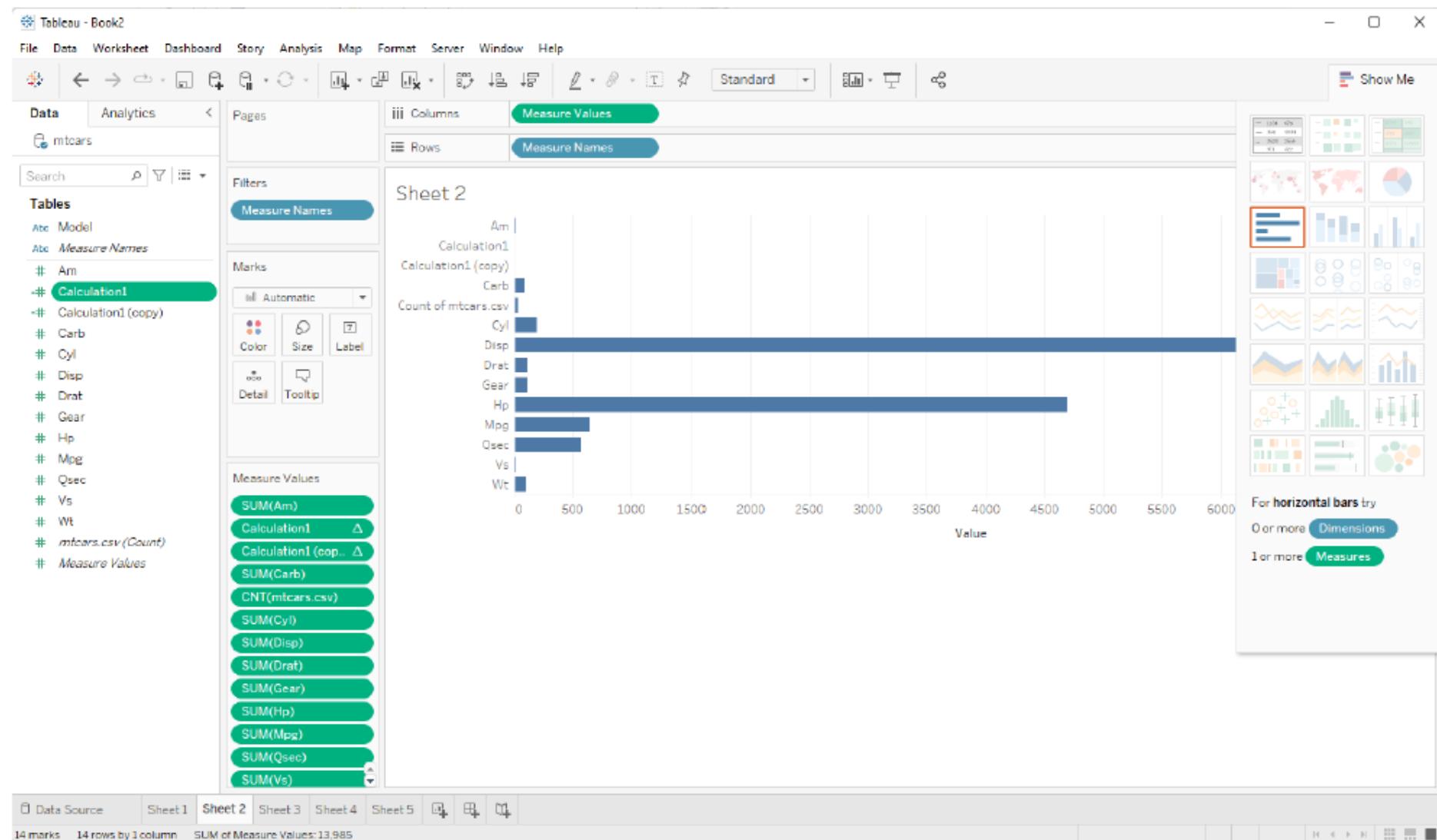
Apply

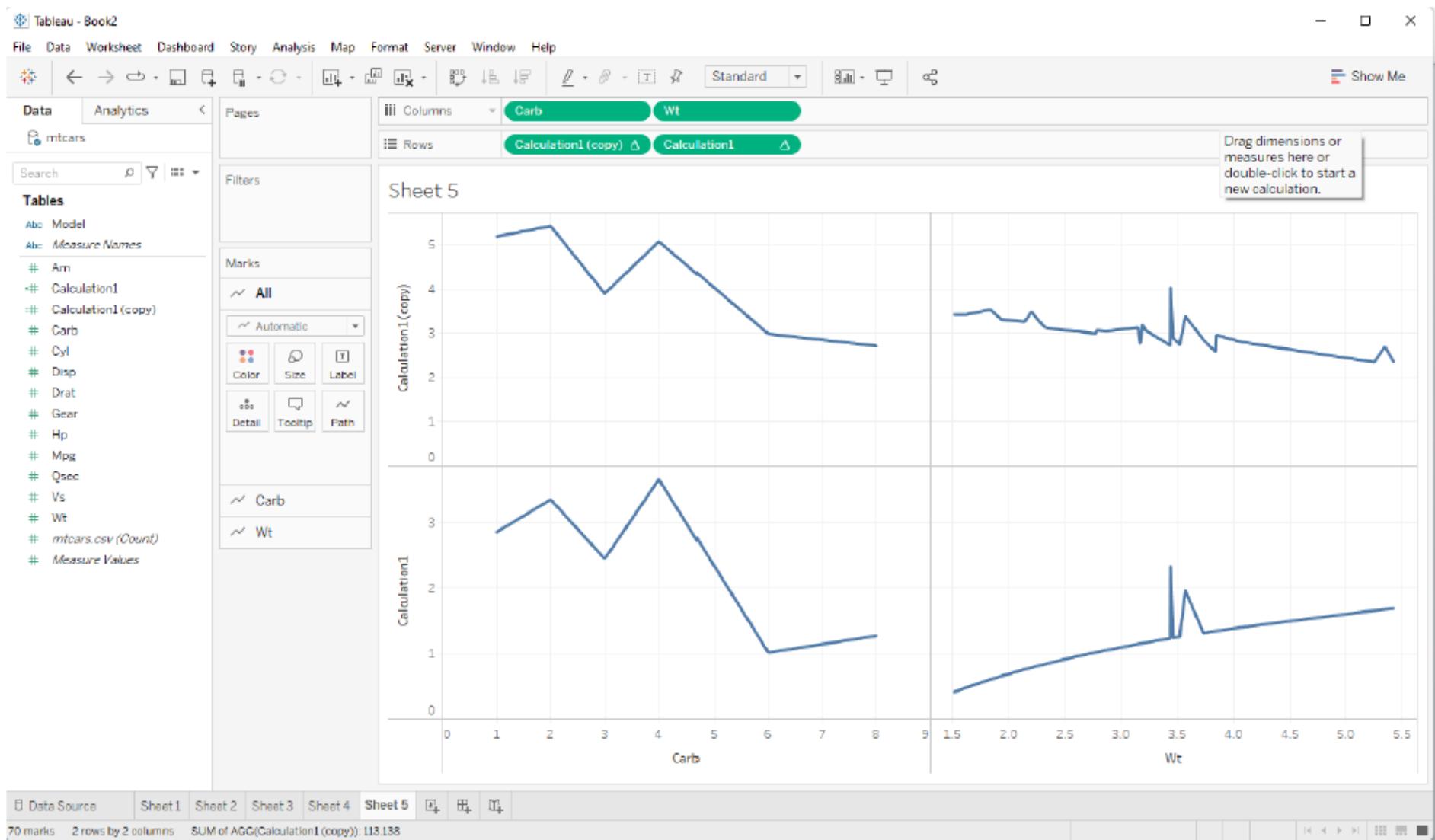
OK

Sheet 1:



Sheet2:





Case Study 7:

Explore the network visualization by considering the “Olympic Athletes” _dataset, using R to conclude the following using igraph packages and visualizations

- Different Countries with different medals and same country with different medals across all the years
- No. of models won by the same county with same model with same sports repeatedly

Explore the visual insight by Creating 5 Sheets with one Dashboard in tableau using the above dataset. Apply Tableau R integration to show the impact of specific sports with respect to specific country.

```

8 ````{r}
9 library(igraph)
10 ````

11 ````{r}
12 df= read.csv("C:/Users/admin/Downloads/OlympicAthletes_dataset.csv")
13 ````
14 ````{r}
15 df
16 ````
```

| Closing.Ceremony.Date | Sport | Gold.Medals | Silver.Medals | Bronze.Medals | Total.Medals |
|-----------------------|------------|-------------|---------------|---------------|--------------|
| 8/24/2008 | Swimming | 8 | 0 | 0 | 8 |
| 8/29/2004 | Swimming | 6 | 0 | 2 | 8 |
| 8/12/2012 | Swimming | 4 | 2 | 0 | 6 |
| 8/24/2008 | Swimming | 1 | 2 | 3 | 6 |
| 10/1/2000 | Gymnastics | 2 | 1 | 3 | 6 |
| 8/12/2012 | Swimming | 1 | 3 | 1 | 5 |
| 8/12/2012 | Swimming | 4 | 0 | 1 | 5 |
| 8/12/2012 | Swimming | 2 | 2 | 1 | 5 |
| 8/12/2012 | Swimming | 3 | 1 | 1 | 5 |
| 8/29/2004 | Swimming | 2 | 2 | 1 | 5 |

1-10 of 8,618 rows | 5-10 of 10 columns

Previous 1 2 3 4 5 6 ... 100 Next

```

17 ````{r}
18 # i
19 barplot(table(df$Country))
20 ````
21 ````
22 ````
```

```
```{r}
u= unique(df$Country)
```
```{r}
edge= c()
num= c()

for (x in u){

 temp= df[df$Country==x,]

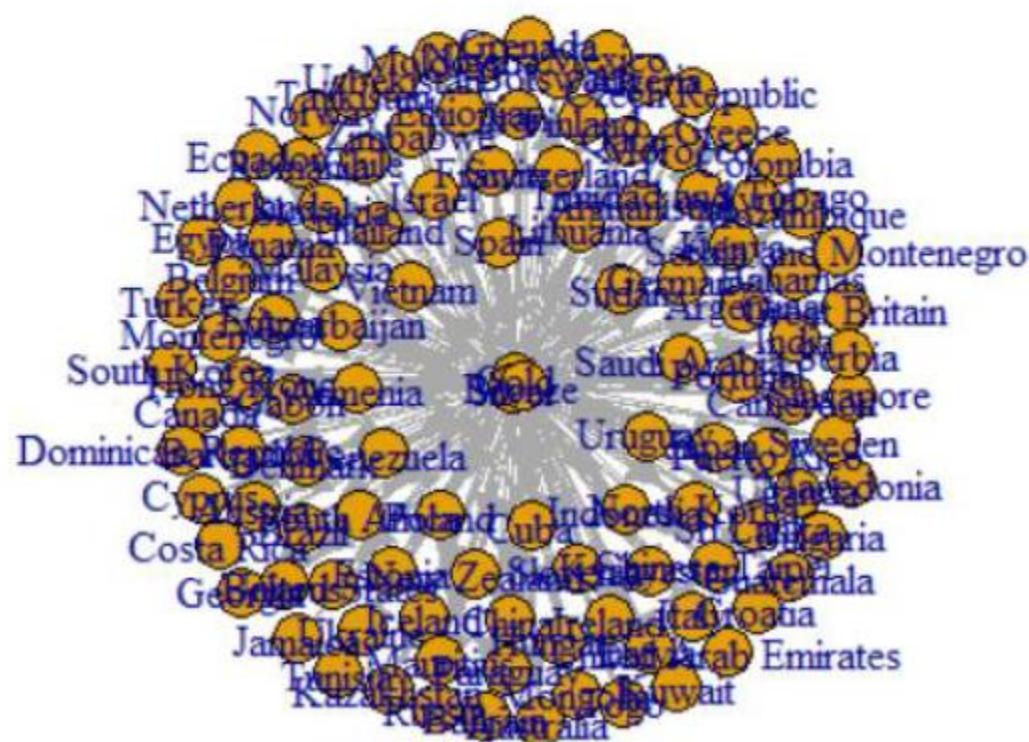
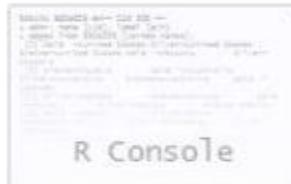
 edge= append(edge, "Gold")
 edge= append(edge, x)
 num= append(num, sum(temp$Gold.Medals))

 edge= append(edge, "Silver")
 edge= append(edge, x)
 num= append(num, sum(temp$Silver.Medals))

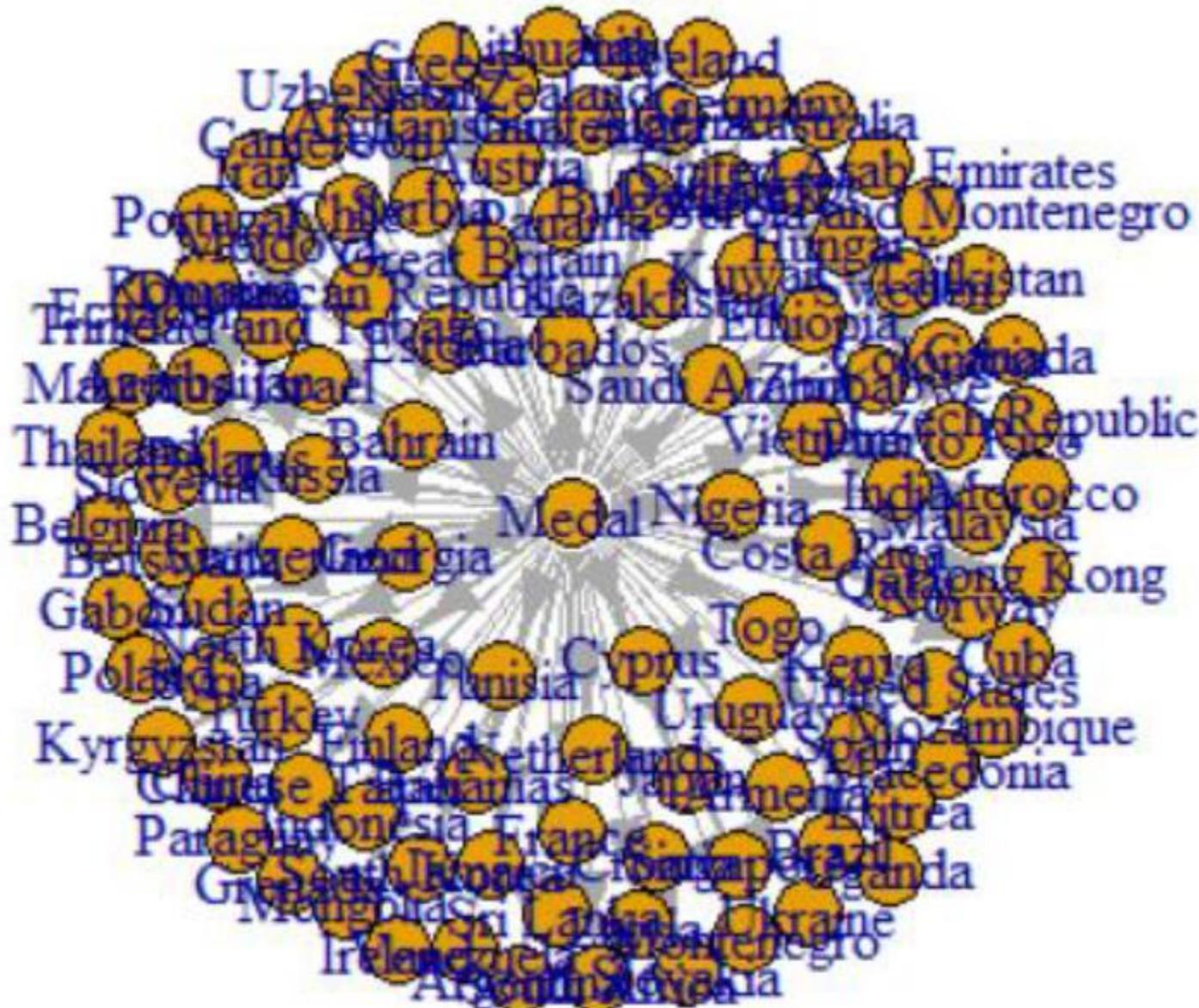
 edge= append(edge, "Bronze")
 edge= append(edge, x)
 num= append(num, sum(temp$Bronze.Medals))
}

```
```

```
49 ````{r}
50 g= graph(edges= edge)
51 
52 
53 
54 ````{r}
55 set_edge_attr(g, "label", value= num)
56 plot(g)
57 ````
```



(ii)



```
58 ````{r}
59 # ii
60
61 edge= c()
62 num= c()
63
64 for (x in u){
65
66   temp= df[df$Country==x, ]
67
68   edge= append(edge, "Medal")
69   edge= append(edge, x)
70   num= append(num,sum(temp$Total.Medals))|
71 }
72
73 ````{r}
74 ````{r}
75 g2= graph(edges= edge)
76
77
78
79 ````{r}
80 set_edge_attr(g2, "label", value= num)
81 plot(g2)
82
```

Clustering



Results are computed along Table (across).

```
SCRIPT_INT('
  km <- kmeans(data.frame(.arg1, .arg2, .arg3), 3);
  km$cluster;
  ',
  SUM([Bronze Medals]), SUM([Silver Medals]), SUM([Gold Medals])
)|
```

[Default Table Calculation](#)

The calculation is valid.

1 Dependency ▾

[Apply](#)

[OK](#)

Manage Analytics Extensions Connection

X



New RServer Connection

Require SSL

Hostname

localhost

Port

6311

Action Completed

X



Successfully connected to the analytics extension.

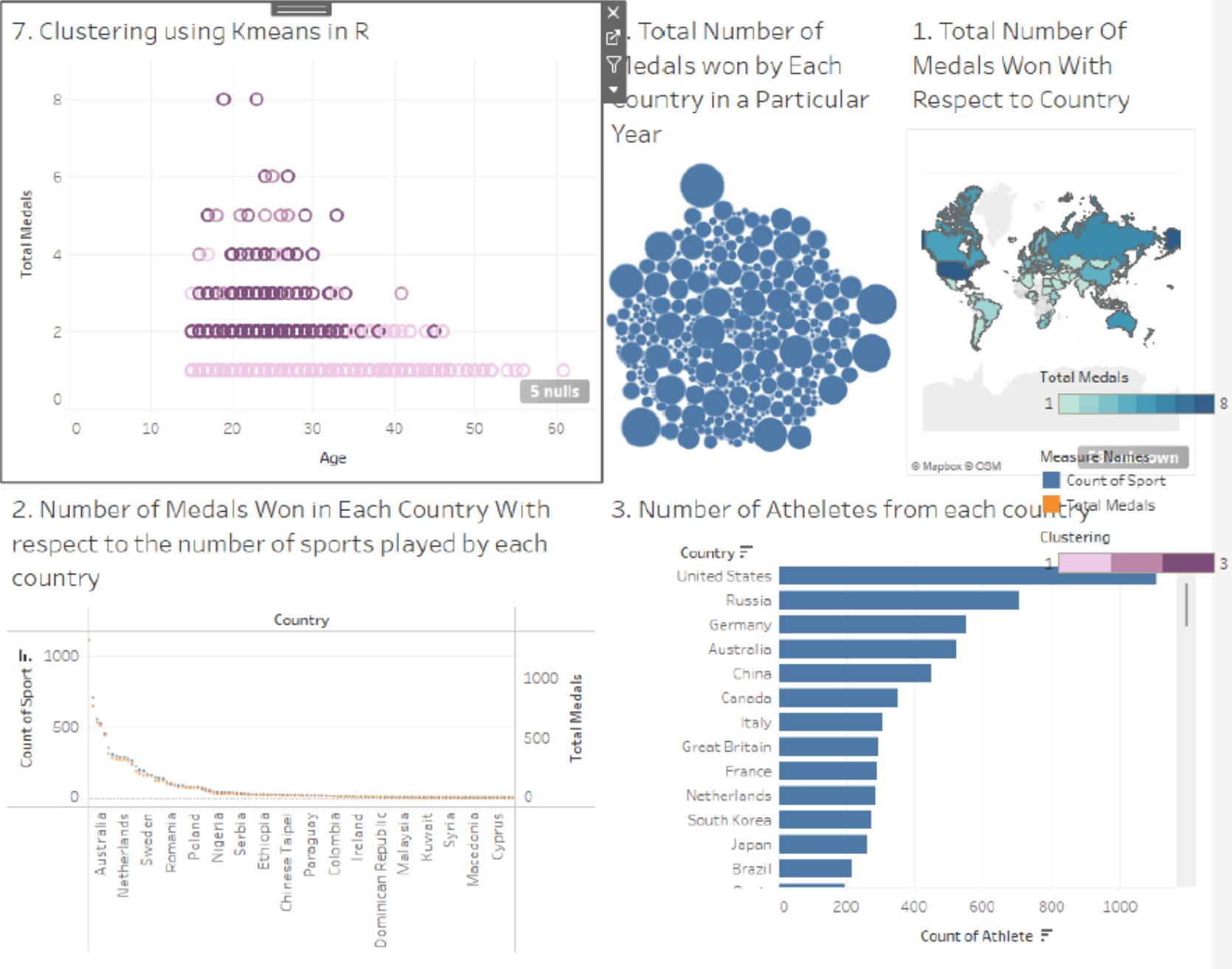
OK

Test Connection

Back

Save

Dashboard:



Chapter 6

Knime Analytics Platform

Data Visualization with Knime Analytics Platform

Introduction

KNIME Analytics Platform is an open-source software for working on the analytical aspect of data which helps in understanding data and different machine learning or data science related operations can be done easily on those data without coding. Using KNIME Analytics Platform we can make the virtual flow or workflows with the help of the nodes made available by the software using a very easy, open, intuitive,perceptive, drag and drop style of user interface.

Key Features of KNIME TOOL are –

It is an intuitive and open platform for integrating new developments. This makes its reusable components easily accessible to everyone.

Creating visual workflows without the need of coding and dragging- and dropping style has made it far known to a large part of the audience.

It has a wide number of nodes which makes it easy to model each and every step of the data analysis performed.Knime users can control the flow of their data and can continuously change it.

In Knime Software you can not only work with a variety of different nodes, you can also blend a number of tools from various domains in a single workflow like scripting in [R](#) & Python, [machine learning](#), or connectors to [Apache Spark](#).

Knime Hub has already existing workflows which are publicly available. These can be used as tutorials for the new users to provide the overview of the platform and make it understandable for them.

Knime fit in with several other open-source platforms, eg.- machine learning algorithms from [Weka](#), [Keras](#), [Spark](#), the [R project](#) and [LIBSVM](#); as well as [JFreeChart](#), [ImageJ](#), and the [Chemistry Development Kit](#).

Blending of data from sources to open and combine various text formats and unstructured data types. This also helps in connecting the host to different data bases and data warehouses to integrate access and retrieve data from a number of sources.

Knime helps its users to perform and modify their data in a number of ways.This includes transforming the raw data into classified, aggregated, sorted and filtered data.

Performing mathematical functions and applying statistical methods to your data is also possible with knime. This makes our model to be tested and verified for hypothesis.

Knime not only helps us in performing various tasks. It also helps us in detecting and rectifying the errors in our model with the help of outlier and anomaly detection algorithms.

Cleaning and extraction of our data is comparatively a lot easier with the help of Knime software. The datasets available can be further prepared for Machine Learning with genetic algorithms.

Building Machine Learning models with Knime Software helps in a lot of ways. We can optimize the model performance, validate models along with explaining the models at the same time in the same workflow.

Visualizing the data or the result obtained from the analysis can be done from the vast array of classic as well as advanced charts that can be further customized.

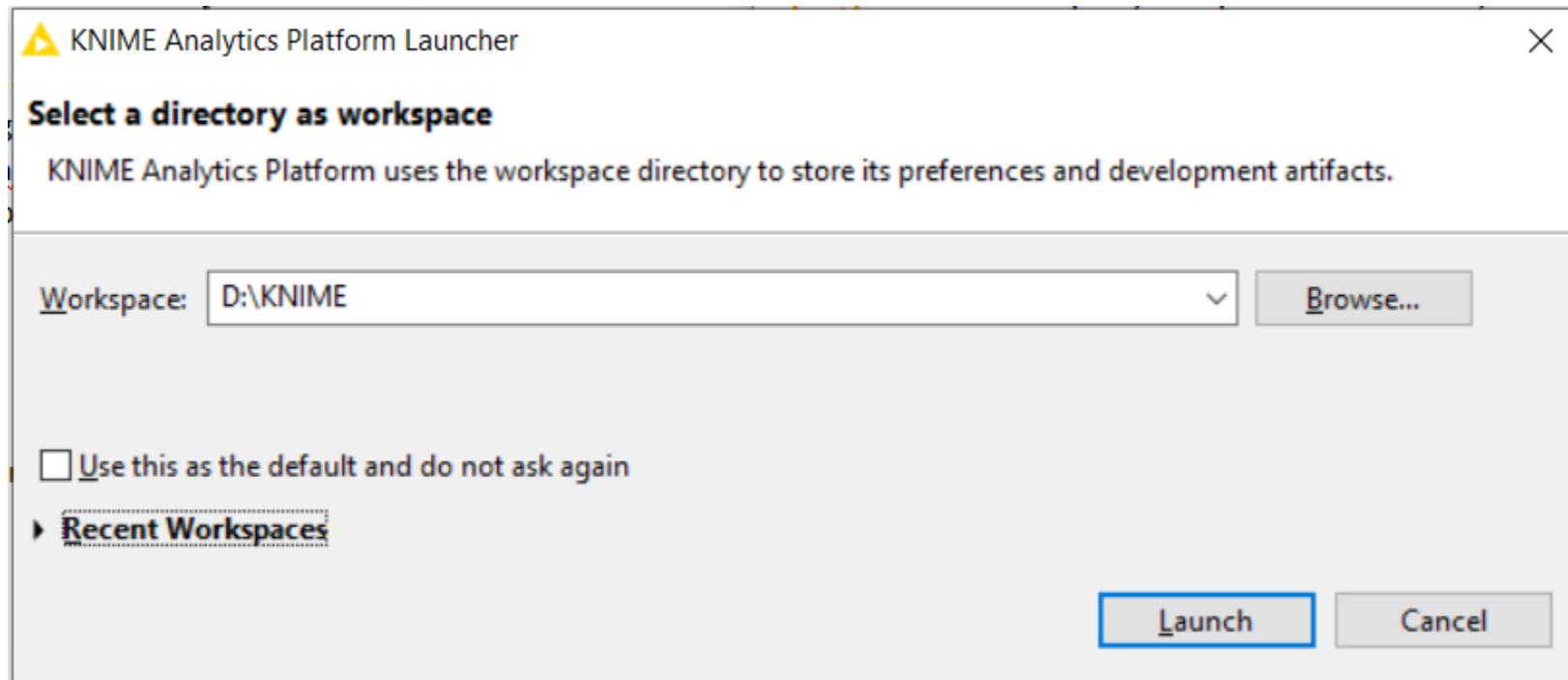
Displaying the summary of the model can also be done using Knime software. Users can also export reports in various formats. It also provides us with options to store our processed data or results in file formats.

Knime provides us the platform to build workflow prototypes to discover, inspect and save intermediate results for quick feedbacks and recovery to create better solutions.

With the help of Knime software we can Scale workflow performance and enhance the power of in-database processing to increase the level of our obtained results.

Start KNIME Analytics Platform :-

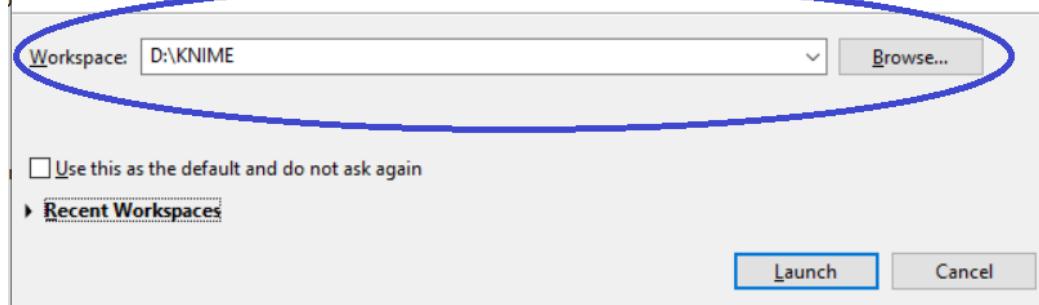
When we start KNIME Analytics Platform “KNIME Analytics Platform Launcher” window appears



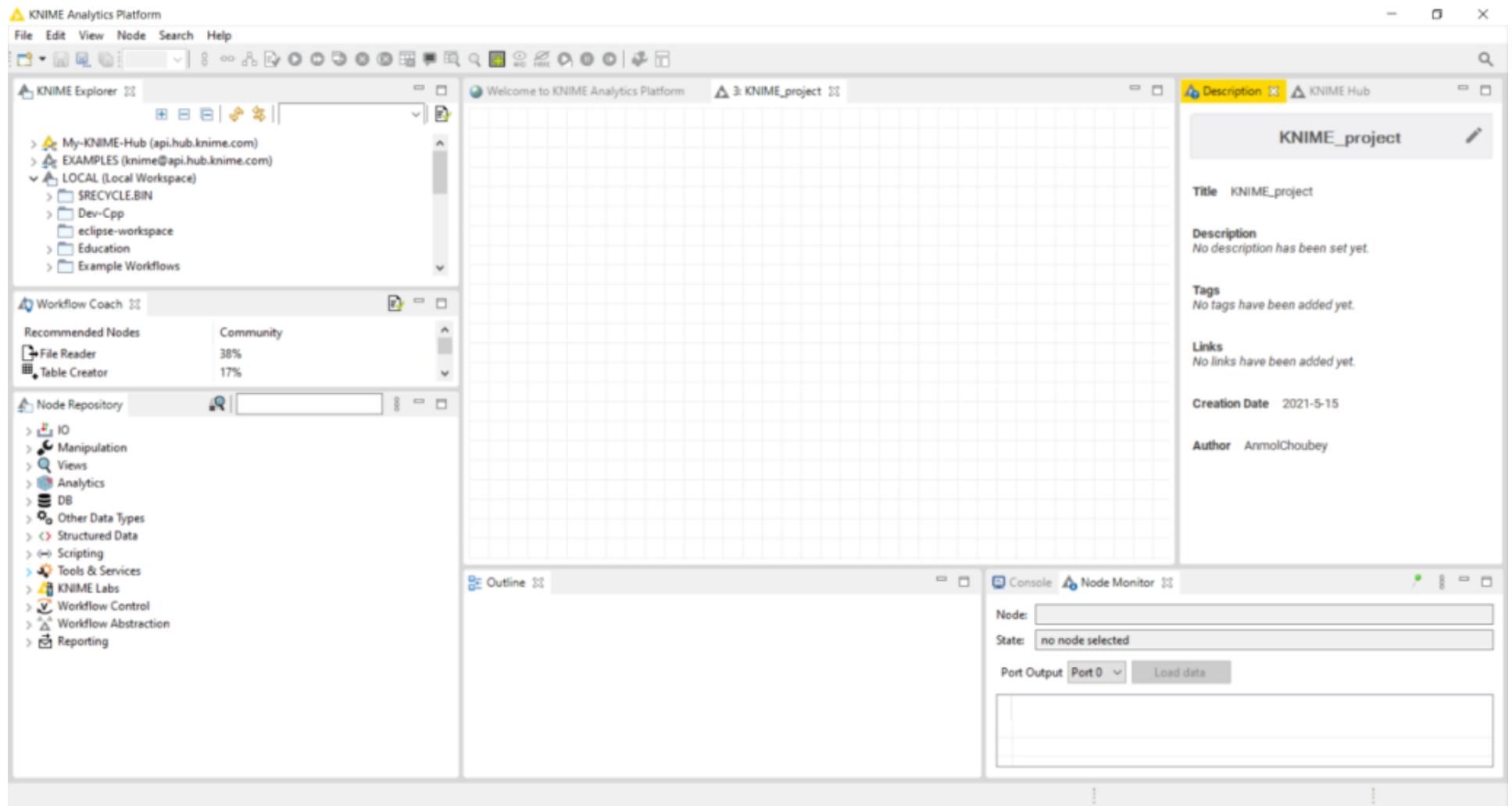
KNIME Analytics Platform uses this workspace address or directory available in users local computer which is stated or given by the user to store its preference and development artifacts which includes node settings, workflows, data produced by the flow, in short user work will be saved to this directory or in KNIME words to this workspace.

Select a directory as workspace

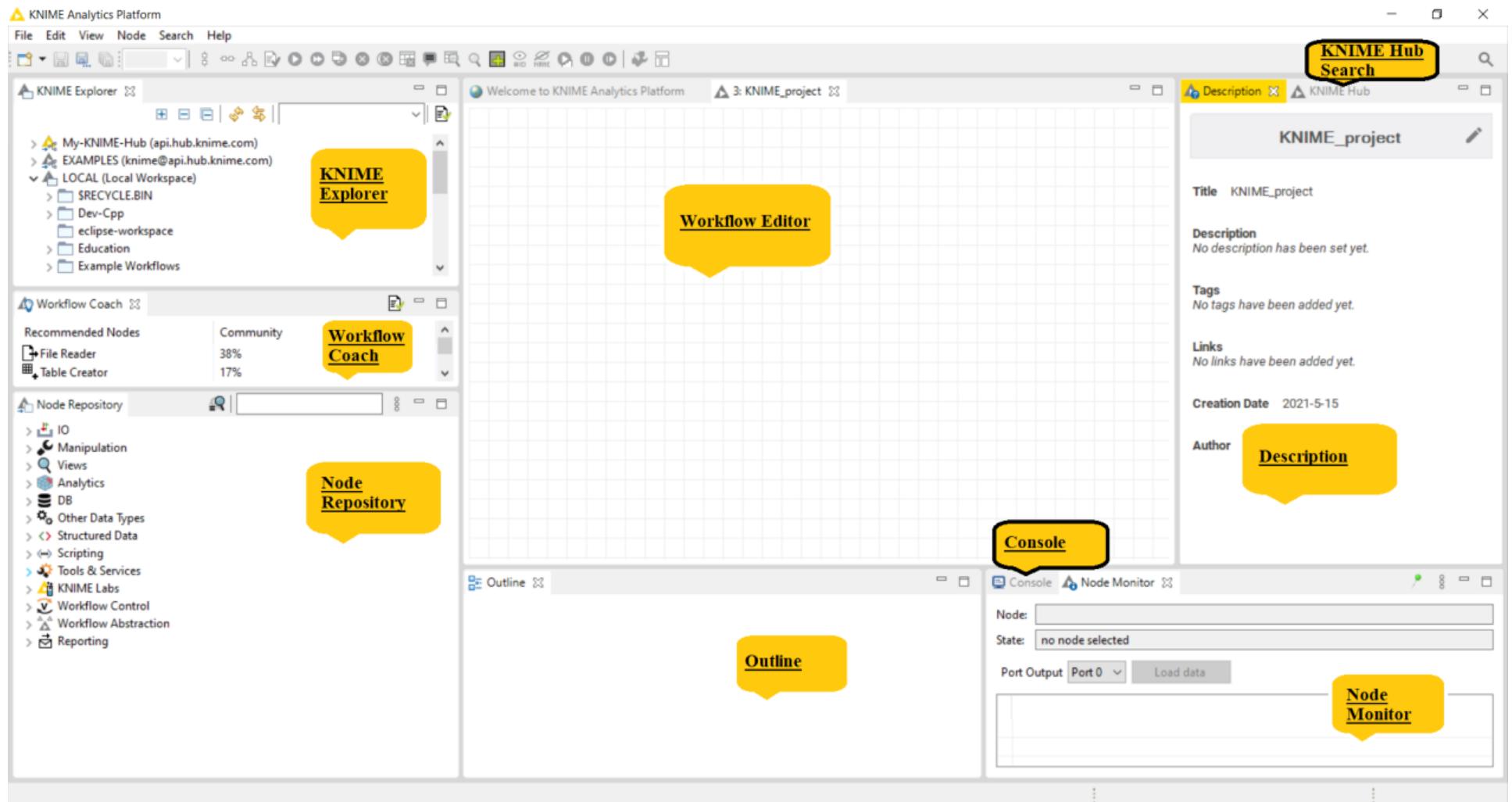
KNIME Analytics Platform uses the workspace directory to store its preferences and development artifacts.



After selecting the desired directory or folder click “Launch” , it will open the workbench where all the hard work is done. Lets move forward and try to get familiar with the components of the KNIME workbench and its interface.

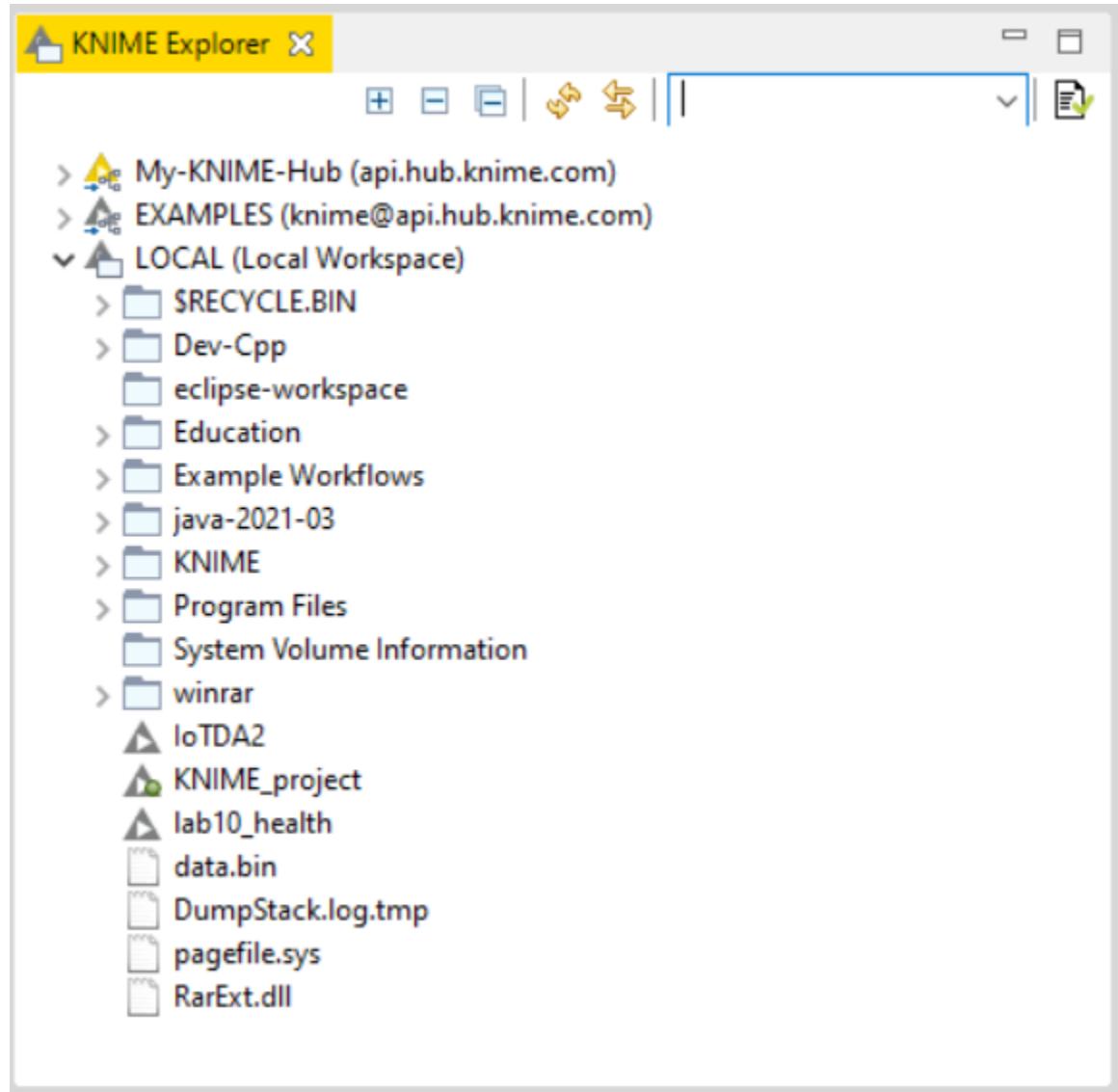


This is how the workbench looks like after clicking on “Launch”. Lets talk denote the components of the KNIME workbench.



The KNIME Workbench consists of following components:-

KNIME Explorer :- It allows the user to browse their workflows or workflow groups which includes user's local workspace, KNIME Servers and personal KNIME Hub space thus enables the user to share workflows and collaborate with colleagues using shared resources. In KNIME Explorer user can mount multiple repositories at the same time allowing user to work on workflows from different repositories simultaneously and to copy or move workflows from one repository to another.



In order to add more content to the view, click “configuration”  icon on the right top corner of KNIME Explorer window.

type filter text

KNIME

KNIME Explorer

KNIME Explorer

Setup mount points for usage in KNIME Explorer view.

List of configured mount points:

| MountID | Mounted Content | Mounted Type |
|--|---|--|
| <input checked="" type="checkbox"/> My-KNIME-Hub |  api.hub.knime.com |  KNIME Community Hub |
| <input checked="" type="checkbox"/> EXAMPLES |  knime@api.hub.knime.com |  KNIME Example Server |
| <input checked="" type="checkbox"/> LOCAL |  Local Workspace |  Local Workspace |

New...
Edit...
Remove
Up
Down

Link components when sharing on Server or Local Workspace Prompt

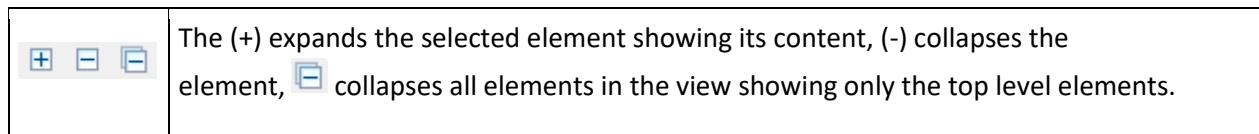
Show a warning dialog when connecting to a server via EJB

Restore Defaults Apply

Apply and Close Cancel

We will talk about the different function available in it further.

Explorer Toolbar



| | |
|---|---|
|  | Refreshes the view, in case it is out-of-sync with the underlying file system. |
|  | If a workflow located in the Team Space is shown in an editor, this workflow is selected in the Team Space view. |
| Filter | If you add text to the field and press Enter , the Explorer will filter to items that contain the text in their name or are in a group containing the text in its name. |
|  | Opens the Explorer preference page, allowing you to select the content displayed in the view or to add/remove mount points. |

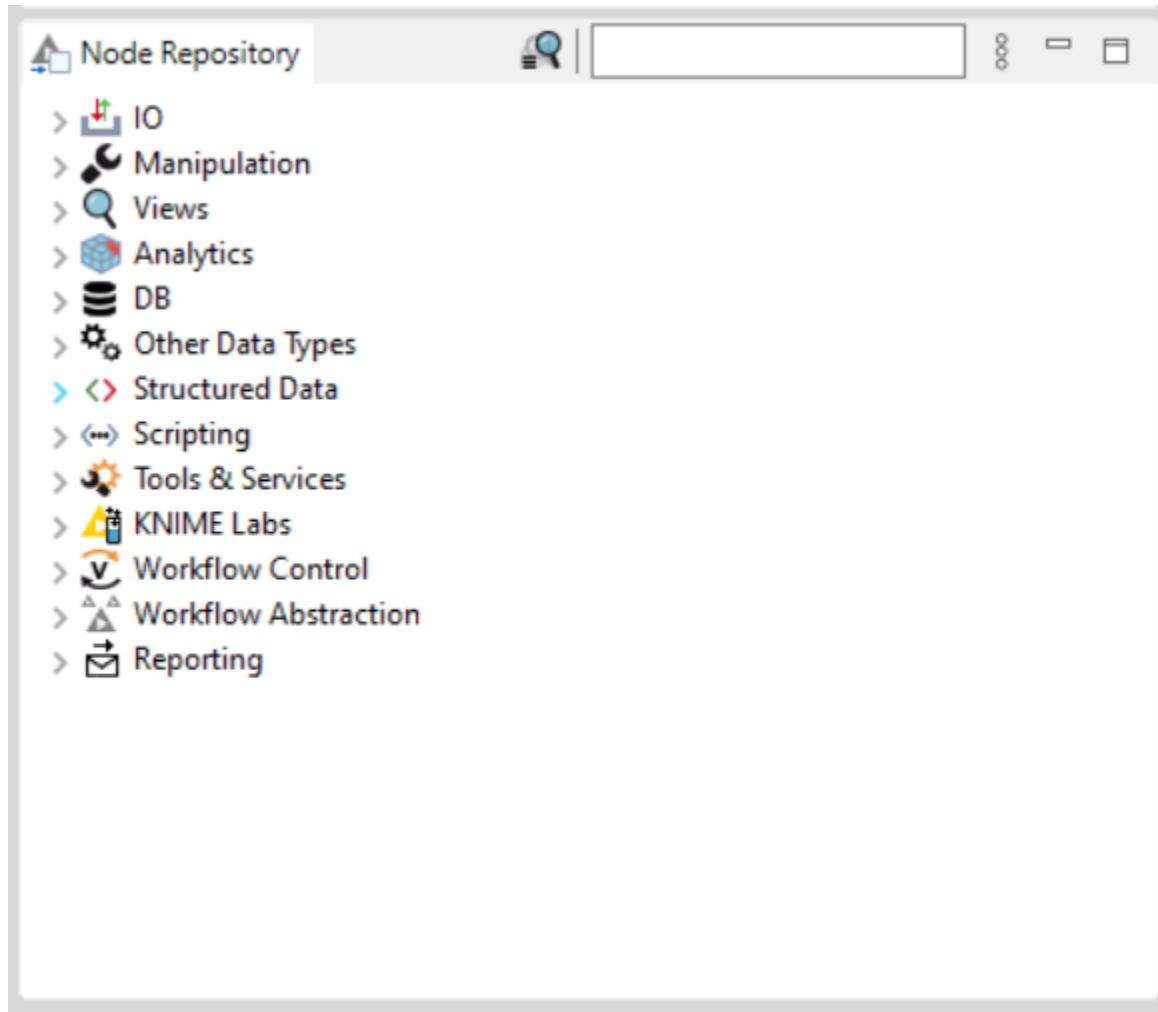
Workflow Coach :- Lists node recommendations based on the workflows built by the wide community of KNIME users. It is inactive if the user don't allow KNIME to collect user's usage statistics. If the user selects a node in the workflow editor, the workflow coach shows the most popular nodes to follow the selected node. The recommendations are based on KNIME community usage statistics about workflows built in KNIME Analytics Platform. User can drag and drop or double click to add nodes in workflow editor.

Workflow Coach

| Recommended Nodes | Community |
|----------------------------------|-----------|
| File Reader | 38% |
| Table Creator | 17% |
| Database Reader (legacy) | 8% |
| Table Reader | 6% |
| Database Connection Table Rea... | 3% |
| DB Reader | 2% |
| Database Connector (legacy) | 2% |
| Database Table Selector (legacy) | 2% |
| DB Query Reader | 2% |
| DB Table Selector | 2% |
| Create Date&Time Range | 2% |
| ARFF Reader | 2% |
| Database Table Connector (leg... | 1% |
| MySQL Connector (legacy) | 1% |
| Data Generator | 1% |
| XML Reader | 1% |
| JSON Reader | <1% |
| Microsoft SQL Server Connecto... | <1% |
| DB Connector | <1% |
| PostgreSQL Connector (legacy) | <1% |
| SQLite Connector (legacy) | <1% |

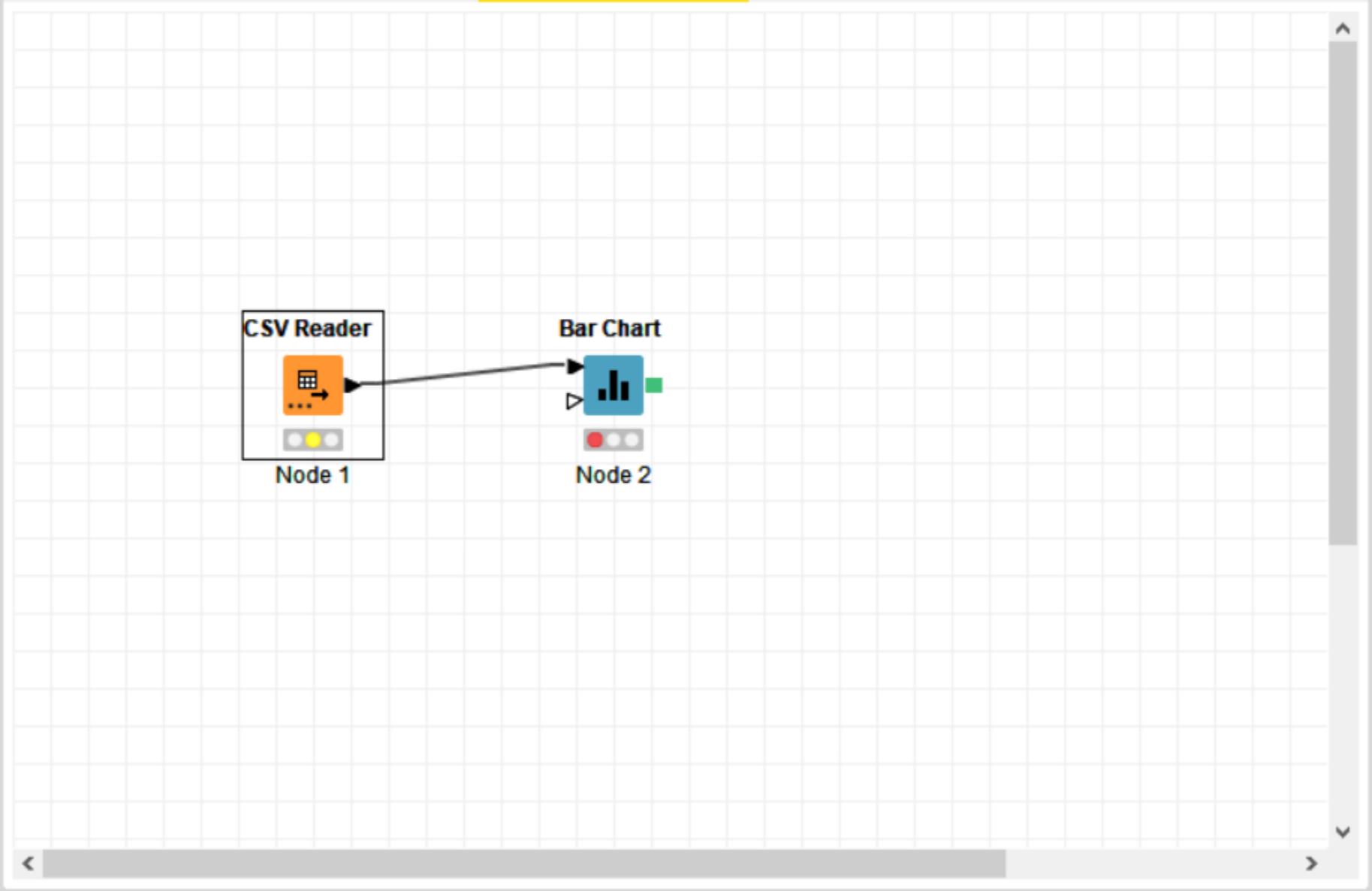
Node Repository:- All nodes available in core KNIME Analytics Platform and in the extensions you have installed are listed here. The nodes are organised by categories but user can also use the search box on the top of the node repository to find nodes.

Search Box



User can add a node from node repository by drag and drop, or by double click. The default search mode is “crisp search”, the interface will return all the nodes that either have the search term in their names, or they are in a subcategory whose name include the search term. We can switch the search mode to “fuzzy search”, the interface will return all nodes that are related to the search term.

Workflow Editor:- It is the canvas where you work or edit the currently active workflow. Workflows are assembled here.



Nodes are dragged and dropped in the workflow editor from the Node Repository then we create the workflow by connecting, configuring and executing them.

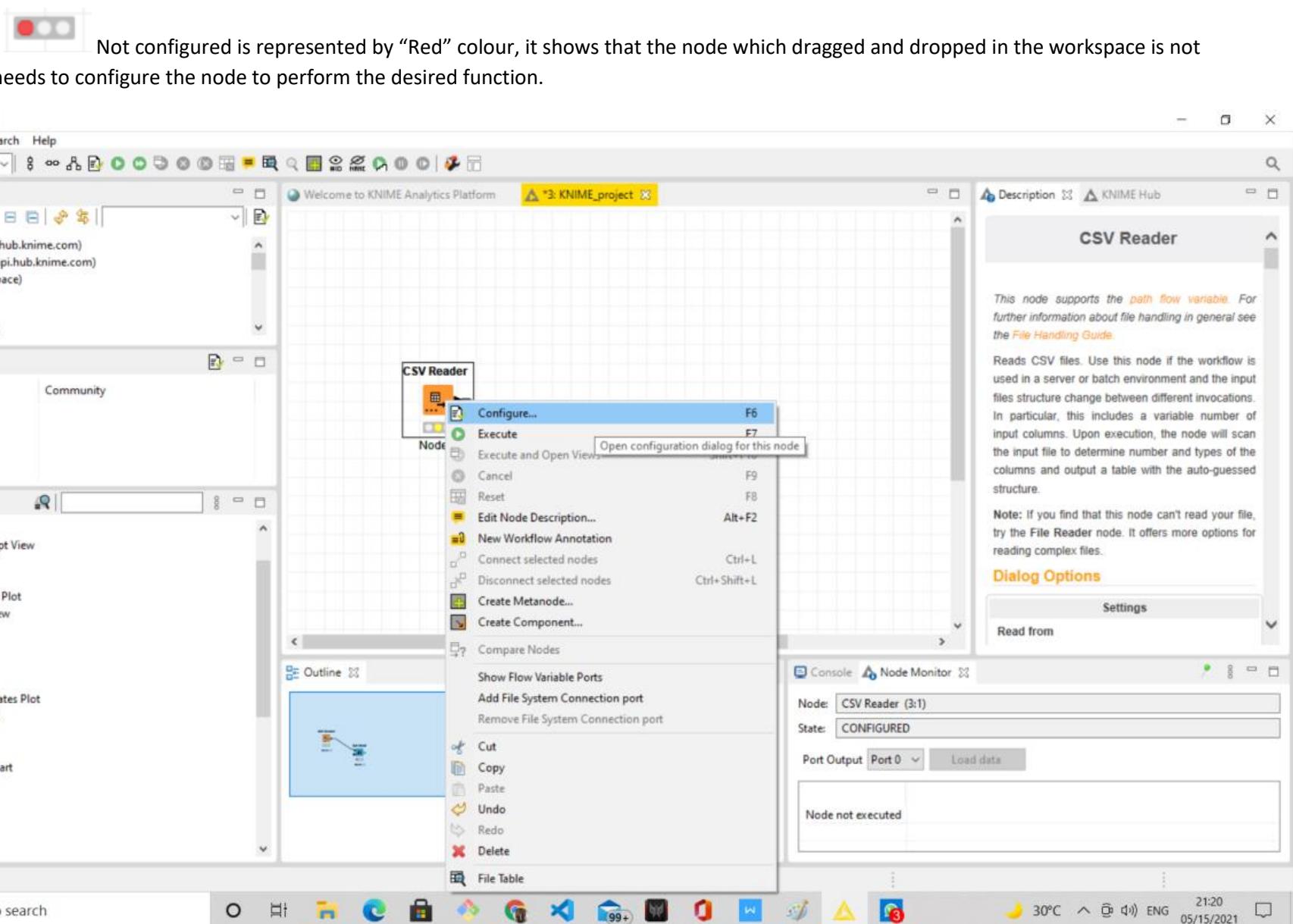
Lets talk a little bit about the node status, there are four node status :-

Not configured

Configured

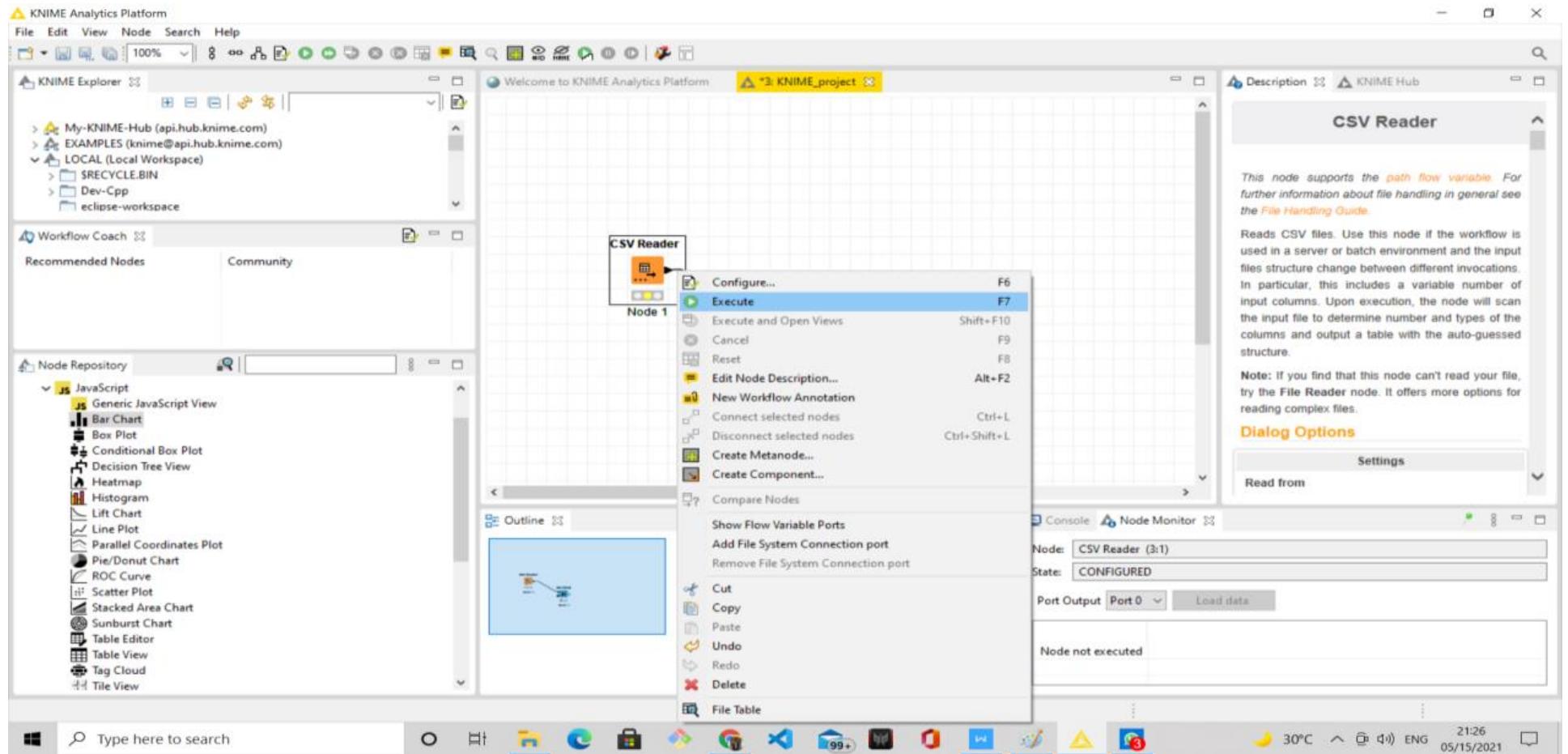
Executed

Error



After configuring the node, its status changes to "Configured" which is of yellow colour.

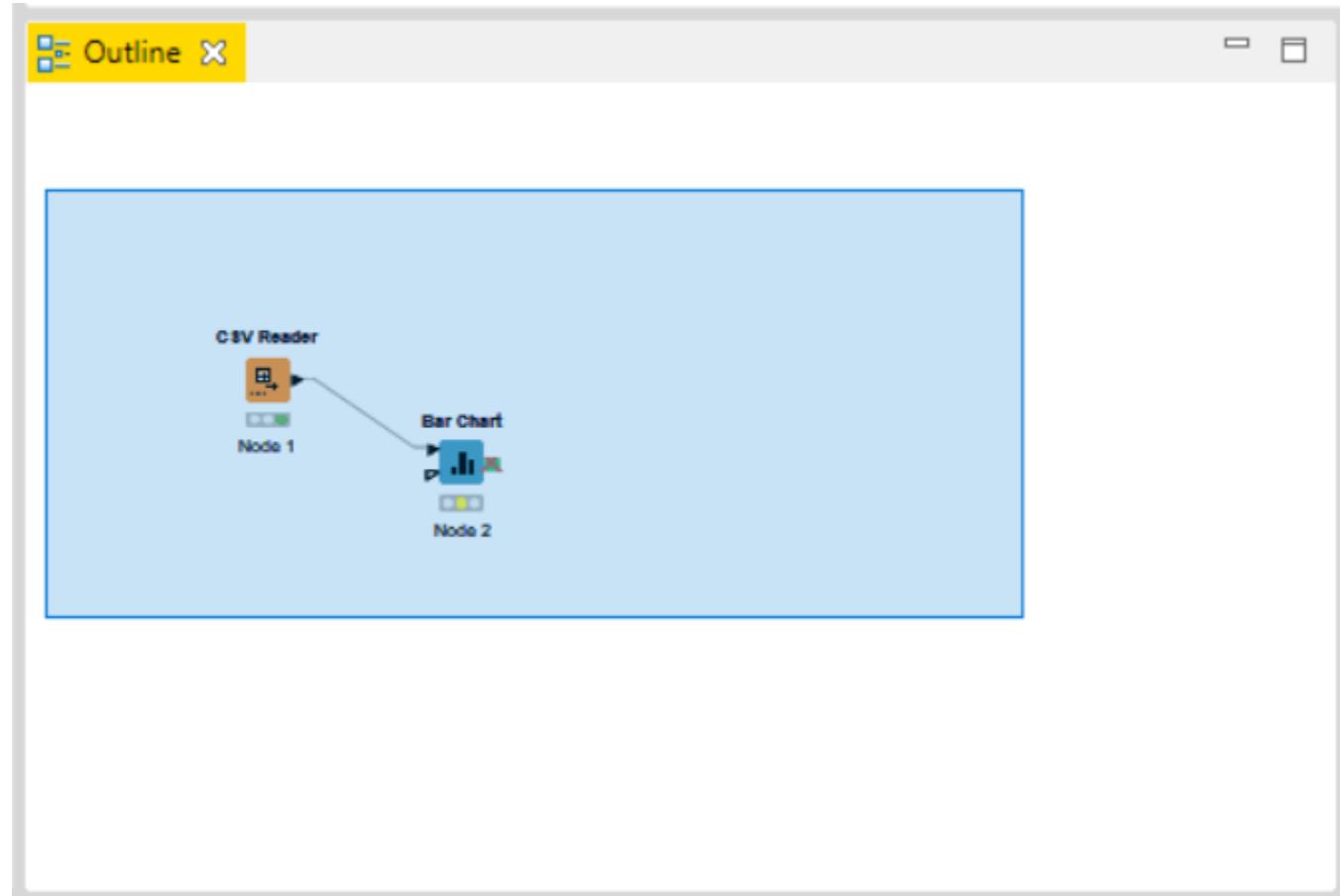
Configured :- Configured is represented by "Yellow" colour, it depicts that the node is configured and ready to be executed. We need to click "Execute" option after Right click on the node to change the status of the node to Executed.



Executed: Executed status is represented by “Green” colour, it depicts that the node is successfully executed. If the execution fails, user will see an error sign which shows “Error” status.

Error: Error status is represented with a cross inside a red circle it depicts that the node has not executed properly or there is an error while configuring it.

Outline :- User can see an overview of the currently active workflow. Lets consider the workflow does not fit in the workflow edit, user can change the active area by positioning the blue, transparent rectangle.



Console:- Console shows all the warning and error messages if there with the nodes in the workflow execution, in short is shows the execution messages taking place under the hood.

Console Node Monitor

KNIME Console

```
*****
***      Welcome to KNIME Analytics Platform v4.3.2.v202103051236 ***
***      Copyright by KNIME AG, Zurich, Switzerland ***
*****
```

Log file is located at: D:\.metadata\knime\knime.log

ERROR CSV Reader 3:1 Writing of table to file at node CSV Reader 3:1 at workflow KNIME_project 3 encountered error:
 ERROR CSV Reader 3:1 Table will be held in memory until node is cleared.
 ERROR CSV Reader 3:1 Workflow can't be saved in this state.
 ERROR Bar Chart 3:2 Execute failed: No column selected for category values.
 ERROR Histogram 3:2 Execute failed: No column selected for binning.

Node Monitor:- Node monitor allows the user to inspect intermediate output tables in their workflow. There many other functions available like Show Variables, Show Configuration, etc which can be performed in Node Monitor

Console Node Monitor

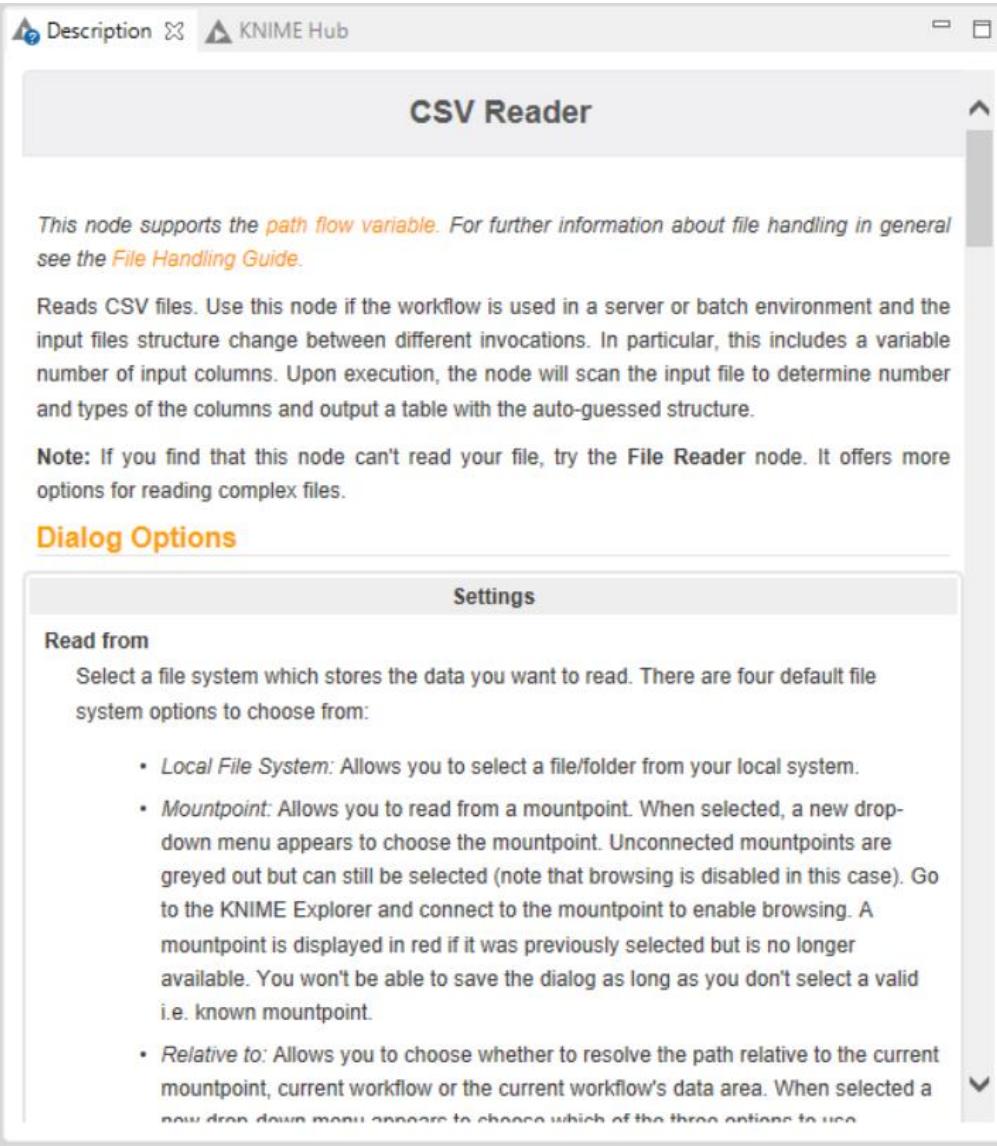
Node: CSV Reader (3:1)

State: EXECUTED

Port Output Port 0 ▾ Load data Rows: 275, Columns: 475

| ID | Province/State | Country/Region | Lat | Long | 22/01/2020 | 23/01/2020 | 24/01/2020 | 25/01/2020 | 26/01/2020 | 27/01/2020 | 28/01/2020 |
|-------|------------------------------|----------------|----------|----------|------------|------------|------------|------------|------------|------------|------------|
| Row6 | ? | Argentina | -38.4161 | -63.6167 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Row7 | ? | Armenia | 40.0691 | 45.0382 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Row8 | Australian Capital Territory | Australia | -35.4735 | 149.0124 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Row9 | New South Wales | Australia | -33.8688 | 151.2093 | 0 | 0 | 0 | 0 | 3 | 4 | 4 |
| Ro... | Northern Territory | Australia | -12.4634 | 130.8456 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ro... | Queensland | Australia | -27.4698 | 153.0251 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ro... | South Australia | Australia | -34.9285 | 138.6007 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ro... | Tasmania | Australia | -42.8821 | 147.3272 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ro... | Victoria | Australia | -37.8136 | 144.9631 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Description :- Description of the selected node or the workflow or the workspace is represented here.



The above image is of "Description" window listing the description of the selected node i.e. CSV READER.

KNIME Hub :- KNIME Hub gives the online support or help or guide if user needs with respect to workflows, nodes and more..

The screenshot shows the KNIME Hub interface. At the top, there are two tabs: 'Description' and 'KNIME Hub', with 'KNIME Hub' being the active tab. Below the tabs are four circular icons: a house (Home), a left arrow (Back), a right arrow (Forward), and a 'C' (Copy). To the right of these icons is a button labeled 'Open in browser'. A search bar below the icons contains the placeholder text 'Search workflows, nodes and more...'. The main content area features a large heading 'Welcome to the **KNIME Hub**'. Below the heading, a subtext reads 'Solutions for data science: find workflows, nodes and components, collaborate in spaces.' At the bottom of the main area, there is a light gray box containing three large numbers: '4 011' under 'Nodes', '515' under 'Components', and '4 711' under 'Workflows'. Each number is preceded by a horizontal line.

CASE STUDIES: REAL TIME APPLICATIONS

Case Studies 1: Fitness Tracker with Calories Burn Prediction Using Knime Analytics Platform

This exercise aims to perform calorie burn prediction using a random forest predictor through the Knime analytics platform.

List of Nodes Required:

| |
|--------------------------------|
| • Column Filter Node |
| • Missing Value Node |
| • Rule Engine Node |
| • X- Partitioner Node |
| • X-Aggregator Node |
| • Linear Correlation Node |
| • Random Forest Learner Node |
| • Random Forest Predictor Node |
| • Box Plot Node |
| • Scorer Node |
| • Bar Chart Node |

Procedure to build Fitness Tracker flow:

- Download the “FitBit Fitness tracker data” dataset from the following link: <https://www.kaggle.com/arashnic/fitbit>.
- This data set consists of 15 column attributes namely: Id, Activity date, Total steps, Total distance, Logged activity distance, Very active distance, Moderately active distance, Light active distance, Sedentary active distance, Very active minutes, Fairly active minutes, Lightly active minutes, Sedentary minutes and Calories. All of the first 14 columns determine the individual’s overall calories burnt in a day. With this dataset, we can identify if an individual has a high, low, or good calorie bum for the day.
- Drag and drop the CSV file in the Knime platform to get the CSV reader node.
- Connect this with the Column filter and Missing value node for preprocessing the data. In the column filter, exclude the ActivityDate column, as the date doesn’t matter for this prediction. In the missing value node, enter all the missing values as the mean of the entire column.
- Connect the Missing value node with the Rule Engine node. Here, give the rules as follows:
 - If Calories > 2500 => High calories bum
 - If Calories < 2000 => Low calories burn
 - If Calories ≥ 2000 and Calories ≤ 2500 => Good calories bum Append this data into a new column named “Calories bum”.
- Connect this with the box plot, linear correlation and X-partitioner node.
- In the box plot, exclude all the columns except Calories and visualize the plot. We can visualize the robust statistical parameters of minimum, lower quartile, median, upper quartile, and maximum value.
- The linear correlation node calculates for each pair of selected columns a correlation coefficient, i.e. a measure of the correlation of the two variables. Here, all columns are included to calculate the correlation coefficient.

- The X-Partitioner node is used for performing the cross-validation in the dataset. This node is the first in a cross-validation loop. At the end of the loop, there must be an X-Aggregator to collect the results from each iteration. All nodes in between these two nodes are executed as many times as iterations should be performed. We perform 10 validations using random sampling and the best out of these 10 is given as the output.
- Connect the 1st partition with the Random forest learner node. Select the Target column as calories burn and let the rest of the options be the same as the default. Select the split criteria as Information Gain Ratio.
- Now, connect the Random forest predictor node with 2nd partitioning and the Random forest learner node.
- Connect this node with the X-Aggregator node. This node collects the result from a predictor node, compares the predicted class and real class, and outputs the predictions for all rows and the iteration statistics. Select the target column as Calories burn and the Prediction Column as Prediction (Calories burn).
- Connect this node with the scorer and bar chart node.
- In the scorer node, select the first column as Calories bum and the second column as Prediction (Calories bum) and save it.
- In the bar chart, enter the column as prediction (calories burn) and name the chart, then save it. Finally, execute all the nodes and view the output.

COMPLETE FLOW:

The complete flow to perform calorie bum prediction using a random forest predictor is presented in figure 1.

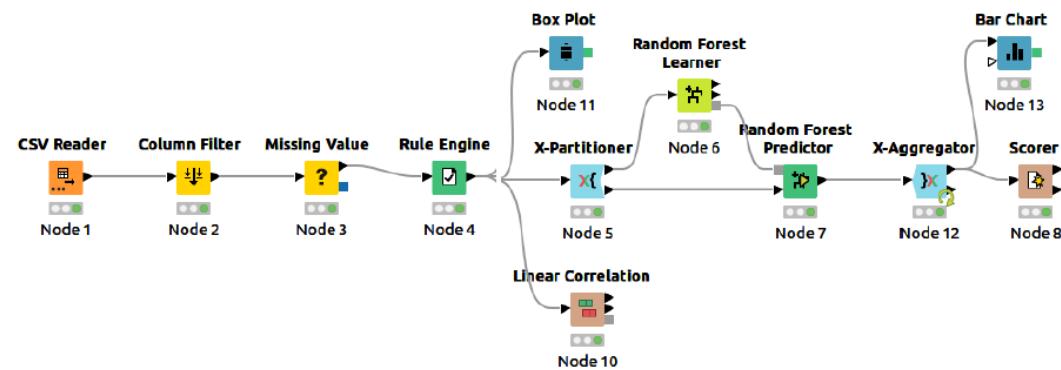


Figure 1: Knime Flow for Calorie Bum Prediction Using A Random Forest Predictor

Node Configurations:

- The configuration window of every node is presented in this section.
- CSVREADER:**

CSV READER :

The screenshot shows the configuration interface for a CSV Reader node in KNIME. The top navigation bar includes tabs for Settings, Transformation, Advanced Settings, Limit Rows, Encoding, Flow Variables, and Memory Policy. The 'Input location' section is active, showing 'Read from Custom/KNIME URL' with a timeout of 1,000 ms. A warning message states that Custom/KNIME URL does not support listing/browsing of files. The 'Mode' is set to 'File'. The 'URL' field contains 'file:/home/faerie-mattins/Downloads/archive/Fitabase%20Data%204.12.16-5.12.16/dailyA'. The 'Reader options' section includes 'Format' settings for column delimiter (,), row delimiter (Line break or \r\n), quote character ("), quote escape character (\"), and comment character (#). It also includes checkboxes for 'Has column header' (checked), 'Has row ID' (unchecked), 'Support short data rows' (unchecked), and 'Prepend file index to row ID' (unchecked). The 'Preview' section displays a table with 6 rows and 13 columns, showing data for rows Row0 through Row5. The columns are labeled: Row ID, Id, Activit..., TotalS..., TotalD..., Track..., Logge..., VeryA..., Moder..., LightA..., and Seden... .

| Row ID | Id | Activit... | TotalS... | TotalD... | Track... | Logge... | VeryA... | Moder... | LightA... | Seden... |
|--------|-------------|------------|-----------|-----------|----------|----------|----------|----------|-----------|----------|
| Row0 | 15039603... | 4/12/2016 | 13162 | 8.5 | 8.5 | 0 | 1.88 | 0.55 | 6.06 | 0 |
| Row1 | 15039603... | 4/13/2016 | 10735 | 6.97 | 6.97 | 0 | 1.57 | 0.69 | 4.71 | 0 |
| Row2 | 15039603... | 4/14/2016 | 10460 | 6.74 | 6.74 | 0 | 2.44 | 0.4 | 3.91 | 0 |
| Row3 | 15039603... | 4/15/2016 | 9762 | 6.28 | 6.28 | 0 | 2.14 | 1.26 | 2.83 | 0 |
| Row4 | 15039603... | 4/16/2016 | 12669 | 8.16 | 8.16 | 0 | 2.71 | 0.41 | 5.04 | 0 |
| Row5 | 15039603... | 4/17/2016 | 9705 | 6.48 | 6.48 | 0 | 3.19 | 0.78 | 2.51 | 0 |

- **Figure 2: Configuration of CSV Reader Node**
- **COLUMN FILTER:**
- This column filter node provides three distinct modes for filtering: manually, by name, and by type. Through Add and Remove buttons, you manually choose which columns to retain and which to exclude.- by name, you decide which columns to keep using wildcards and regular expressions.

- By type option, you choose which columns to retain based on their type, such as all Strings or all Integers. The configuration window of this column filter is shown in figure 3.

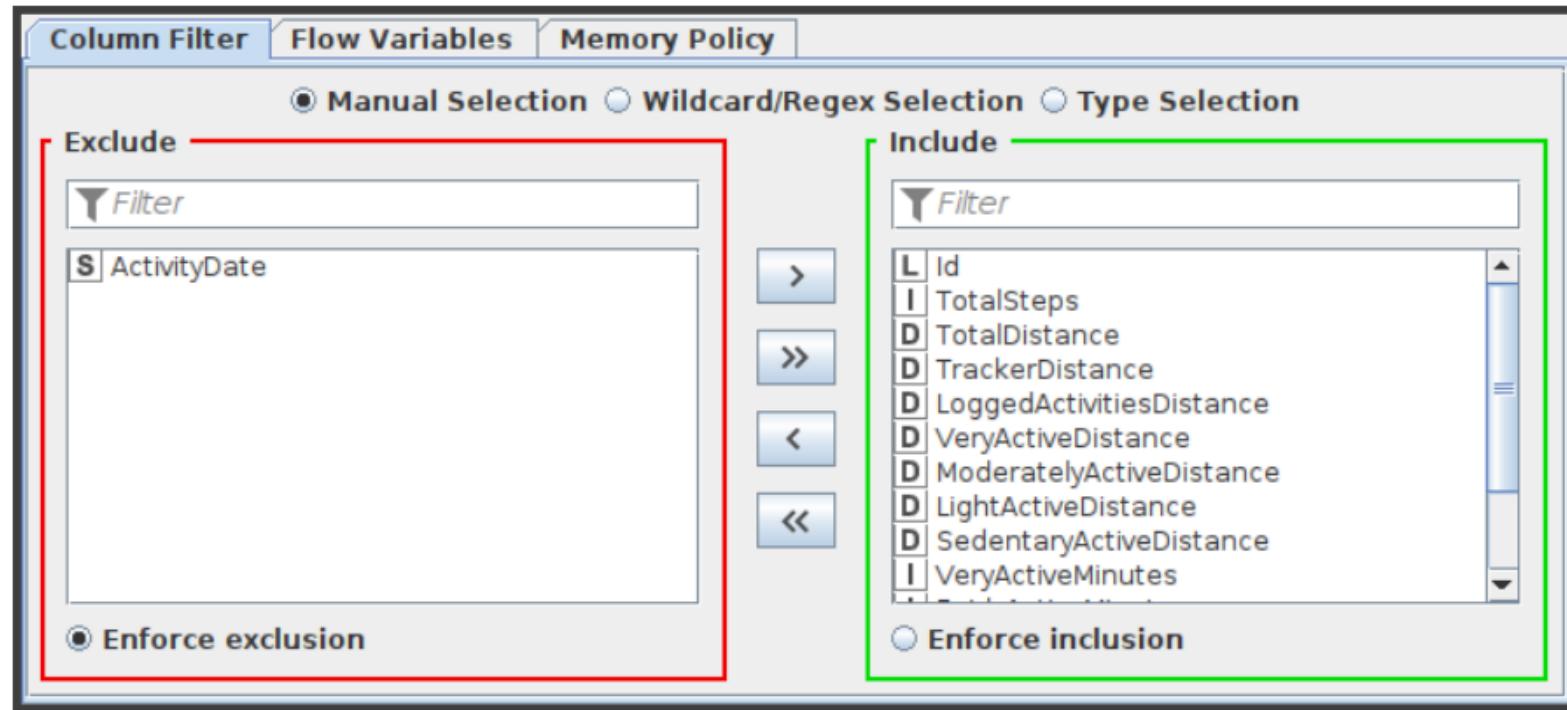


Figure 3:Configuration of Column Filter Node

MISSING VALUE:

- This node aids in handling cells in the input table that have missing values. All columns of a particular kind have default handling options available on the dialog's first tab, "Default." All columns in the input table that aren't specifically listed in the second tab, "Individual," have these settings applied to them. Individual settings for each available column are possible using this second tab (thus, overriding the default). Use the second method by selecting the column or group of columns that requires extra treatment, clicking "Add," and then setting the settings. All covered columns in the column list will be selected when you click on the label with the column name(s). Click the "Remove" button for this column to get rid of the extra treatment and replace it with the default handling. Asterisk-denoted options (*) will produce non-standard PMML. A warning will be displayed throughout the execution of the node if you choose this option, and the caution label in the dialogue will turn red. Extensions used by non-standard PMML cannot be read by any software other than Knime. The configuration window of missing value node is captured and indicated in figure 4.

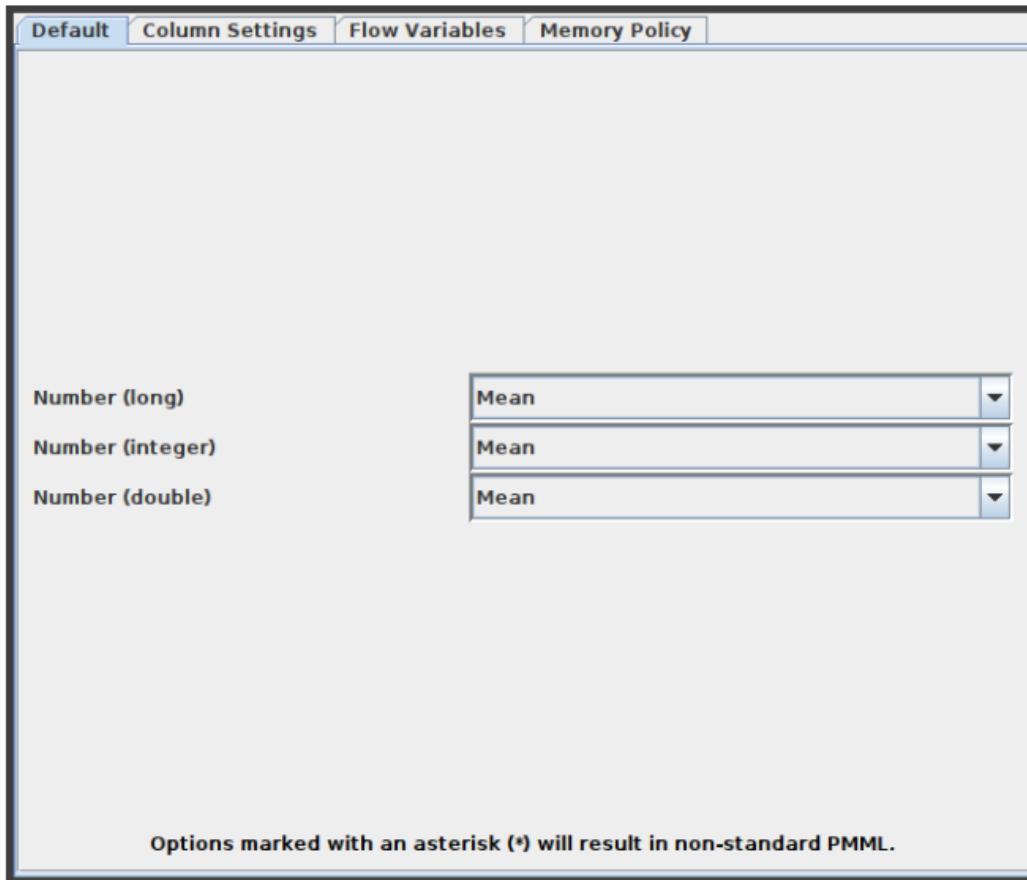


Figure 4: Configuration of Missing Value Node

RULE ENGINE:

This node attempts to match a list of user-defined rules to each row in the input table. If a rule is satisfied, the result value is added to a new column. The outcome is determined by the first matching rule in definition order. One line represents each regulation. To insert comments, begin a line with / (comments can not be placed in the same line as a rule). Anything following / will not be considered a rule. Rules consist of a condition part (antecedent) that must evaluate to true or false and a consequence (consequent, after the => symbol) that is placed in the new column if the rule matches. A rule's output may be a string (between " and /), a number, a boolean constant, a reference to another column, or the value of a flow variable. The type column represents the common supertype of all potential outcomes (including the rules that can never match). If no rule is applicable, the result is a missing value. Columns are denoted by their respective names surrounded by \$, while numbers are presented in the customary decimal format. Note that strings cannot contain (double) quotation marks. TypeCharacterAndFlowVarName is used to represent flow variables. The TypeCharacter for double (real) values should be 'D', for integer values 'I', and for strings 'S'. You can manually insert column names and flow variables or use the respective lists in the dialogue. The setup window for the rule engine node is depicted in figure 5.

Category

All

Description

Function

- ? < ?
- ? <= ?
- ? = ?
- ? > ?
- ? >= ?
- ? AND ?
- ? IN ?
- ? LIKE ?
- ? MATCHES ?
- ? OR ?
- ? XOR ?
- FALSE
- MISSING?

Expression

```

S 1 $Calories$ > 2500 => "High calories burn"
S 2 $Calories$ < 2000 => "Low calories burn"
S 3 $Calories$ >= 2000 AND $Calories$ <= 2500 => "Good calories burn"

```

Append Column: Calories burn

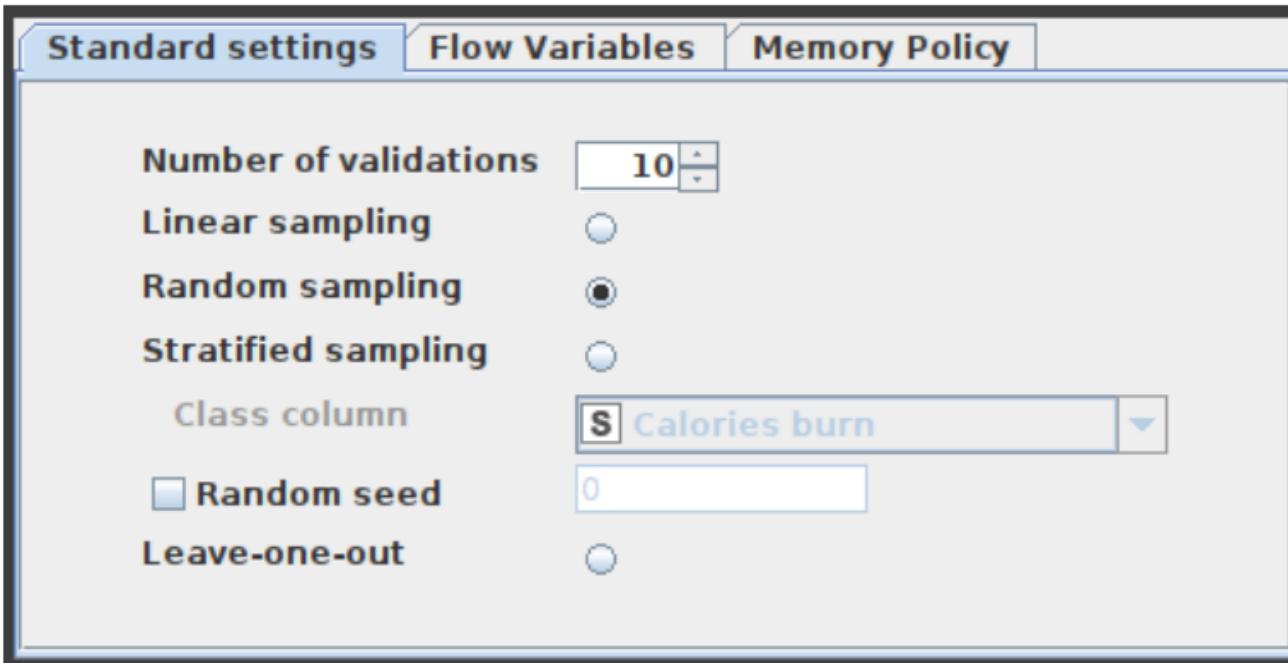
Replace Column: Calories

- Figure 5: Configuration of Rule Engine Node

X-Partitioner:

This node begins the cross validation loop. A X-Aggregator is required at the conclusion of the loop to collect the results of each iteration. All nodes between these two nodes are run as many times as required by the number of iterations. Figure 6 shows the configuration window for the X-Partitioner node that is now selected for this application.

X-PARTITIONER :



- **Figure 6: Configuration of X-Partitioner Node**
- **Random Forest Learner:**
- Minitab, LLC has registered RANDOM FORESTS as a trademark, and its usage is authorised. It is comprised of a predetermined number of decision trees. Each decision tree model is constructed using a unique collection of rows (records), and for each branch, a random set of columns (describing attributes) is employed. Each decision tree's row sets are generated using bootstrapping and have the same size as the original input table. The attribute set for a decision tree split is determined by randomly selecting \sqrt{m} attributes from the available attributes, where m is the total number of learning columns. The properties may also be supplied in bit (fingerprint), byte, or double vector formats. The output model is applied to the associated predictor node and defines a random forest. This node provides a subset of the Tree Ensemble Learner's capabilities corresponding to a random forest. The configuration window for the Random Forest node that has now been chosen for this application can be seen in Figure 7.

RANDOM FOREST LEARNER :

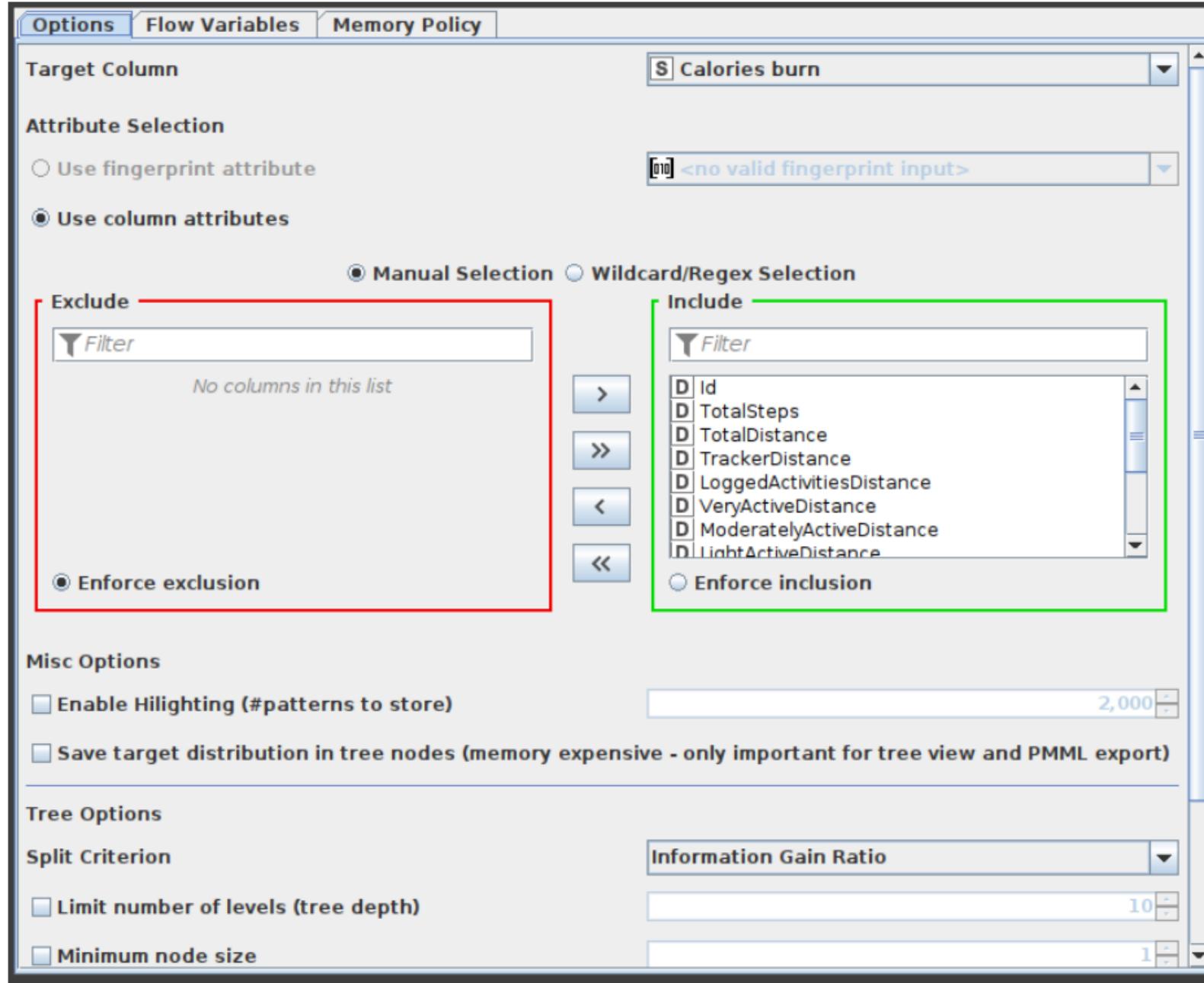


Figure 7: Configuration of Rule Engine Node

Random Forest Predictor:

- In a random forest model, this function makes pattern predictions based on an aggregation of the predictions made by the individual trees. Figure 8 displays the settings window for the Random Forest Predictor node.

RANDOM FOREST PREDICTOR :

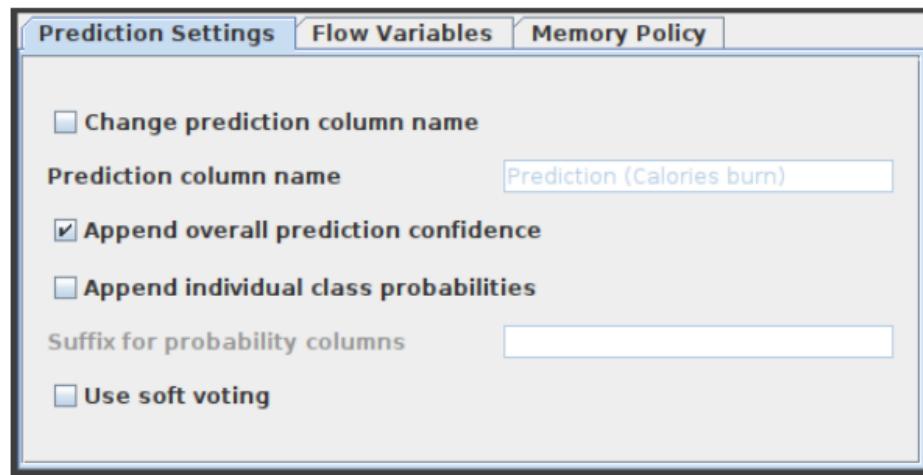


Figure 8:Configuration of Random Forest Predictor Node

5.8 X-Aggregator Node:

This node must be located at the conclusion of a cross validation loop and must be the node that comes after an X-Partitioner node. It does this by first collecting the result from a predictor node, then comparing the predicted class to the actual class, and finally outputting the predictions for all rows together with the iteration statistics. Figure 9 illustrates the window for the X-Aggregator node's configuration options.

X-AGGREGATOR NODE :

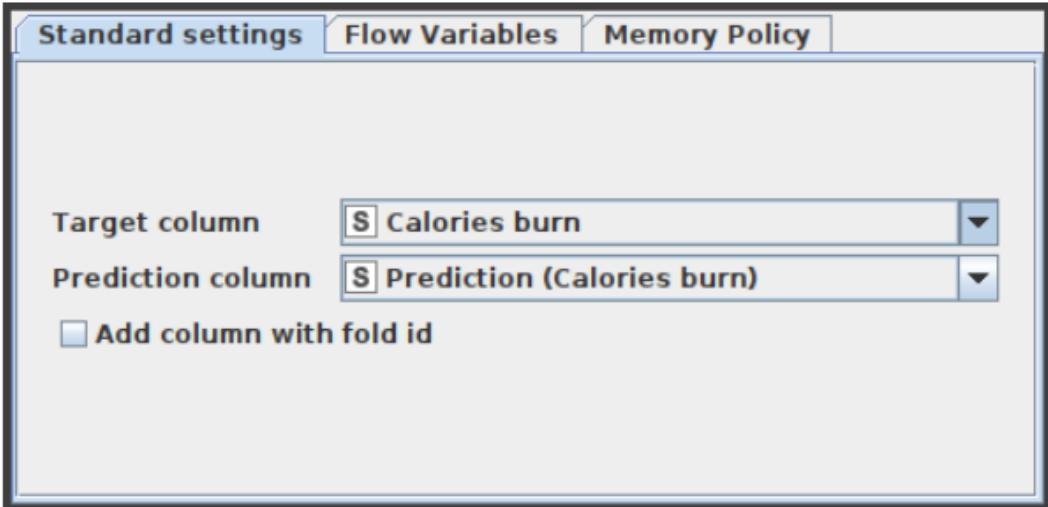


Figure 9: Configuration of X-Aggregator Node

5.9 Scorer:

Displays the confusion matrix (the number of rows that share an attribute with their classification) by comparing two columns based on attribute value pairs. You may also highlight cells in this matrix to reveal the underlying rows. The comparison dialogue lets you pick two columns to compare; the rows of the confusion matrix reflect the values in the first column you pick, while the columns reflect the values in the second column. The node returns a confusion matrix where each cell indicates the amount of matches. Statistics on recall, precision, sensitivity, specificity, F-measure, overall accuracy, and Cohen's kappa are also reported on the second output port. The X-Aggregator node's settings window is shown in Figure 10.

SCORER :

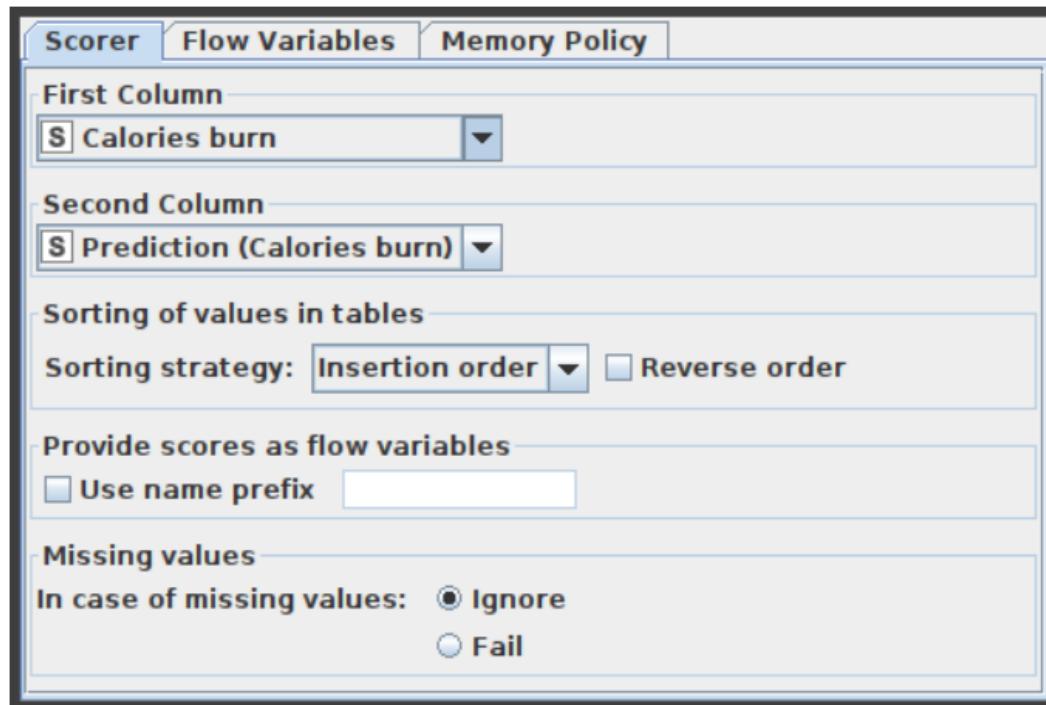


Figure 10: Configuration of Scorer Node

5.10 Bar Plot:

Custom CSS styling is supported through a bar char node. The node configuration dialogue allows you to easily condense CSS rules into a single string and set it as the flow variable “customCSS.” On our documentation page, you can discover a list of the classes that are available as well as their descriptions. Figure 11 exhibits the parameters configuration window for the Bar Plot node.

BAR PLOT :

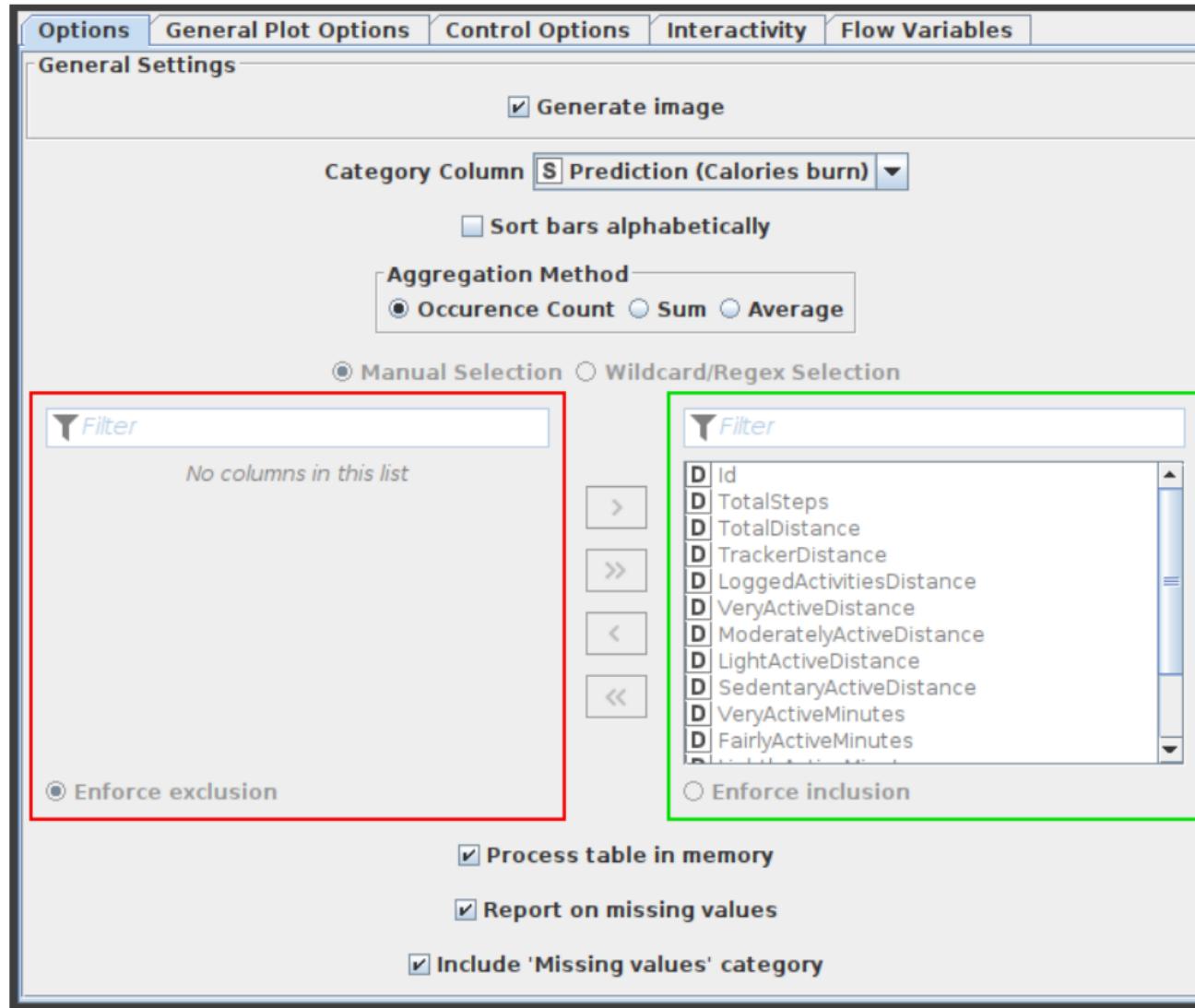


Figure 11: Configuration of Bar Plot Node

Results Screenshots:

6.1 Scorer Node Results:

The prediction of calories dataset is presented here in figure 12 as the output analysis of scorer node.

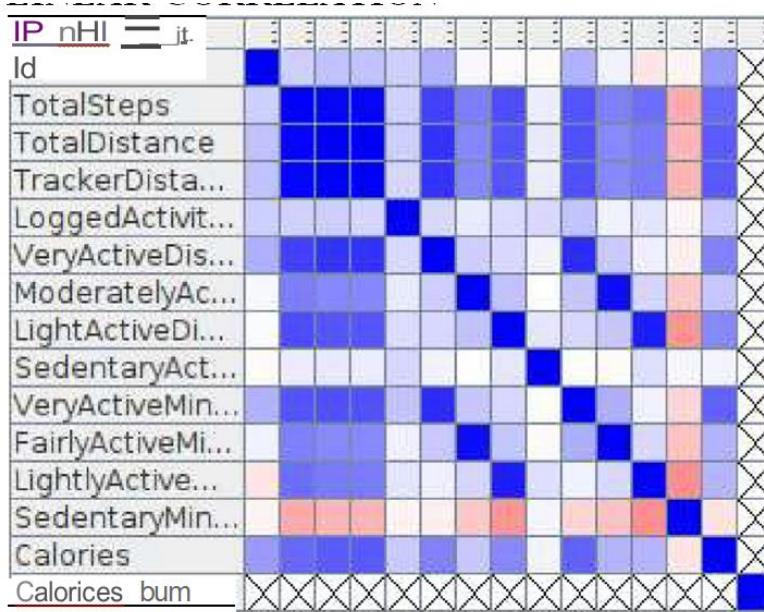
| Calories ... | | | |
|---------------|-------------|--------------|-----|
| Low calori... | Good cal... | High calo... | |
| Low calor... | 369 | 0 | 0 |
| Good cal... | 0 | 234 | 0 |
| High calo... | 0 | 4 | 333 |

| | |
|--|----------------------------|
| Correct classified: 936 | Wrong classified: 4 |
| Accuracy: 99.574% | Error: 0.426% |
| Cohen's kappa (κ): 0.994% | |

- **Figure 12:Result Outcome of Scorer Node**

6.2 Linear Correlation Output:

The correlation matrix of the linear correlation analysis is presented in figure 13.

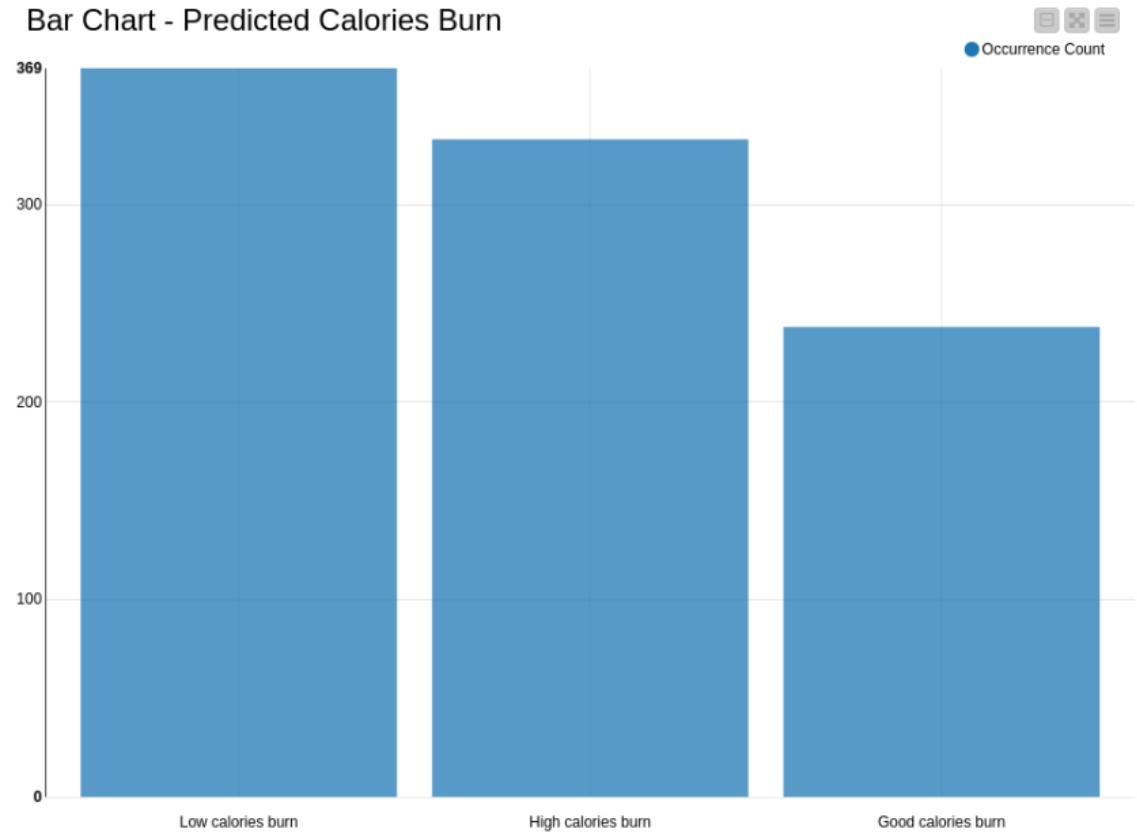


- **Figure 13: Result Outcome of Linear Correlation Node**

6.3 Graphs – Bar Graph:

Appropriate analysis has been made and the bar graph for the predicted calories burn is illustrated in figure 14 interperpting the occurrence count of low calories burn, good calories burn and high calories burn.

Bar Chart - Predicted Calories Burn

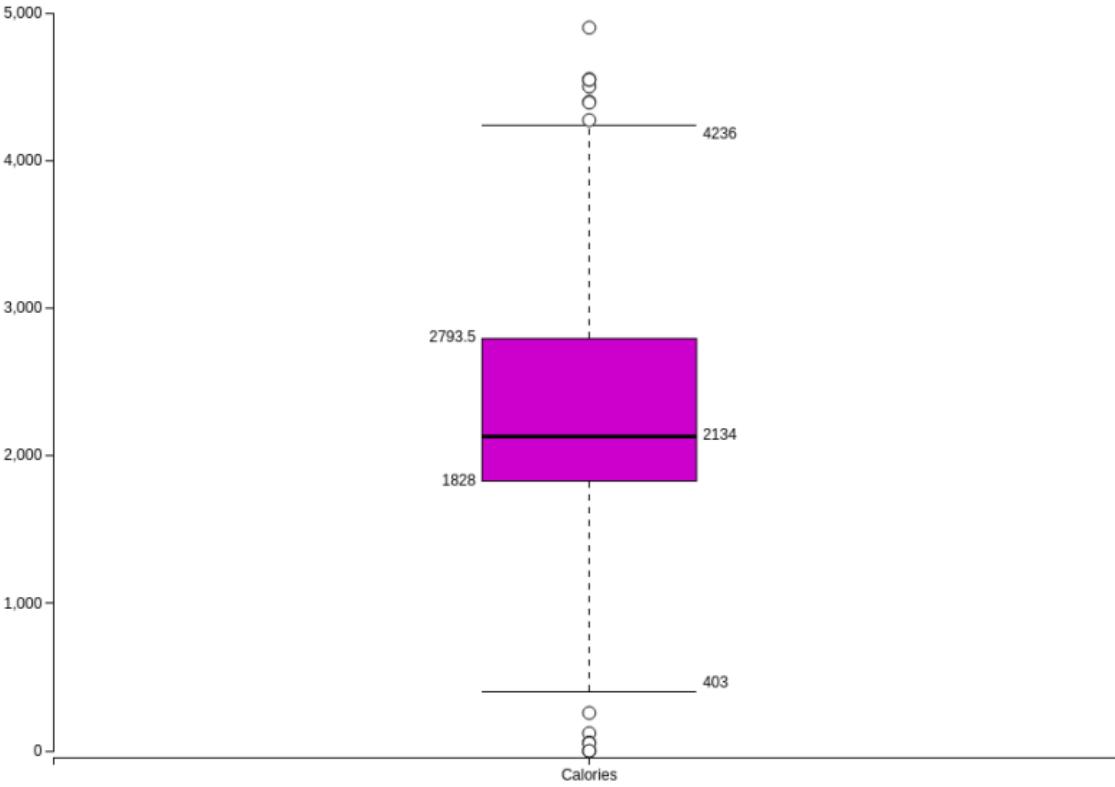


- **Figure 14:Bar Graph Analysis of Calories Burn**

6.4 Box Plot:

The box plot for the calories analysis is presented in figure 15.

Box Plot - Calories



- **Figure 15:Box Plot Analysis of Calories Burn**

6.5 Error Rate in Cross Validation:

The error rate in cross validations using X-Partitioner and X-Aggregator or 10 validations is depicted in figure 16.

| Row ID | D Error in % | I Size of Test Set | I Error Count |
|--------|--------------|--------------------|---------------|
| fold 0 | 0 | 94 | 0 |
| fold 1 | 1.064 | 94 | 1 |
| fold 2 | 0 | 94 | 0 |
| fold 3 | 0 | 94 | 0 |
| fold 4 | 0 | 94 | 0 |
| fold 5 | 0 | 94 | 0 |
| fold 6 | 0 | 94 | 0 |
| fold 7 | 1.064 | 94 | 1 |
| fold 8 | 1.064 | 94 | 1 |
| fold 9 | 1.064 | 94 | 1 |



Figure 16: Error Rate in Cross Validation

RESULT: The final accuracy shown by the random forest model is 99.574% and the error is 0.426%. Knime Analytics Platform was successfully used to perform data analysis operations and random forest algorithm for Fitness tracker dataset.

Chapter 7

Knime Analytics – Case Studies

Case Study 1:

AIM: To create an application to preprocess and analyze data (in this case video game sales) using various nodes from Knime.

THEORY:

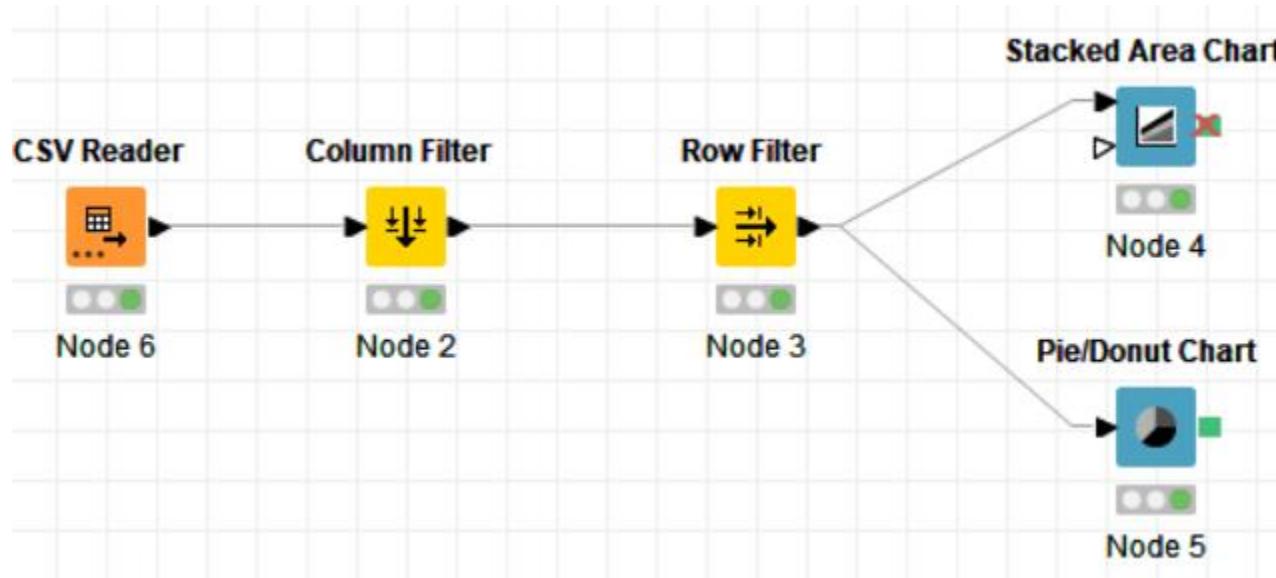
CSV Files are a mine of data and information. However, even they possess data only in the raw form which is not of much use. Graphical outputs such as pie charts and histograms give a very good clue about the trend in the various patterns of the dataset. Knime is a tool that lets the user pre-process and analyse data and then display it in graphical format

NODES REQUIRED:

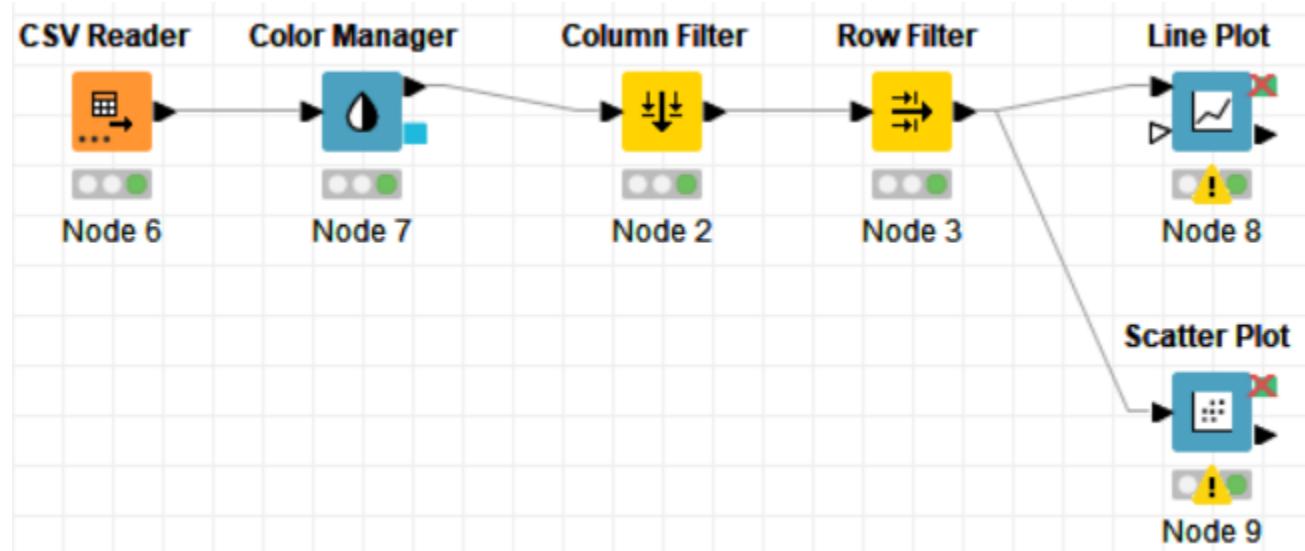
The nodes required here are the csv root node with the .csv file, a column filter and row filter to filter out unnecessary values, and output nodes such as pie chart and stacked area chart. The data is preprocessed up to the row filter after which it is graphically represented.

FLOW DIAGRAM DESIGN:

Flow1:



Flow 2:



PROCEDURE:

Drag the .csv file into Knime to create and initialize the CSV Reader node. If necessary, create a color manager node and choose color for specified column. Create a column filter, connect it to the color manager and filter the industry subgroup column. Connect said node to a row filter where all rows having arts as an industry is removed by using pattern matching. Pass the processed data to the pie chart node where it is segregated by occupation and the stack area chart node as well as scatter plot and line plot nodes where the value is displayed.

PROPERTIES SET FOR EACH NODE WITH EXPLANATION:

Flow 1:

Column Filter | Flow Variables | Memory Policy

Manual Selection Wildcard/Regex Selection Type Selection

Exclude

Filter

Industry subgroup

Enforce exclusion

Include

Filter

- Year
- Sex
- Age group (years) at date of injury
- Geographic region where injury occurred
- Employment status
- Occupation
- Injury/illness/disease group
- Type of injury/illness/disease
- Industry
- Value
-

Enforce inclusion

Column value matching

Column to test: filter based on collection elements

Matching criteria

 use pattern matching case sensitive match contains wild cards regular expression

- Include rows by attribute value
- Exclude rows by attribute value
- Include rows by number
- Exclude rows by number
- Include rows by row ID
- Exclude rows by row ID

 use range checkinglower bound: upper bound: only missing values match

General Settings

 Generate image

Maximum number of rows

2,500

Choose column for x-axis Status Sort for x-axis column Manual Selection Wildcard/Regex Selection

 Filter

Year

Enforce exclusion

 Filter

Value

Enforce inclusion

>

>>

<

<<

Flow 2:

Select one Column

S Industry

 Nominal

- █ Information media and telecommunications
- █ Manufacturing
- █ Mining
- █ Not specified
- █ Other services
- █ Professional scientific and technical services
- █ Public administration and safety
- █ Rental hiring and real estate services
- █ Retail trade
- █ Total
- █ Transport postal and warehousing
- █ Wholesale trade

 Range

Preview

[Palettes](#) [Swatches](#) [HSV](#) [HSL](#) [RGB](#) [CMYK](#) [Alpha](#) Set 1 Set 2 Set 3 (colorblind safe) Custom

Manual Selection Wildcard/Regex Selection Type Selection

Exclude



- Industry subgroup
- Measure
- Status



Include



- Year
- Sex
- Age group (years) at date of injury
- Geographic region where injury occurred
- Employment status
- Occupation
- Injury/illness/disease group
- Type of injury/illness/disease
- Industry
- Value

Enforce inclusion

Enforce exclusion

Column value matching

Column to test: filter based on collection elements

Matching criteria

 use pattern matching case sensitive match contains wild cards regular expression use range checkinglower bound: upper bound: only missing values match

- Include rows by attribute value
- Exclude rows by attribute value
- Include rows by number
- Exclude rows by number
- Include rows by row ID
- Exclude rows by row ID

Create image at output

Maximum number of rows:

2,500

Choose column for x-axis

 <RowID>

Choose columns for y-axis

 Manual Selection Wildcard/Regex Selection

Exclude



- I Year
- S Sex
- S Age group (years) at date of injury
- S Geographic region where injury occurred
- S Employment status
- S Occupation
- S Injury/illness/disease group
- S Type of injury/illness/disease

 Enforce exclusion

Include



D Value

 Enforce inclusion Report on missing values

Missing value handling (y-axis)

 Connect

Create image at output

Maximum number of rows:

2,500

Selection column name:

Selected (Scatter Plot)

Choose column for x axis

Injury/illness/disease group

Choose column for y axis

Value

Report on missing values

OUTPUT:

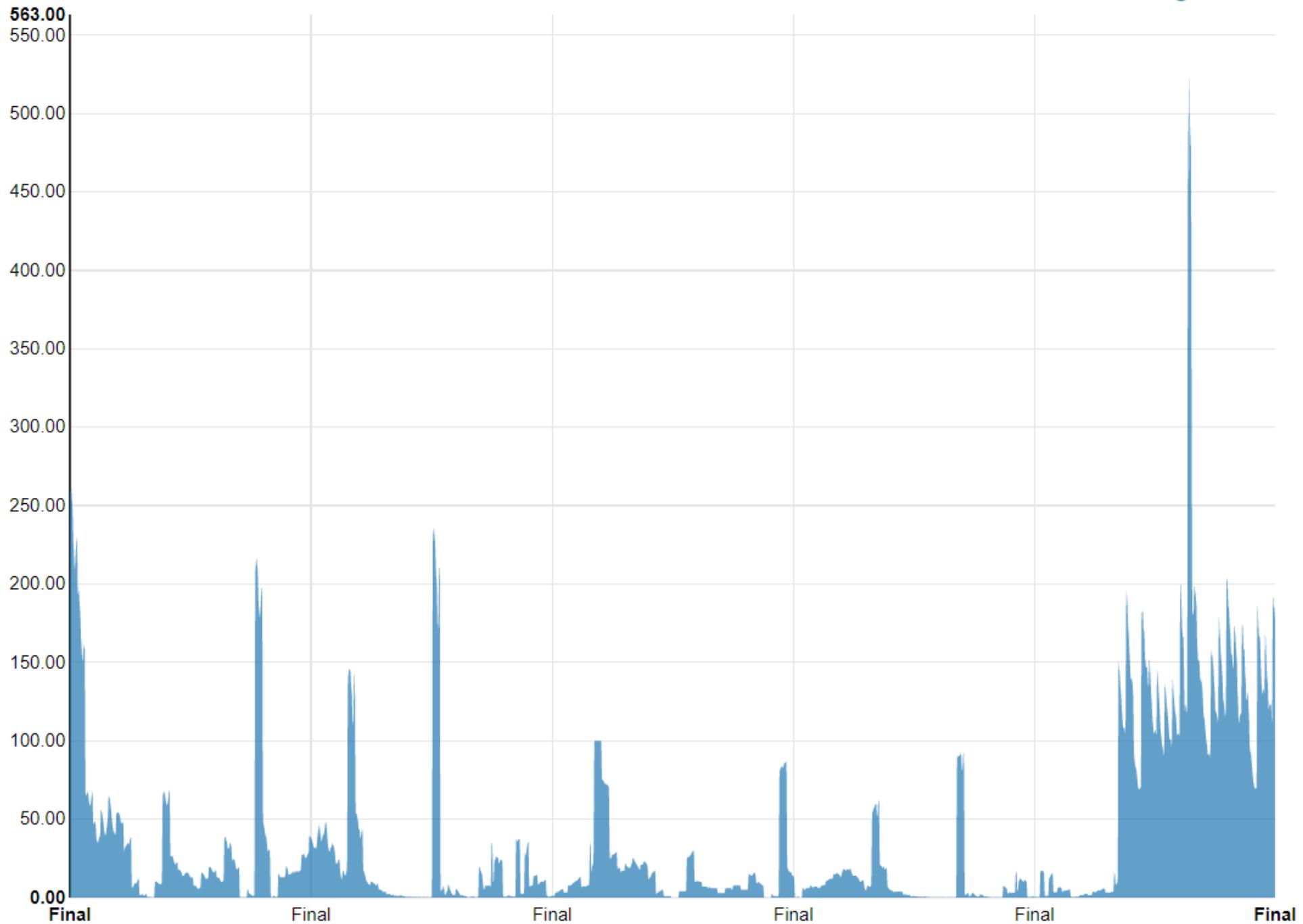
SNAPSHOTS:

Flow 1:

Stacked Area Chart



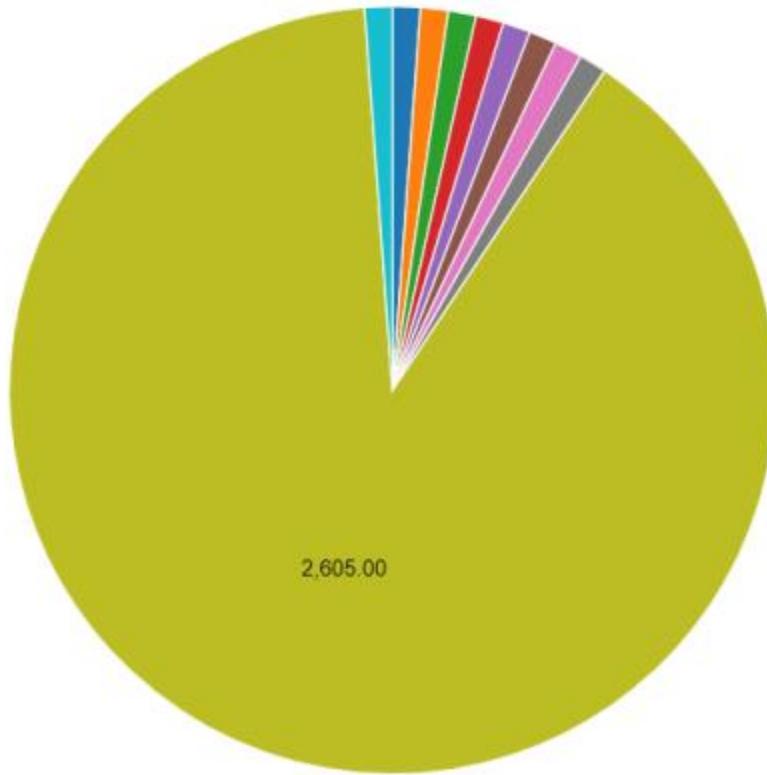
Value



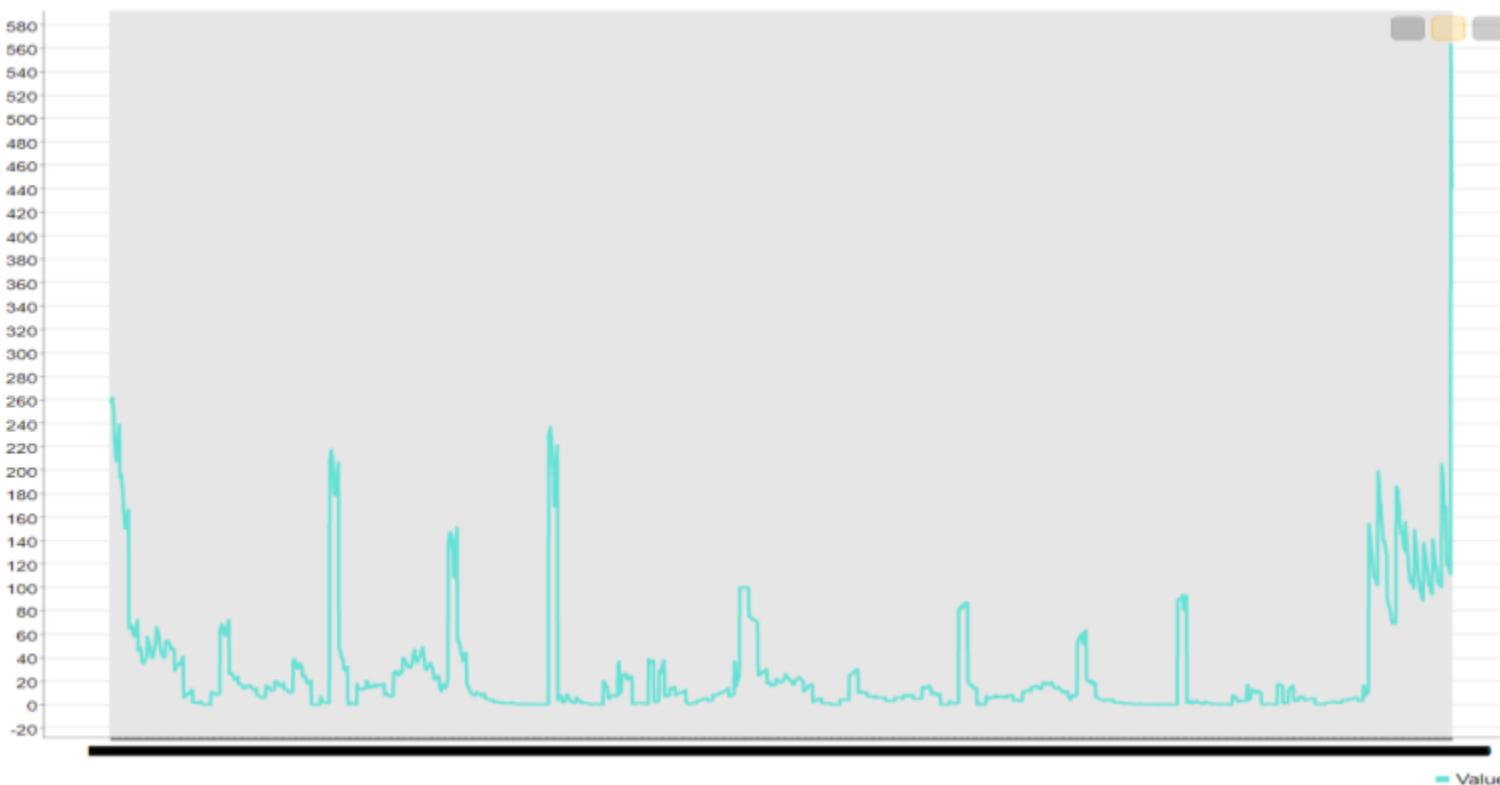
Pie Chart

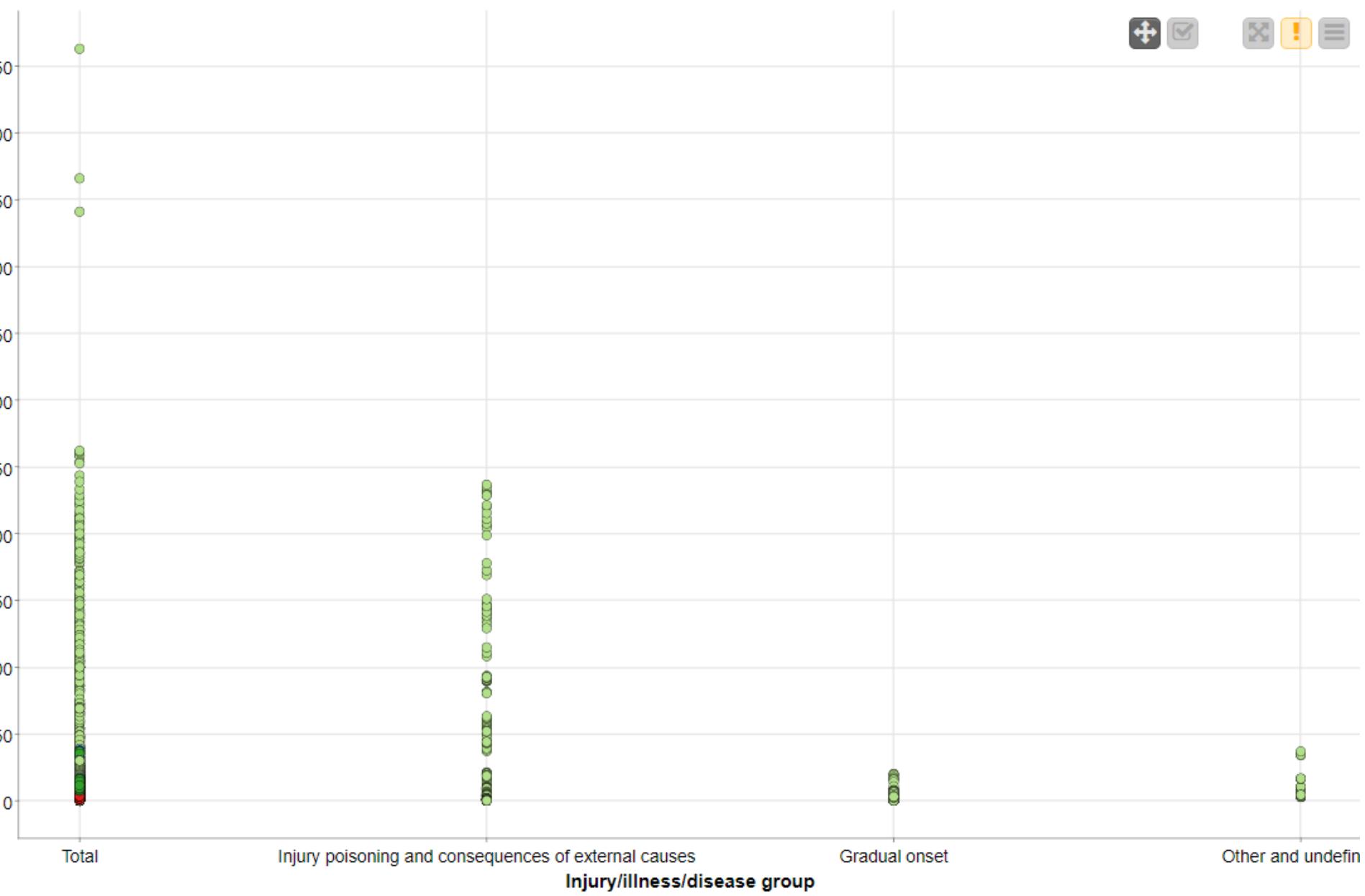


- Agriculture and fishery workers
- Elementary occupations
- Legislator administrators and managers
- Not specified
- Plant and machine operators and assemblers
- Professionals
- Service and sales workers
- Technicians and associate professionals
- Total
- Trades workers



Flow 2:





RESULT:

The CSV data from the file was preprocessed and displayed using various graphical outputs.

Knime Rule Engine

AIM: To create an application to preprocess and analyze data (in this case video game sales) using various nodes from Knime.

THEORY:

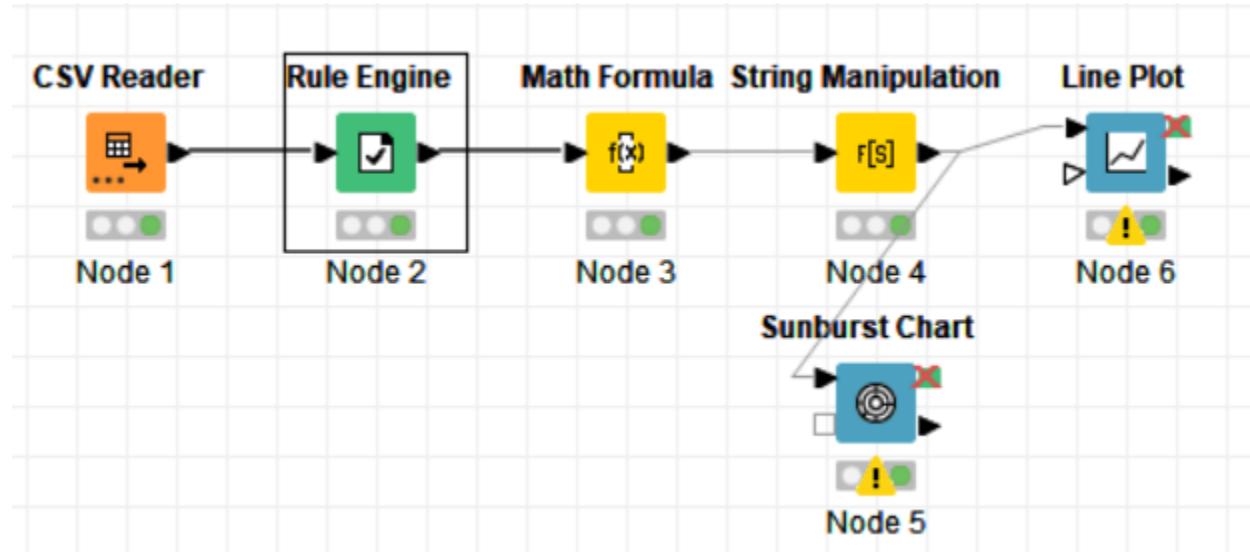
CSV Files are a mine of data and information. However, even they possess data only in the raw form which is not of much use. Graphical outputs such as pie charts and histograms give a very good clue about the trend in the various patterns of the dataset. Knime is a tool that lets the user pre-process and analyse data and then display it in graphical format. The rule engine node in knime takes a list of user-defined rules and tries to match them to each row in the input table. If a rule matches, its outcome value is added into a new column. The first matching rule in order of definition determines the outcome.

ALGORITHM:

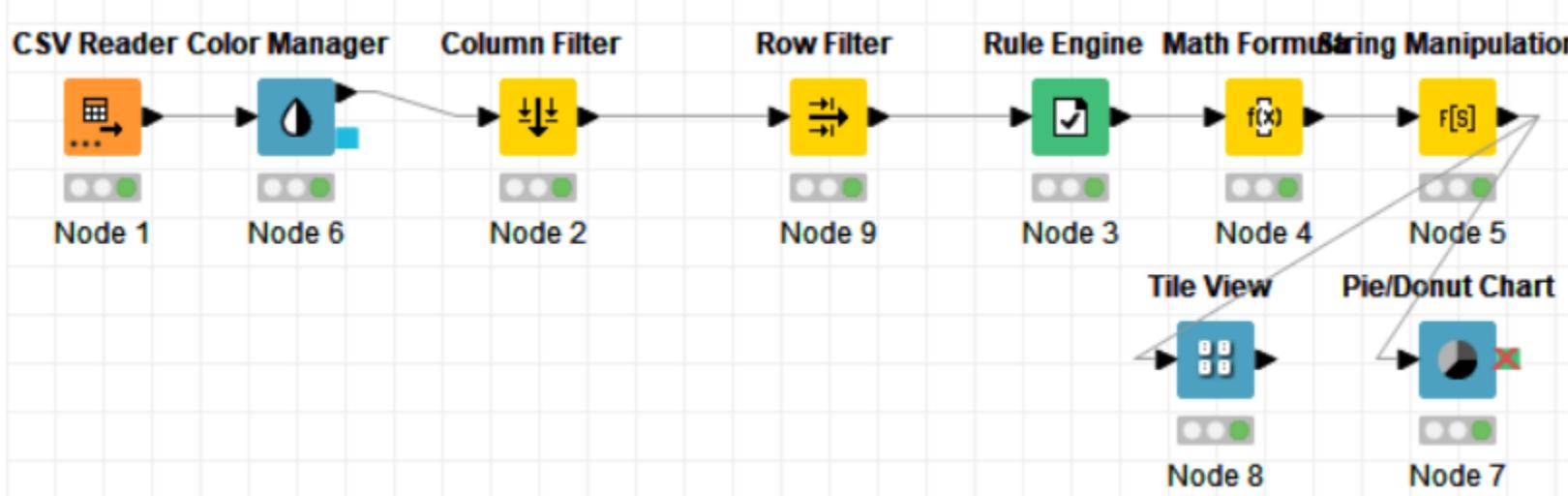
The nodes required here are the csv root node with the .csv file, a rule engine node, math formula and string manipulation nodes for numerical and string data respectively, and output nodes such as line plot and sunburst chart. The data is preprocessed, sent to the rule engine where values are classified, changed using the math and string nodes after which it is graphically represented.

FLOW DIAGRAM DESIGN:

Flow1:



Flow 2:



PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. If necessary, create a color manager node and choose color for specified column. Create a column filter, connect it to the color manager and filter the industry subgroup column. Connect this to a rule engine node, where specific conditions can be set. This is then sent to the Math Formula node where mathematical operations can be applied to the columns. Next, the String Manipulation node carries out string operations on designated columns. Pass the processed data to the pie chart node where it is segregated by occupation and the sunburst chart node as well as line plot node where the value is displayed.

PROPERTIES SET FOR EACH NODE:

Flow 1:

Column List

ROWID
ROWINDEX
ROWCOUNT
I Year
S Sex
S Age group (years) at date of injury
S Geographic region where injury occurred
S Employment status
S Occupation
S Injury/illness/disease group
S Type of injury/illness/disease
S Industry
S Industry subgroup
D Value
S Measure

Flow Variable List

S knime.workspace

Category

Description

All

Function

? < ?
? <= ?
? = ?
? > ?
? >= ?
? AND ?
? IN ?
? LIKE ?
? MATCHES ?
? OR ?
? XOR ?
FALSE
MISSING ?
NOT ?

Expression

```
1 // enter ordered set of rules, e.g.:  
2 // $double column name$ > 5.0 => "large"  
3 // $string column name$ LIKE "*blue*" => "small and blue"  
4 // TRUE => "default outcome"  
5 $Value$<=50=>"Safe Workplace"  
6 $Value$ > 50 AND $Value$ <= 150 =>"Moderately dangerous Workplace"  
7 $Value$ > 150 =>"Very dangerous Workplace"
```

Append Column: prediction S

Replace Column: S Status

Column List

ROWINDEX
ROWCOUNT
 Year
 Value

Category

All

Description

Function

ROWCOUNT
ROWINDEX
pi
e
COL_MIN(col_name)
COL_MAX(col_name)
COL_MEAN(col_name)
COL_MEDIAN(col_name)
COL_SUM(col_name)
COL_STDDEV(col_name)
COL_VAR(col_name)
ln(x)
log(x)
log10(x)

Flow Variable List

Expression

1 round(\$Value\$)

| Column List | Category | Description |
|---|---|--|
| <i>ROWID</i>
<i>ROWINDEX</i>
<i>ROWCOUNT</i>

I Year
S Sex
S Age group (years) at date of injury
S Geographic region where injury occurred
S Employment status
S Occupation
S Injury/illness/disease group
S Type of injury/illness/disease
C Industry | <input type="text" value="All"/> <input type="button" value="▼"/>

Function

capitalize(str)
capitalize(str, chars)
compare(str1, str2)
count(str, toCount)
count(str, toCount, modifiers)
countChars(str, chars)
countChars(str, chars, modifiers)
indexOf(str, toSearch)
indexOf(str, toSearch, modifiers)
indexOf(str, toSearch, start)
indexOf(str, toSearch, start, modifiers) | <input type="button" value="▲"/>

<input type="button" value="▼"/> |
| Flow Variable List

s knime.workspace | Expression

1 upperCase(\$Sex\$) | <input type="button" value="▲"/>

<input type="button" value="▼"/> |
| <input checked="" type="radio"/> Append Column: <input type="text" value="new column"/> | <input type="checkbox"/> Insert Missing As Null | |
| <input type="radio"/> Replace Column: <input type="text" value="D new column"/> <input type="button" value="▼"/> | <input checked="" type="checkbox"/> Syntax check on close | |

General Settings

Generate image

Maximum number of rows

Manual Selection Wildcard/Regex Selection



- S** Sex
S Age group (years) at date of injury
S Geographic region where injury occurred
S Employment status
S Occupation
S Measure
S Status
S prediction
S new column (#1)

>
>>
<
<<



- S** Injury/illness/disease group
S Type of injury/illness/disease
S Industry
S Industry subgroup

Enforce exclusion

Enforce inclusion

Value Column

Filter out small nodes

Threshold for filtering (radians)

Create image at output

Maximum number of rows:

2,500

Choose column for x-axis

? <RowID>

Choose columns for y-axis

Manual Selection Wildcard/Regex Selection

Exclude



- I Year
- S Sex
- S Age group (years) at date of injury
- S Geographic region where injury occurred
- S Employment status
- S Occupation
- S Injury/illness/disease group
- S Type of injury/illness/disease

Enforce exclusion

Include



D Value

Enforce inclusion

Report on missing values

Missing value handling (y-axis)

Connect

Flow 2:

Select one Column

S Type of injury/illness/disease

 Nominal Range

- █ Foreign body in orifice or eye
- █ Fracture or dislocation
- █ Hernia
- █ Industrial deafness
- █ Inhalation or ingestion non-gradual
- █ Laceration puncture or sting
- █ Local inflammation
- █ Occupational disease
- █ Pain syndromes
- █ Soft tissue injury
- █ Total
- █ Trauma-induced hearing loss

[Palettes](#) [Swatches](#) [HSV](#) [HSL](#) [RGB](#) [CMYK](#) [Alpha](#) Set 1 Set 2 Set 3 (colorblind safe) Custom

Manual Selection Wildcard/Regex Selection Type Selection

Exclude



I Year
S Status

Enforce exclusion

Include



S Sex
S Age group (years) at date of injury
S Geographic region where injury occurred
S Employment status
S Occupation
S Injury/illness/disease group
S Type of injury/illness/disease
S Industry
S Industry subgroup
D Value
C Measure

Enforce inclusion

[Filter Criteria](#)[Flow Variables](#)[Memory Policy](#)

Column value matching

Column to test: filter based on collection elements

Matching criteria

 use pattern matching  case sensitive match contains wild cards regular expression use range checking

lower bound:

upper bound:

 only missing values match

- Include rows by attribute value
- Exclude rows by attribute value
- Include rows by number
- Exclude rows by number
- Include rows by row ID
- Exclude rows by row ID

| Column List | Category | Description |
|--|----------|---|
| <i>ROWID</i> | All | Checks whether the value of the left expression is like the wildcard pattern defined by the right expression
For example: \$Col0\$ LIKE "H?llo*" |
| <i>ROWINDEX</i> | | |
| <i>ROWCOUNT</i> | | |
| S Sex | | |
| S Age group (years) at date of injury | | |
| S Geographic region where injury occurred | | |
| S Employment status | | |
| S Occupation | | |
| S Injury/illness/disease group | | |
| S Type of injury/illness/disease | | |
| S Industry | | |
| S Industry subgroup | | |
| D Value | | |
| S Measure | | |

| Flow Variable List | Expression |
|--------------------------|--|
| s knime.workspace | <pre> ? 1 // enter ordered set of rules, e.g.: ? 2 // \$double column name\$ > 5.0 => "Large" ? 3 // \$string column name\$ LIKE "*blue*" => "small and blue" ? 4 // TRUE => "default outcome" S 5 \$Value\$<=50=>"Safe Workplace" S 6 \$Value\$ > 50 AND \$Value\$ <= 150 =>"Moderately dangerous Workplace" S 7 \$Value\$ > 150 =>"Very dangerous Workplace" </pre> |

Append Column: **S**
 Replace Column: **D**

Column List

ROWINDEX

ROWCOUNT

 Value

Category

All

Description

mean in column: COL_MEAN(\$col_name\$)

Function

ROWCOUNT

ROWINDEX

pi

e

COL_MIN(col_name)

COL_MAX(col_name)

COL_MEAN(col_name)

COL_MEDIAN(col_name)

COL_SUM(col_name)

COL_STDDEV(col_name)

COL_VAR(col_name)

ln(x)

log(x)

log10(x)

Expression

1 COL_MEAN(\$Value\$)

 Append Column:

Mean

 Replace Column:

S prediction

 Convert to Int

| Column List | Category | Description |
|---|---|---|
| <i>ROWID</i> | All | Replaces multiple characters in a String in one go. |
| <i>ROWINDEX</i> | | |
| <i>ROWCOUNT</i> | | |
| Sex | | |
| Age group (years) at date of injury | | |
| Geographic region where injury occurred | | |
| Employment status | | |
| Occupation | | |
| Injury/illness/disease group | | |
| Type of injury/illness/disease | | |
| Industry | | |
| Industry subcategory | | |
| Flow Variable List | | |
| knime.workspace | | |
| | Function | |
| | removeDiacritic(str) | |
| | removeDuplicates(str) | |
| | replace(str, search, replace) | |
| | replace(str, search, replace, modifiers) | |
| | replaceChars(str, chars, replace) | |
| | replaceChars(str, chars, replace, modifiers) | |
| | replaceUmlauts(str, omitE) | |
| | reverse(str) | |
| | string(x) | |
| | strip(str...) | |
| | stripEnd(str...) | |
| | Examples: | |
| | replaceChars("abcABC", "ac", "") = "bABC" | |
| | replaceChars("abcABC", "ac", "x") = "xbABC" | |
| | replaceChars("abcABC", "ac", "xy") = "xbyABC" | |
| | replaceChars("abcABC", "ac", "xyz") = "xbyABC" | |
| | replaceChars(null, *, *) = null | |
| | replaceChars("", *, *) = "" | |
| | replaceChars("any", null, *) = "any" | |
| | replaceChars("any", *, null) = "any" | |
| | replaceChars("any", "", *) = "any" | |
| | * can be any character sequence. | |
| | Expression | |
| | 1 replaceChars(\$Employment status\$,"" , "Total") | |
| <input checked="" type="radio"/> Append Column: | new column | <input type="checkbox"/> Insert Missing As Null |
| <input type="radio"/> Replace Column: | D Mean | <input checked="" type="checkbox"/> Syntax check on close |

General Options

No. of rows to display:

Title:

Subtitle:

Display Options

Display row colors

Display column headers

Display fullscreen button

Fixed number of tiles per row (1 - 100)

Fixed tile width (30 - 5000px)

Select text alignment:

Left

Center

Right

Choose a title column:

Columns to display:

Manual Selection Wildcard/Regex Selection Type Selection

Exclude



- Age group (years) at date of injury
- Geographic region where injury occurred
- Employment status
- Injury/illness/disease group
- Type of injury/illness/disease
- Industry
- Industry subgroup
- Measure

Enforce exclusion

Include



- Occupation
- Value
- prediction

Enforce inclusion

Options

General Plot Options

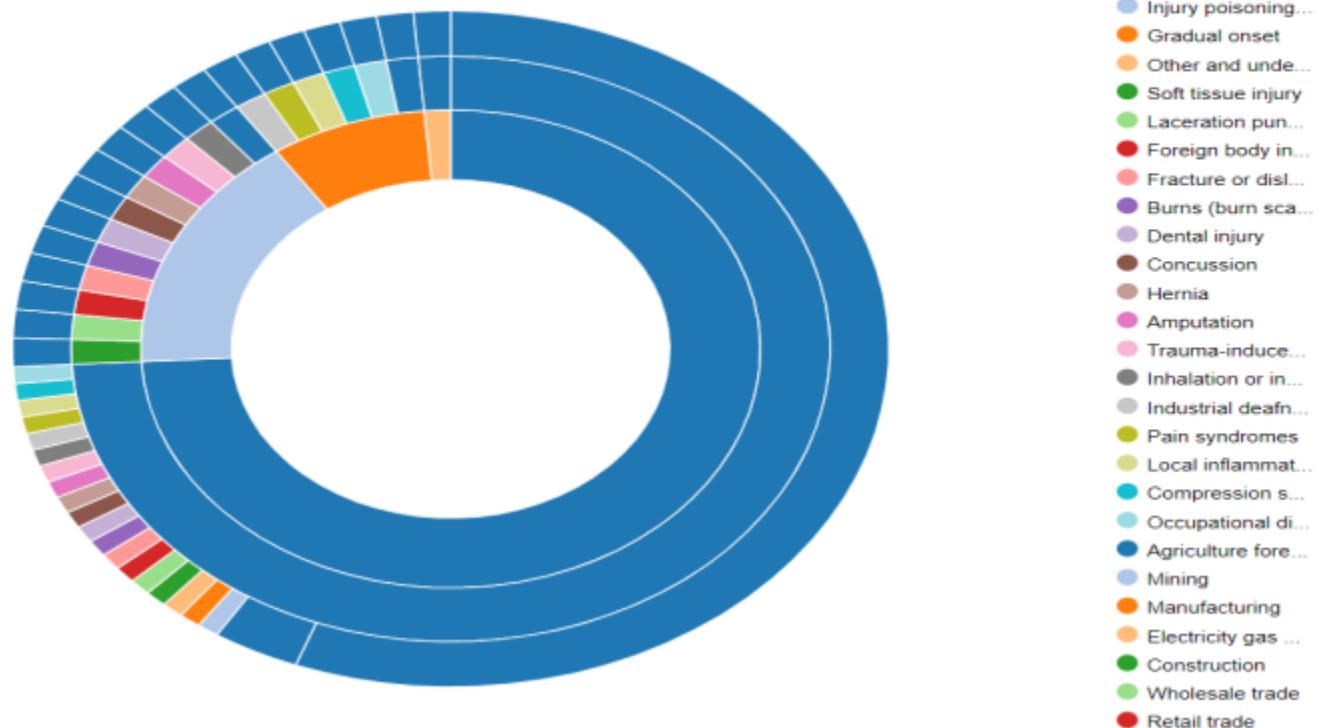
Control Options

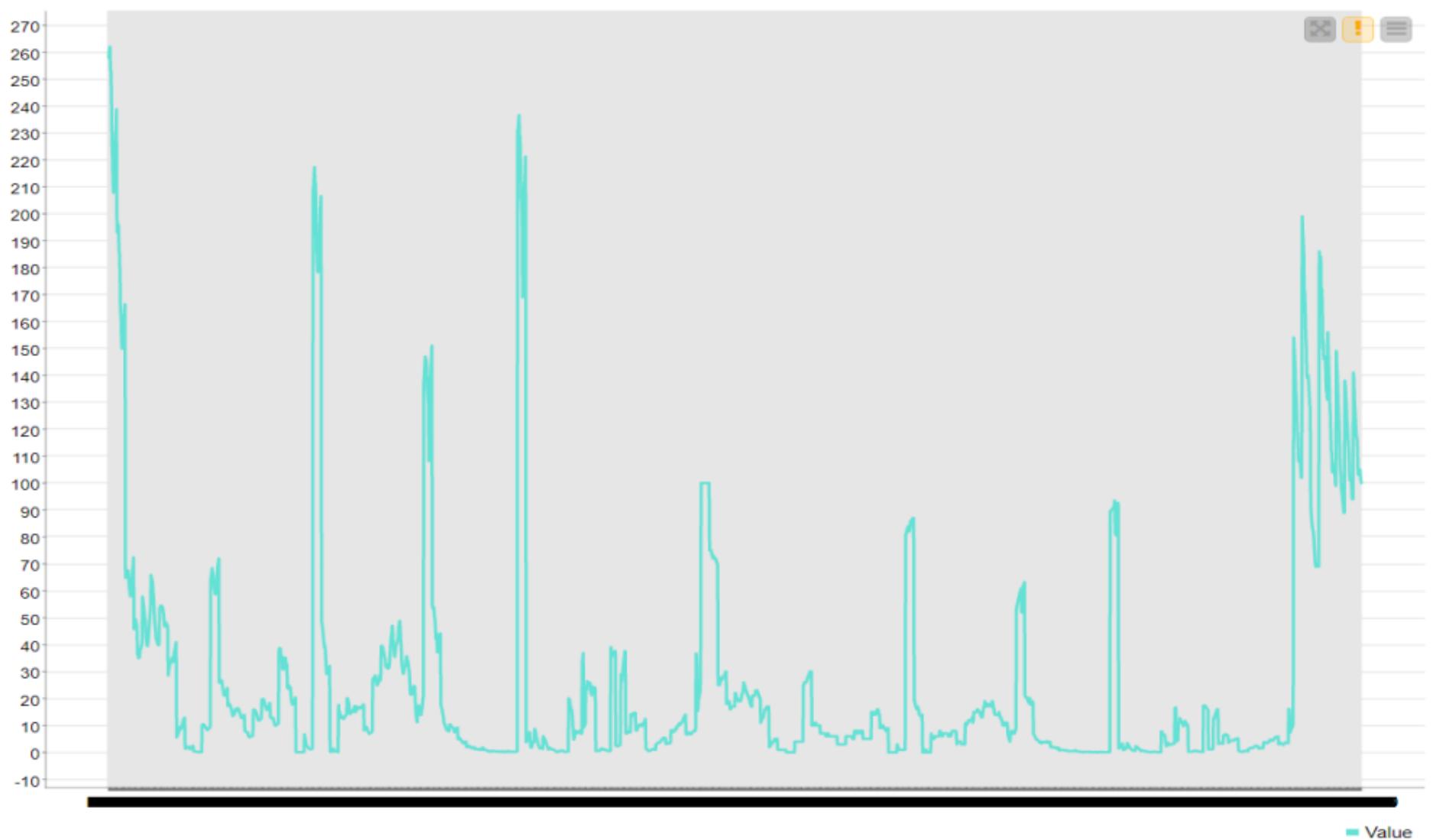
Interactivity

Flow Variables

General Settings Generate imageCategory Column Type of injury/illness/disease **Aggregation Method** Occurrence Count Sum Average Report on missing values Include 'Missing values' categoryFrequency Column Mean Process table in memory**OUTPUT:****SNAPSHOTS:**

Sunburst Chart



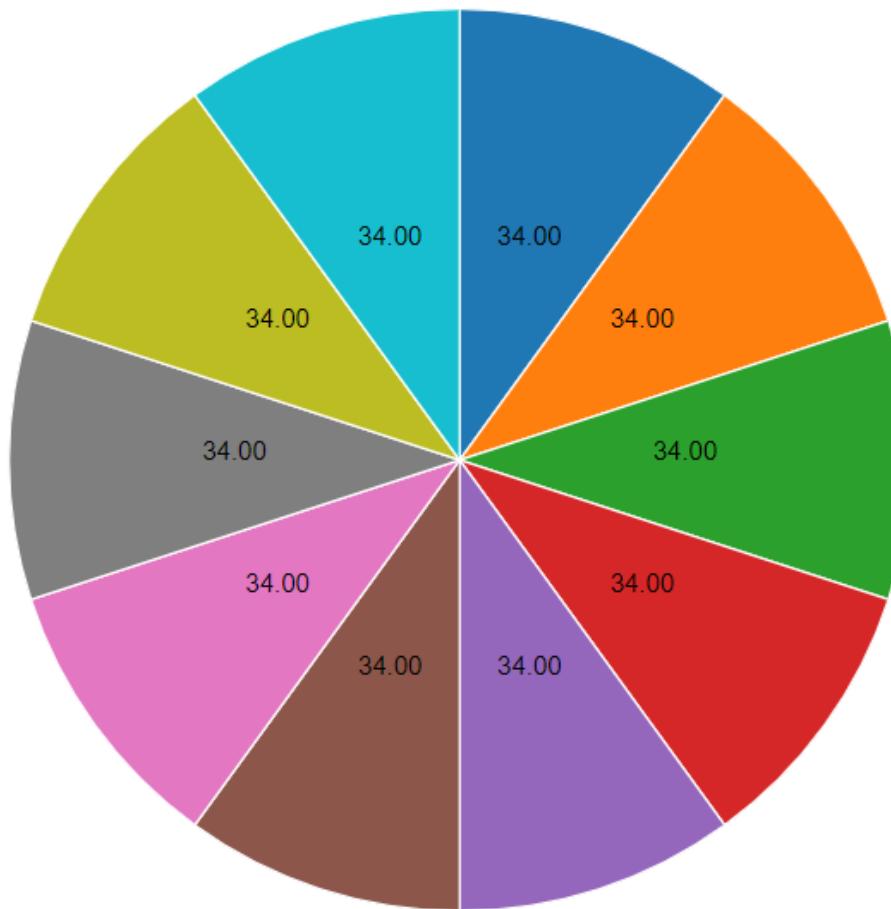


Flow :

Pie Chart



- Agriculture and fishery workers
- Clerks
- Elementary occupations
- Legislators administrators and managers
- Not specified
- Plant and machine operators and assemblers
- Professionals
- Service and sales workers
- Technicians and associate professionals
- Trades workers



RESULT:

The CSV data from the file was preprocessed using the rule engine and then displayed using various graphical outputs.

AIM: To create an application to preprocess and analyze data (in this case video game sales) using various nodes from Knime.

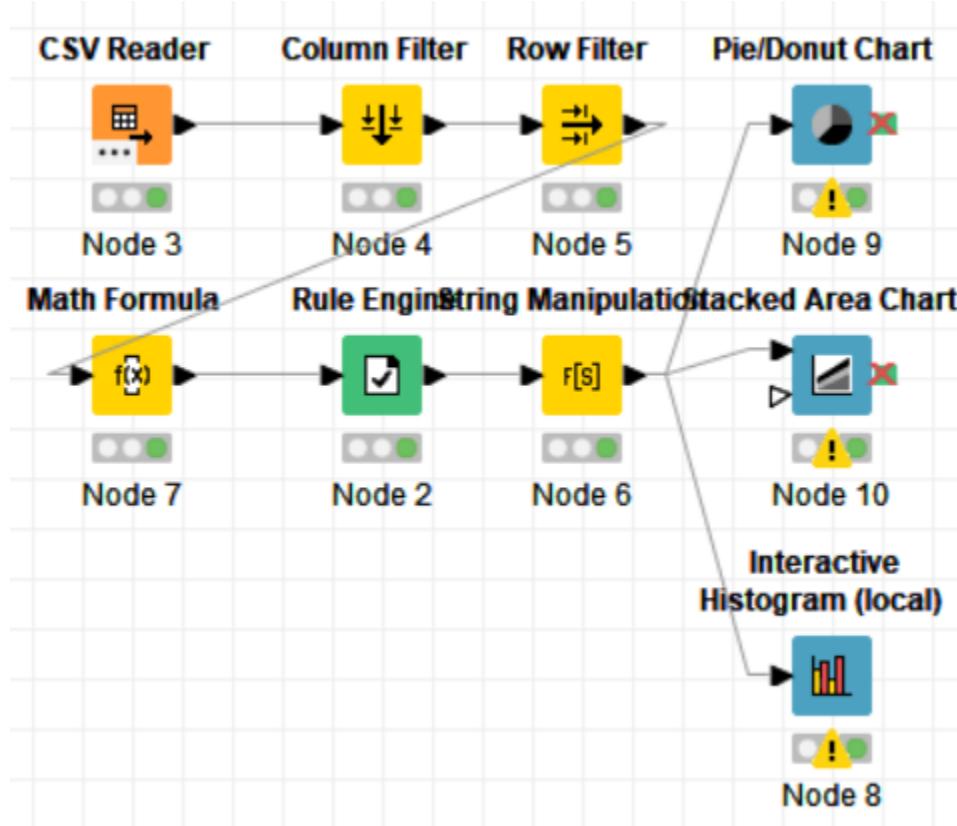
THEORY:

CSV Files are a mine of data and information. However, even they possess data only in the raw form which is not of much use. In this case, the .csv file taken contains information about video game sales with columns such as regional sales, game genre, critics score and rating. These columns contain missing values as well and needs to be properly processed to be of use. Graphical outputs such as pie charts and histograms give a very good clue about the trend in the various patterns in the video game industry.

ALGORITHM:

The nodes required here are the csv root node with the .csv file, a column filter and row filter to filter out unnecessary values, math formula node for calculating an average rating for games from both critics and users, a rule engine to classify the game based on the ratings, a string manipulator to convert the year into a number and finally output nodes such as pie chart, interactive histogram and stacked area chart. The data is preprocessed up to the row engine after which it is graphically represented

FLOW DIAGRAM DESIGN:

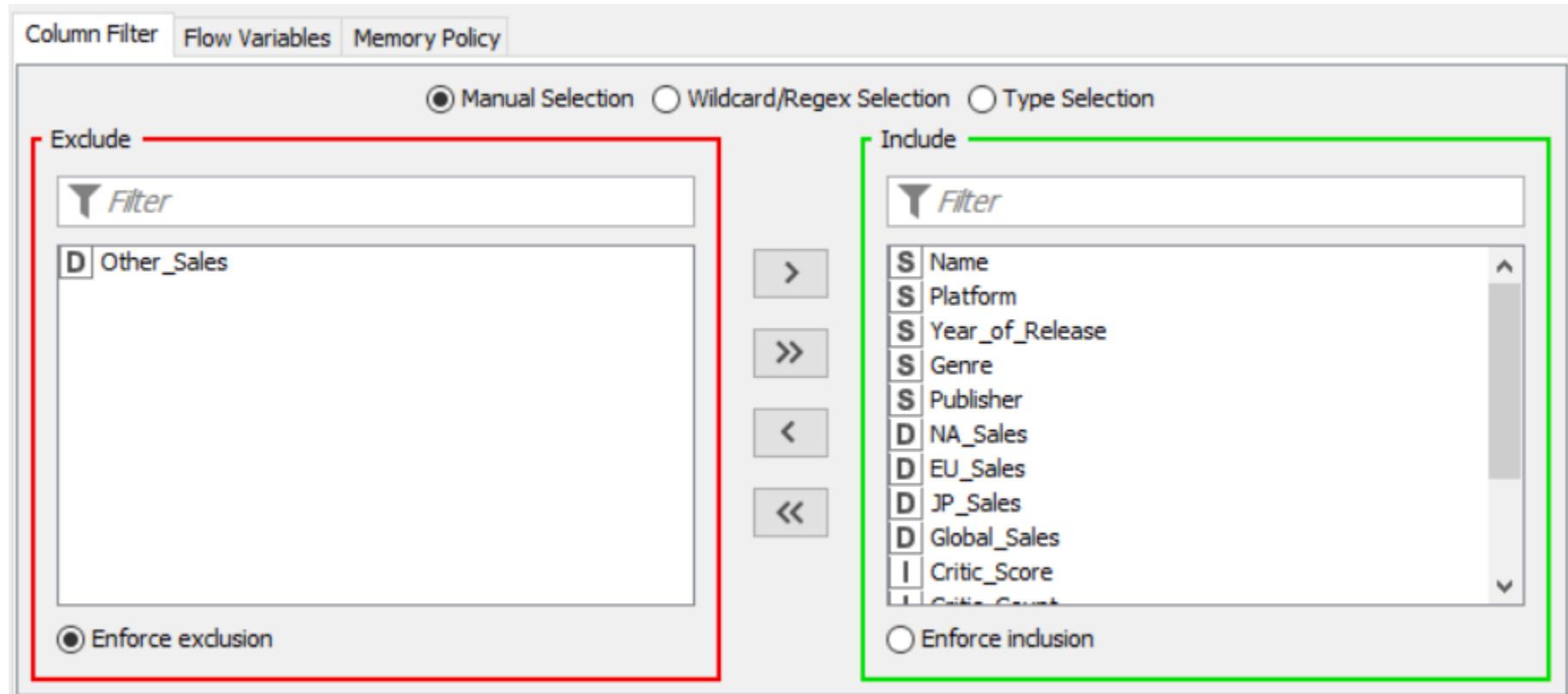


PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. Create a column filter, connect it to the CSV reader and filter the other_sales column. Connect said node to a row filter where all rows having genre as misc is removed by using pattern matching. Next, connect it to a Math Formula node where an average score is calculated by finding average of critics score and users score (users score is multiplied by 10 since the rating is out of 100). This is then passed onto a rule engine where a new column describing the rating is appended to the table (data) where the description is based upon the average rating. Then, a String Manipulator node is added to convert the values in

the year_of_release column to proper integers. Finally, pass the processed data to the pie chart node where it is segregated by genre, the stack area chart node where the average rating is displayed, and the interactive histogram where the sales details per platform is displayed.

PROPERTIES SET FOR EACH NODE WITH EXPLANATION:



[Filter Criteria](#)[Flow Variables](#)[Memory Policy](#)

Column value matching

Column to test: ▼ filter based on collection elements

Matching criteria

 use pattern matching ▼ ● case sensitive match contains wild cards regular expression use range checking

lower bound:

upper bound:

 only missing values match

- Include rows by attribute value
- Exclude rows by attribute value
- Include rows by number
- Exclude rows by number
- Include rows by row ID
- Exclude rows by row ID

Column List

ROWINDEX
ROWCOUNT
D NA_Sales
D EU_Sales
D JP_Sales
D Global_Sales
I Critic_Score
I Critic_Count
D User_Score
I User_Count

Category

All

Description

Multiplication between two numbers.

Function

isNaN(x)
isInfinite(x)
x + y
x - y
x ^ y
!x
+x
-x
x % y
x / y
x * y
x <= y
x >= y
x < y

Flow Variable List

Expression

1 average(\$Critic_Score\$, (\$User_Score\$*10))

 Append Column:

new column

 Replace Column:

S Rating

 Convert to Int

| Column List | Category | Description |
|--------------------------|--|---|
| <i>ROWID</i> | All | Checks whether the argument (a column) contains a missing value or not. |
| <i>ROWINDEX</i> | | |
| <i>ROWCOUNT</i> | | |
| S Name | | |
| S Platform | | |
| S Year_of_Release | | |
| S Genre | | |
| S Publisher | | |
| D NA_Sales | | |
| D EU_Sales | | |
| D JP_Sales | | |
| D Global_Sales | | |
| I Critic_Score | | |
| I Critic_Count | | |
| D User_Score | | |
| User_Count | | |
| Flow Variable List | | |
| s knime.workspace | | |
| | Function | |
| | ? < ? | |
| | ? <= ? | |
| | ? = ? | |
| | ? > ? | |
| | ? >= ? | |
| | ? AND ? | |
| | ? IN ? | |
| | ? LIKE ? | |
| | ? MATCHES ? | |
| | ? OR ? | |
| | ? XOR ? | |
| | FALSE | |
| | MISSING ? | |
| | NOT ? | |
| | Expression | |
| | | <pre> 1 // enter ordered set of rules, e.g.: 2 // \$double column name\$ > 5.0 => "Large" 3 // \$string column name\$ LIKE "*blue*" => "small and blue" 4 // TRUE => "default outcome" 5 \$new column\$ >= 80 =>"Good Game" 6 \$new column\$ >= 60 AND \$new column\$ <80 =>"Average Game" 7 \$new column\$ <= 60 =>"Bad Game" 8 MISSING \$new column\$=>"Not Rated" </pre> |
| | <input checked="" type="radio"/> Append Column: prediction S | |
| | <input type="radio"/> Replace Column: S Rating ▼ | |

| Column List | Category | Description |
|--|---|--|
| <i>ROWID</i>
<i>ROWINDEX</i>
<i>ROWCOUNT</i>
S Name
S Platform
S Year_of_Release
S Genre
S Publisher
D NA_Sales
D EU_Sales
D JP_Sales
n Global_Sales | All | Converts input to an integer value. |
| | Function | Examples:
toInt((String)null) = null
toInt(2.0) = 2
toInt("2") = 2
toInt("2.0") = null |
| | replaceOccurrences(str, oldstr, newstr) | |
| | reverse(str) | |
| | string(x) | |
| | strip(str...) | |
| | stripEnd(str...) | |
| | stripStart(str...) | |
| | substr(str, start) | |
| | substr(str, start, length) | |
| | toBoolean(x) | |
| | toDouble(x) | |
| | toEmpty(str...) | |
| | toInt(x) | |

Flow Variable List
 knime.workspace

Expression

```
1 toInt($Year_of_Release$)
```

Append Column:

Insert Missing As Null

Replace Column:

Syntax check on close

General Settings

 Generate imageCategory Column

Aggregation Method

 Occurrence Count Sum Average Report on missing values Include 'Missing values' categoryFrequency Column Process table in memory

Options Flow Variables

Rows to display:

Display all rows

No. of rows to display:

Column selection:

Binning column:

Aggregation column:

Available columns

- D Global_Sales
- I Critic_Score
- I Critic_Count
- D User_Score
- I User_Count
- D new column
- I new column (#1)

Aggregation columns

- NA_Sales
- EU_Sales
- JP_Sales

General Settings

 Generate image

Maximum number of rows

2,500

Choose column for x-axis new column (#1) Sort for x-axis column Manual Selection Wildcard/Regex Selection

Filter

| |
|-------------------|
| D NA_Sales |
| D EU_Sales |
| D JP_Sales |
| D Global_Sales |
| I Critic_Score |
| I Critic_Count |
| D User_Score |
| I User_Count |
| I new column (#1) |

Enforce exclusion

>
 >>
 <
 <<

Filter

| |
|--------------|
| D new column |
|--------------|

Enforce inclusion

RESULT:

The CSV data from the file was preprocessed and displayed using various graphical outputs.

Colour and Histogram Node Based Application

AIM: To use a color node and a histogram node to process and display data properly.

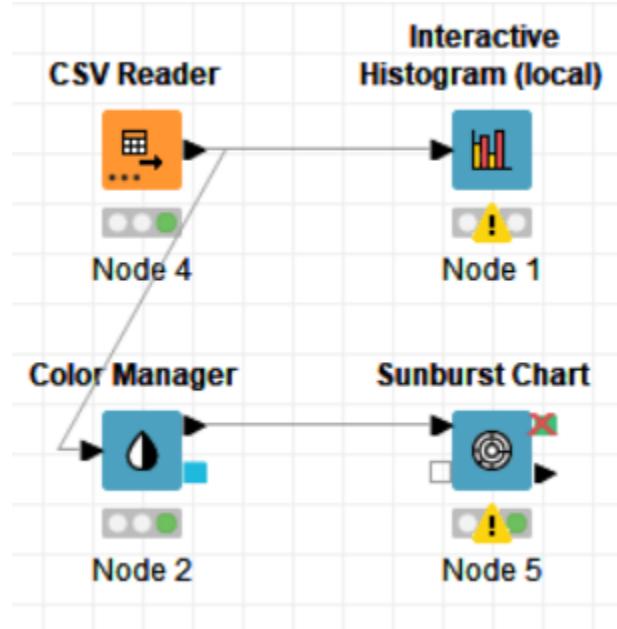
THEORY:

A histogram is a graphical display of data using bars of different heights. In a histogram, each bar groups numbers into ranges. Taller bars show that more data falls in that range. A histogram displays the shape and spread of continuous sample data. Colours can be crucial in displaying data clearly, as the same colour for different types can lead to confusion and unpredictability. A colour node comes with pre-set colours, or can be assigned to each data type for a lucid display.

ALGORITHM:

The nodes required here are the csv root node with the .csv file, a histogram node to display the data of user count with respect to the genre, a color manager node to assign each genre a significant color and finally a sunburst node to display the genre and sales side by side.

FLOW DIAGRAM DESIGN:



PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. Connect it to a color manager, where each genre can be assigned a specific color under swatches or a preset under palettes. A histogram is connected to the CSV reader to display the number of users to have rated a game per genre. Finally, attach a sunburst chart to the color manager node, taking sales and genre are the inputs and displaying the data properly. Compile and execute all the nodes at each stage of connection to get proper outputs.

PROPERTIES SET FOR EACH NODE WITH EXPLANATION:

Options Flow Variables

Rows to display:

Display all rows

No. of rows to display:

Column selection:

Binning column:

Aggregation column:

Available columns

- D NA_Sales
- D EU_Sales
- D JP_Sales
- D Other_Sales
- D Global_Sales
- I Critic_Score
- I Critic_Count
- D User_Score

Aggregation columns

- User_Count

Select one Column

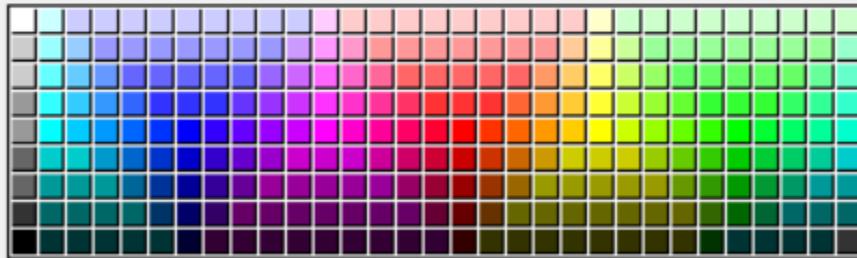
S | Genre

 Nominal

- Action
- Adventure
- Fighting
- Misc
- Platform
- Puzzle
- Racing
- Role-Playing
- Shooter
- Simulation
- Sports
- Strategy

 Range

Preview

[Palettes](#) [Swatches](#) [HSV](#) [HSL](#) [RGB](#) [CMYK](#) [Alpha](#)

Recent:



General Settings

Generate image

Maximum number of rows

Manual Selection Wildcard/Regex Selection



Name
 Year_of_Release
 Publisher
 Developer
 Rating

>
»
<
«



Platform
 Genre

Enforce exclusion

Enforce inclusion

Value Column

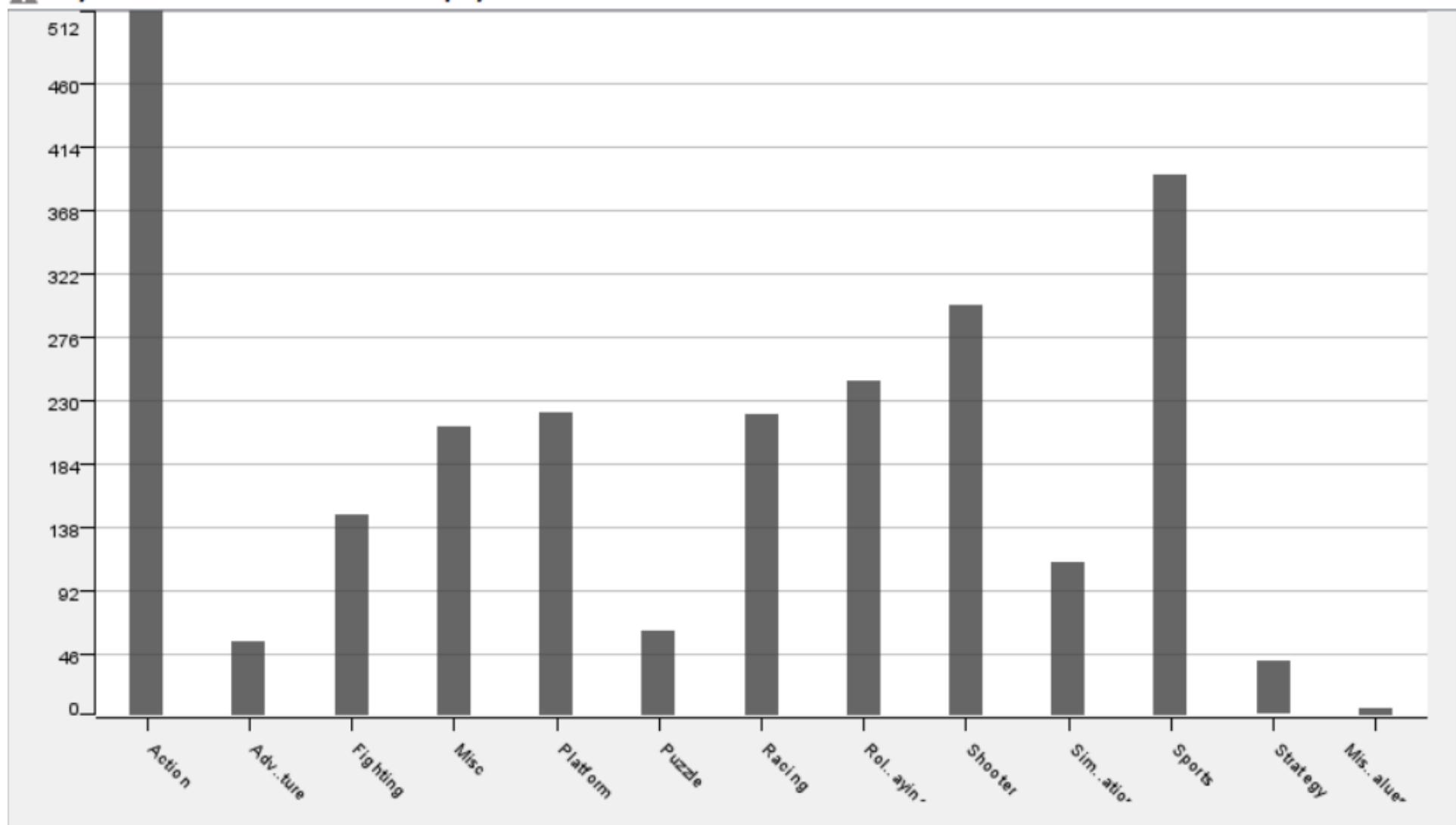
Filter out small nodes

Threshold for filtering (radians)

OUTPUT:

SNAPSHOTS:

⚠ Only the first 2500 of 16719 rows are displayed.



[Default Settings](#) [Column/Aggregation settings](#) [Bin settings](#) [Visualization settings](#) [Details](#)

Mouse Mode

Selection

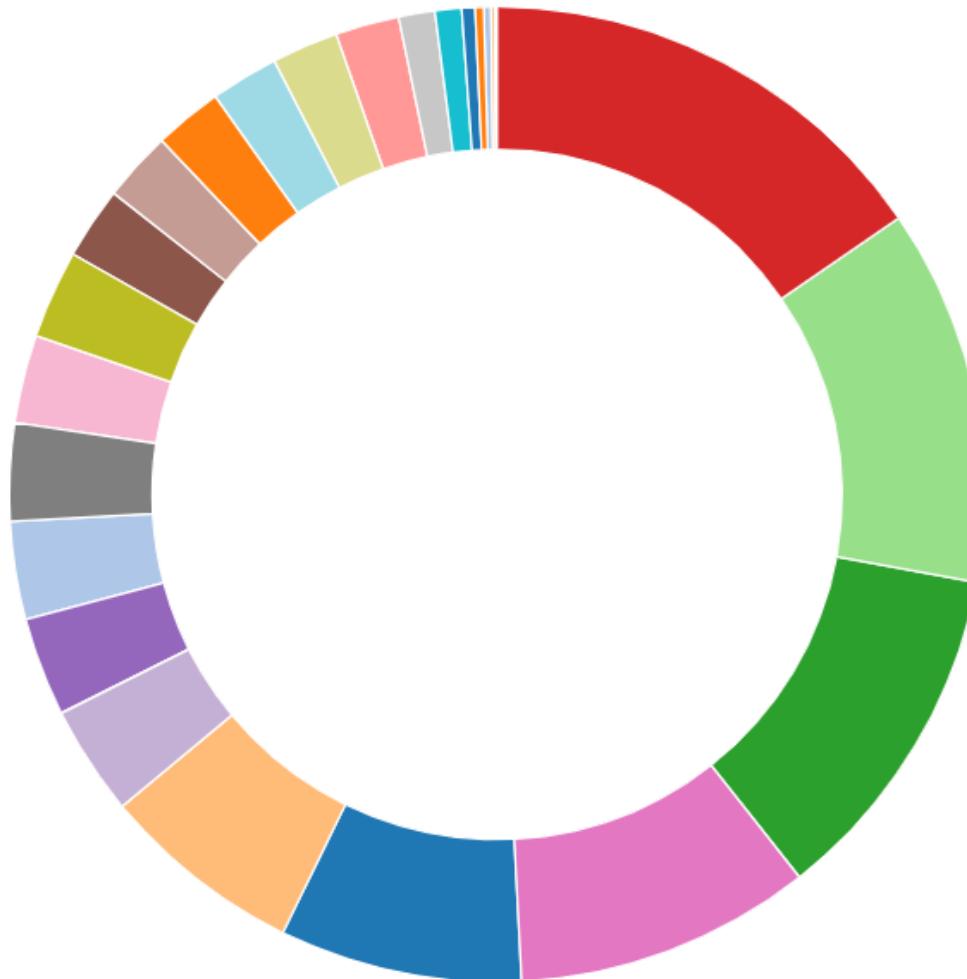


Fit to size

Background Color

Use anti-aliasing

Sunburst Chart



- Wii
- NES
- GB
- DS
- X360
- PS3
- PS2
- SNES
- GBA
- PS4
- 3DS
- N64
- PS
- XB
- PC
- 2600
- PSP
- XOne
- WiiU
- GC
- GEN
- DC
- PSV
- SAT
- SCD
- Sports
- Platform

RESULT:

The color manager node and histogram node were both created, compiled and executed upon a dataset and results were obtained as shown in the output.

Decision Tree Learner and Predictor

AIM: To use a decision tree learner and predictor and perform predictive analysis on given data

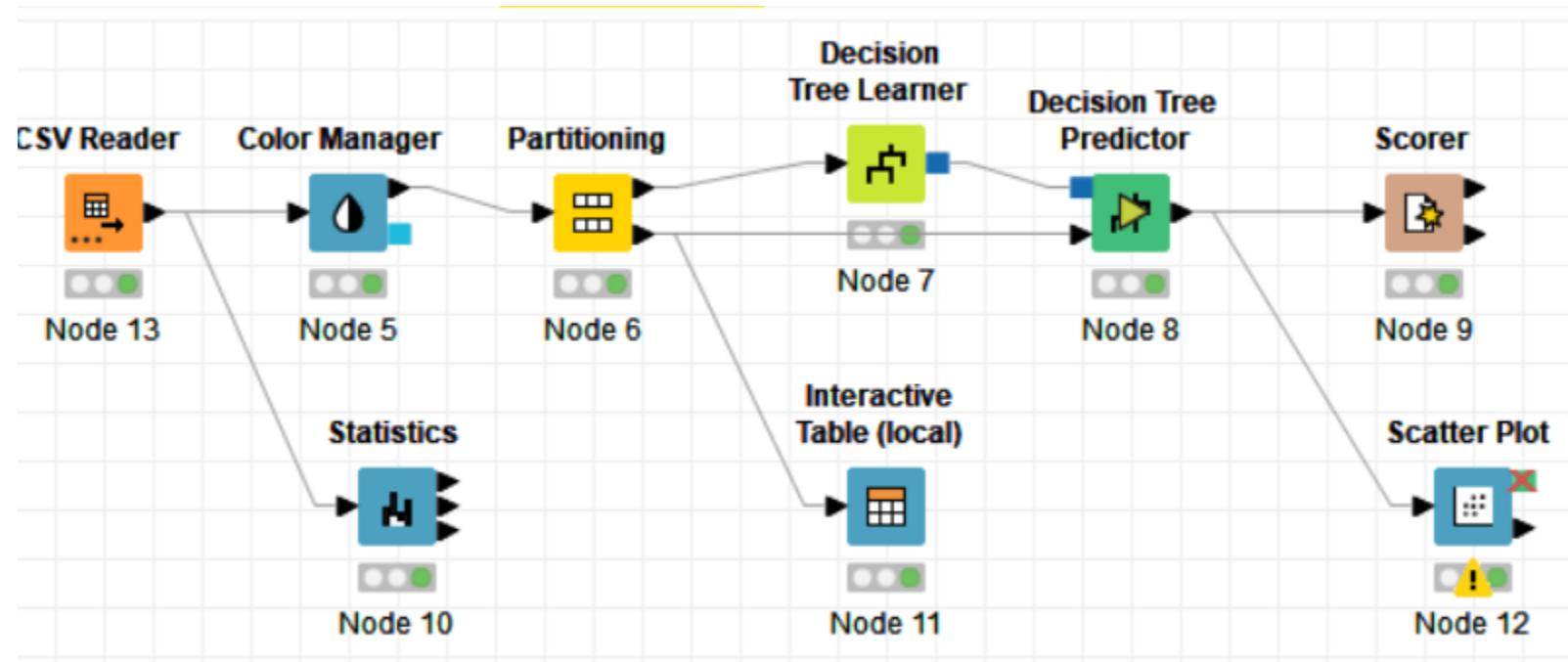
THEORY:

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are used for handling non-linear data sets effectively. The decision tree tool is used in real life in many areas, such as engineering, civil planning, law, and business.

ALGORITHM:

The nodes required here are the csv root node with the .csv file, color manager node to assign colors, statistics node to show the statistical output, partitioning node to partition the data, decision tree learner to work on some sample data to classify the decision tree, decision tree predictor to predict values from the existing pattern, scorer to compare rows and assign a score, interactive table to see how the partitioning works and scatter plot to get output of decision tree prediction

FLOW DIAGRAM DESIGN:



PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. Connect it to a color manager, where each genre can be assigned a specific color under swatches or a preset under palettes. Feed the input csv data to a statistics node to obtain a histogram showing the input data. Partition the data at an 80:20 ratio. Send the smaller partition to the decision tree learner, where income is compared against quality measure in order to classify the values, which can then be predicted by the decision tree predictor. Finally, send the output to the scatter plot node. Compile and execute all the nodes at each stage of connection to get proper outputs.

PROPERTIES SET FOR EACH NODE WITH EXPLANATION:

Select one Column

S income

Nominal

 <=50K
 >50K

Range

Preview

Palettes Swatches HSV HSL RGB CMYK Alpha

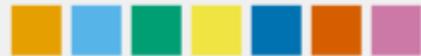
Set 1



Set 2



Set 3 (colorblind safe)



Custom

Calculate median values (computationally expensive)

Nominal values

 Manual Selection Wildcard/Regex Selection Type Selection

Exclude



I fnlwgt
I education-num
S race
I capital-gain
I capital-loss

 Enforce exclusion

Include



I age
S workclass
S education
S marital-status
S occupation
S relationship
S sex
I hours-per-week
S native-country
S income

 Enforce inclusionMax no. of most frequent and infrequent values (in view): Max no. of possible values per column (in output table): Enable HiLite

Choose size of first partition

 Absolute100 Relative[%]80 Take from top Linear sampling Draw randomly Stratified sampling income Use random seed

1,622,475,724,1

General

Class column Quality measure Pruning method Reduced Error PruningMin number records per node Number records to store for view Average split pointNumber threads Skip nominal columns without domain information

Root split

 Force root split columnRoot split column

Binary nominal splits

 Binary nominal splitsMax #nominal Filter invalid attribute values in child nodes

Options Flow Variables Memory Policy

Maximum number of stored patterns for HiLite-ing:

10,000

Change prediction column name

Prediction (income)

Append columns with normalized class distribution

Suffix for probability columns

Scorer Flow Variables Memory Policy

First Column

\$ income

Second Column

\$ Prediction (income)

Sorting of values in tables

Sorting strategy: Insertion order Reverse order

Provide scores as flow variables

Use name prefix

Missing values

In case of missing values: Ignore

Fail

Create image at output

Maximum number of rows:

2,500

Selection column name: Selected (Scatter Plot)

Choose column for x axis

age

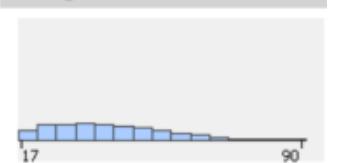
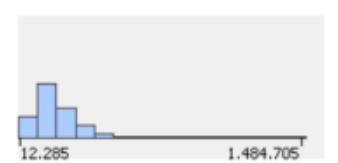
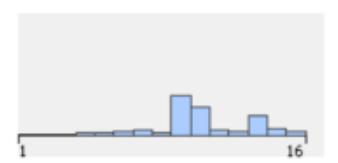
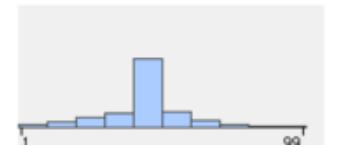
Choose column for y axis

hours-per-week

Report on missing values

OUTPUT:

SNAPSHOTS:

| Column | Min | Mean | Median | Max | Std. Dev. | Skewness | Kurtosis | No. Missing | No. $+\infty$ | No. $-\infty$ | Histogram |
|----------------|--------|--------------|--------|-----------|--------------|----------|----------|-------------|---------------|---------------|--|
| age | 17 | 38.5816 | ? | 90 | 13.6404 | 0.5587 | -0.1661 | 0 | 0 | 0 |  |
| fnlwgt | 12,285 | 189,778.3665 | ? | 1,484,705 | 105,549.9777 | 1.447 | 6.2188 | 0 | 0 | 0 |  |
| education-num | 1 | 10.0807 | ? | 16 | 2.5727 | -0.3117 | 0.6234 | 0 | 0 | 0 |  |
| capital-gain | 0.0 | 1,077.6488 | ? | 99,999 | 7,385.2921 | 11.9538 | 154.7994 | 0 | 0 | 0 |  |
| capital-loss | 0.0 | 87.3038 | ? | 4,356 | 402.9602 | 4.5946 | 20.3768 | 0 | 0 | 0 |  |
| hours-per-week | 1 | 40.4375 | ? | 99 | 12.3474 | 0.2276 | 2.9167 | 0 | 0 | 0 |  |

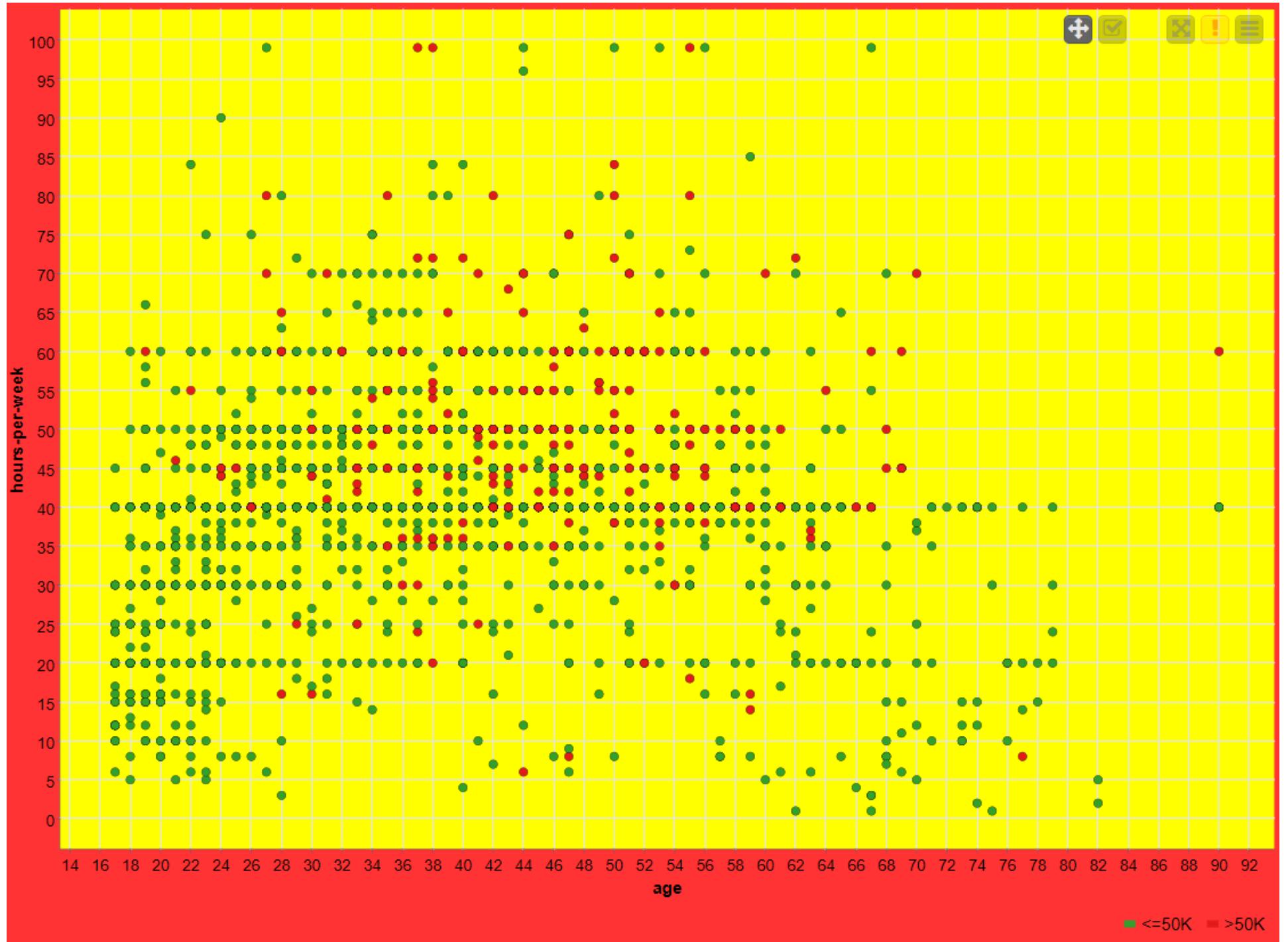
| Row ID | age | workclass | fnlwgt | education | education-num | marital-status | occupation | relationship | race | sex | capital-gain | capital-loss | hours-per-week | native-country | income |
|--------|-----|-------------|--------|--------------|---------------|-----------------------|-------------------|----------------|--------------------|--------|--------------|--------------|----------------|----------------|--------|
| Row2 | 38 | Private | 215646 | HS-grad | 9 | Divorced | Handlers-cleaners | Not-in-family | White | Male | 0 | 0 | 40 | United-States | <=50K |
| Row6 | 49 | Private | 160187 | 9th | 5 | Married-spouse-absent | Other-service | Not-in-family | Black | Female | 0 | 0 | 16 | Jamaica | <=50K |
| Row9 | 42 | Private | 159449 | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | White | Male | 5178 | 0 | 40 | United-States | >50K |
| Row11 | 30 | State-gov | 141297 | Bachelors | 13 | Married-civ-spouse | Prof-specialty | Husband | Asian-Pac-Islander | Male | 0 | 0 | 40 | India | >50K |
| Row12 | 23 | Private | 122272 | Bachelors | 13 | Never-married | Adm-clerical | Own-child | White | Female | 0 | 0 | 30 | United-States | <=50K |
| Row14 | 40 | Private | 121772 | Assoc-voc | 11 | Married-civ-spouse | Craft-repair | Husband | Asian-Pac-Islander | Male | 0 | 0 | 40 | ? | >50K |
| Row15 | 34 | Private | 245487 | 7th-8th | 4 | Married-civ-spouse | Transport-moving | Husband | Amer-Indian-Taiwan | Male | 0 | 0 | 45 | Mexico | <=50K |
| Row20 | 40 | Private | 193524 | Doctorate | 16 | Married-civ-spouse | Prof-specialty | Husband | White | Male | 0 | 0 | 60 | United-States | >50K |
| Row27 | 54 | ? | 180211 | Some-college | 10 | Married-civ-spouse | ? | Husband | Asian-Pac-Islander | Male | 0 | 0 | 60 | South | >50K |
| Row28 | 39 | Private | 367260 | HS-grad | 9 | Divorced | Exec-managerial | Not-in-family | White | Male | 0 | 0 | 80 | United-States | <=50K |
| Row33 | 30 | Federal-gov | 59951 | Some-college | 10 | Married-civ-spouse | Adm-clerical | Own-child | White | Male | 0 | 0 | 40 | United-States | <=50K |
| Row34 | 22 | State-gov | 311512 | Some-college | 10 | Married-civ-spouse | Other-service | Husband | Black | Male | 0 | 0 | 15 | United-States | <=50K |
| Row35 | 48 | Private | 242406 | 11th | 7 | Never-married | Machine-op-inspct | Unmarried | White | Male | 0 | 0 | 40 | Puerto-Rico | <=50K |
| Row36 | 21 | Private | 197200 | Some-college | 10 | Never-married | Machine-op-inspct | Own-child | White | Male | 0 | 0 | 40 | United-States | <=50K |
| Row40 | 31 | Private | 507875 | 9th | 5 | Married-civ-spouse | Machine-op-inspct | Husband | White | Male | 0 | 0 | 43 | United-States | <=50K |
| Row58 | 41 | Private | 147372 | HS-grad | 9 | Married-civ-spouse | Adm-clerical | Husband | White | Male | 0 | 0 | 48 | United-States | <=50K |
| Row63 | 42 | Private | 116632 | Doctorate | 16 | Married-civ-spouse | Prof-specialty | Husband | White | Male | 0 | 0 | 45 | United-States | >50K |
| Row65 | 36 | Private | 155537 | HS-grad | 9 | Married-civ-spouse | Craft-repair | Husband | White | Male | 0 | 0 | 40 | United-States | <=50K |
| Row66 | 28 | Private | 183175 | Some-college | 10 | Divorced | Adm-clerical | Not-in-family | White | Female | 0 | 0 | 40 | United-States | <=50K |
| Row74 | 79 | Private | 124744 | Some-college | 10 | Married-civ-spouse | Prof-specialty | Other-relative | White | Male | 0 | 0 | 20 | United-States | <=50K |
| Row75 | 27 | Private | 213921 | HS-grad | 9 | Never-married | Other-service | Own-child | White | Male | 0 | 0 | 40 | Mexico | <=50K |
| Row78 | 18 | Private | 309634 | 11th | 7 | Never-married | Other-service | Own-child | White | Female | 0 | 0 | 22 | United-States | <=50K |
| Row81 | 52 | Private | 276515 | Bachelors | 13 | Married-civ-spouse | Other-service | Husband | White | Male | 0 | 0 | 40 | Cuba | <=50K |
| Row87 | 33 | Private | 202051 | Masters | 14 | Married-civ-spouse | Prof-specialty | Husband | White | Male | 0 | 0 | 50 | United-States | <=50K |
| Row94 | 34 | Local-gov | 226296 | Bachelors | 13 | Married-civ-spouse | Protective-serv | Husband | White | Male | 0 | 0 | 40 | United-States | >50K |

<

>

File Hilite

| income \ Pr... | <=50K | >50K |
|----------------|-------|------|
| <=50K | 4484 | 428 |
| >50K | 601 | 1000 |



RESULT:

The decision tree learner, decision tree predictor and scorer were used to perform predictive analysis on the data

K-Means Clustering

AIM: To use k-means and cluster assigner nodes to form and identify clusters from the given data

THEORY:

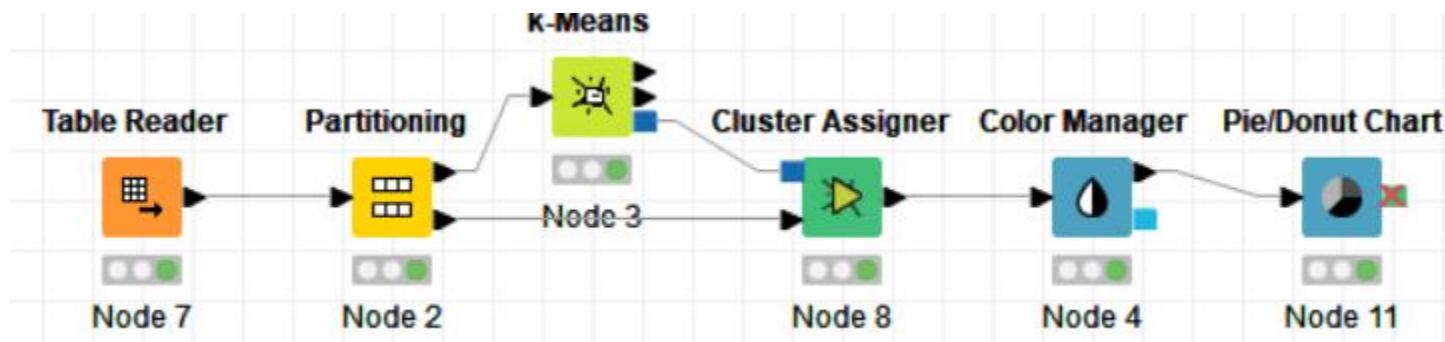
A K-means clustering is a type of unsupervised, which is used when you have unlabelled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K . The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

ALGORITHM:

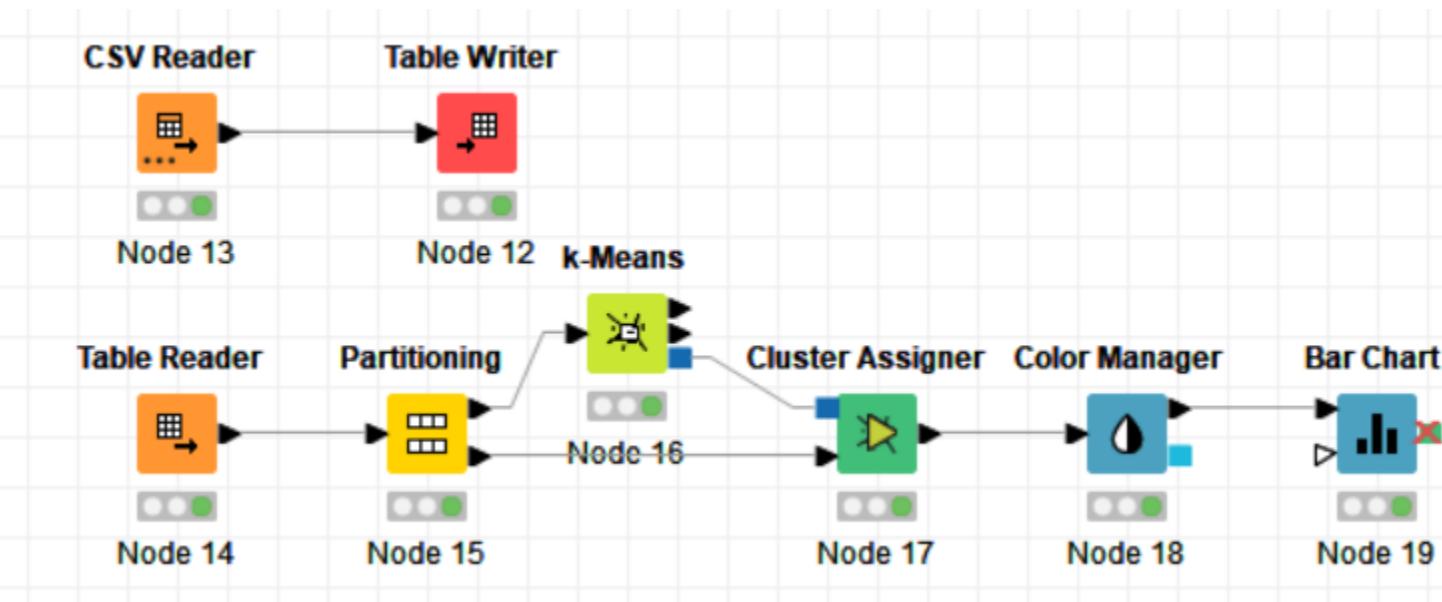
The nodes required here are the csv root node with the .csv file, color manager node to assign colors, partitioning node to partition the data, k-means node to apply k-means clustering, cluster assigner to assign the clusters and pie chart/bar chart to get output of clustering

FLOW DIAGRAM DESIGN:

Flow 1:



Flow 2:



PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. Connect to a table writer to create a table, which is then read in by a table reader node OR directly import a table using the table reader node. Connect it to the partitioning node, partition it at a 70:30 ratio. Pass the data to the k-means node where the k-means algorithm is applied according to the columns given. Connect this to a cluster assigner node to assign cluster. Link this to the color manager node to assign colors and finally pass it to the bar/pie chart to get output. Compile and execute all the nodes at each stage of connection to get proper outputs.

PROPERTIES SET FOR EACH NODE WITH EXPLANATION:

Flow 1:

Choose size of first partition

Absolute

100

Relative[%]

70

Take from top

Linear sampling

Draw randomly

Stratified sampling

Sentiment Analysis

Use random seed

1,622,476,457,

Clusters

Number of clusters:

Centroid initialization:

- First k rows
 Random initialization Use static random seed

Number of Iterations

Max. number of iterations:

Column Selection

Exclude



Include



Always include all columns

Hilite Mapping

Enable Hilite Mapping

Flow Variables

Memory Policy

Select memory policy for data outport(s)

- Cache tables in memory.
- Write tables to disc.

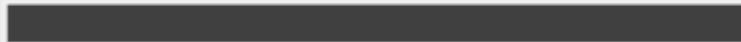
Select one Column

 S | Cluster Nominal

- cluster_0
- cluster_1
- cluster_2
- cluster_3
- cluster_4

 Range

Preview

[Palettes](#) [Swatches](#) [HSV](#) [HSL](#) [RGB](#) [CMYK](#) [Alpha](#) Set 1 Set 2 Set 3 (colorblind safe) Custom

General Settings

Generate image

Category Column

Aggregation Method

Occurrence Count Sum Average

Report on missing values

Include 'Missing values' category

Frequency Column

Process table in memory

File

Options Flow Variables

Output location:

C:\Users\Abhilash\OneDrive\Documents\adult.table

Warning: output file exists

Overwrite OK

Clusters

Number of clusters:

Centroid initialization:

 First k rows Random initialization Use static random seed

Number of Iterations

Max. number of iterations:

Column Selection

Exclude



fnlwgt
 capital-gain
 capital-loss

Include



age
 education-num
 hours-per-week

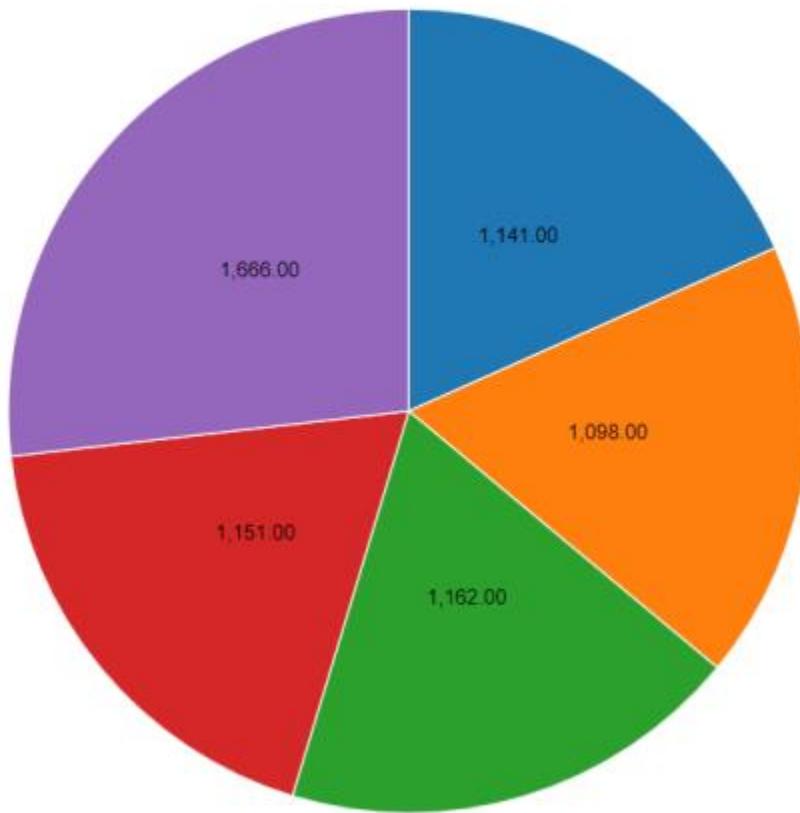
 Always include all columns

OUTPUT:

SNAPSHOTS:

Flow:

Pie Chart



RESULT:

The k-means and cluster assigner node were used to create data clusters and analyze them

Application with Knime/ World Population Analysis

AIM: To create an application to preprocess and analyze data (population and growth of various countries) using various nodes from Knime.

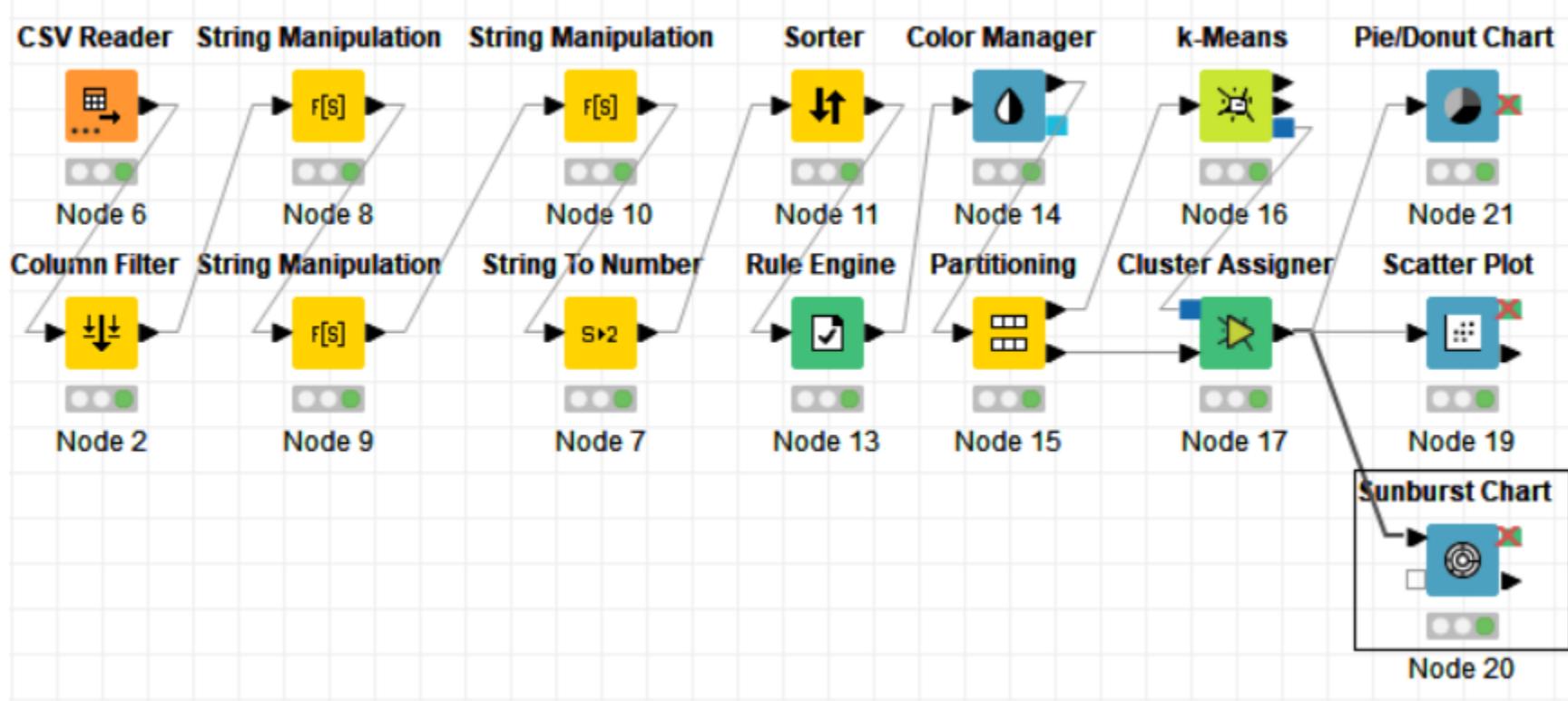
THEORY:

CSV Files are a very valuable source of untapped data. But they possess data in tabular columns, often with wrong data types (numbers being considered as strings, for example) which is not of much use. In this case, the .csv file taken contains information about the populations of various countries with columns such as population in 2020, latest update population, growth rate, area, population density and index. Graphical outputs such as pie charts and scatter plots give a very good clue about the population growth rate and percentage per countries.

ALGORITHM:

The nodes required here are the csv root node with the .csv file, a column filter to filter out unnecessary values, a string manipulator and string to number to help convert three numbers of string type to double type, a rule engine to classify the countries based on growth, a sorter to sort the table, color manager to assign colors, partitioning node to partition the data, k-means and cluster assigner node for assigning clusters and finally output nodes such as pie chart, scatter plot and sunburst chart.

FLOW DIAGRAM DESIGN:



PROCEDURE:

Drag the .csv file into knime to create and initialize the CSV Reader node. Create a column filter, connect it to the CSV reader and filter the iso_code column. Then three String Manipulator nodes are added, each removing the %,sq_km and /sq_km non numerical portions of the growth_rate, area and density_sq_km columns. This is then passed to a string to number node where the above-mentioned columns are all converted to numbers. A sorter node is used to sort the data in descending order with respect to the 2021_last_updated and 2020_population columns. This is then passed onto a rule engine where a new column describing the growth rate is appended to the table (data) where the description is based upon the growth rate parameter. This is then passed onto the color manager where specific colors have been assigned to the values of the newly appended column describing growth rate. Then, a partitioning node is used to partition the dataset to a 70:30 ratio, with the larger partition sent to a k-means node. The k-means node divides the given data into 5 clusters (in this case), which is then assigned using the cluster assigner. Finally, pit is passed to the pie chart node where it is shown as countries by highest population, the scatter plot where the countries are displayed by growth rate, and the sunburst chart where the clusters, population, and country are depicted concentrically.

PROPERTIES SET FOR EACH NODE:

Column Filter

[Column Filter](#)[Flow Variables](#)[Memory Policy](#)

Manual Selection Wildcard/Regex Selection Type Selection

Exclude

 S iso_code Enforce exclusion

Include



- S country
- S 2021_last_updated
- S 2020_population
- S area
- S density_sq_km
- S growth_rate
- D world_%
- I rank

 Enforce inclusion**String Manipulator for growth rate**

Column List

- ROWID
- ROWINDEX
- ROWCOUNT
- S country
- S 2021_last_updated
- S 2020_population
- S area
- S density_sq_km
- S growth_rate
- D world_%
- I rank

Flow Variable List

- s knime.workspace

| Category | Description |
|------------|--|
| All | |
| Function | <ul style="list-style-type: none">capitalize(str)capitalize(str, chars)compare(str1, str2)count(str, toCount)count(str, toCount, modifiers)countChars(str, chars)countChars(str, chars, modifiers)indexOf(str, toSearch)indexOf(str, toSearch, modifiers)indexOf(str, toSearch, start)indexOf(str, toSearch, start, modifiers) |
| Expression | <pre>1 removeChars(\$growth_rate\$, "%")</pre> |

Append Column:

Replace Column:

Insert Missing As Null

Syntax check on close

String Manipulator for area

| Column List | Category | Description |
|--|--|-------------|
| <i>ROWID</i>
<i>ROWINDEX</i>
<i>ROWCOUNT</i>
S country
S 2021_last_updated
S 2020_population
S area
S density_sq_km
S growth_rate
D world_%
I rank | All | |
| | Function | |
| | capitalize(str)
capitalize(str, chars)
compare(str1, str2)
count(str, toCount)
count(str, toCount, modifiers)
countChars(str, chars)
countChars(str, chars, modifiers)
indexOf(str, toSearch)
indexOf(str, toSearch, modifiers)
indexOf(str, toSearch, start)
indexOf(str, toSearch, start, modifiers) | |
| | Expression | |
| | 1 removeChars(\$area\$, "sq_km") | |

Append Column:

Replace Column:

Insert Missing As Null

Syntax check on close

String Manipulator for density per square kilometer

| Column List | Category | Description |
|--|------------|---|
| <i>ROWID</i>
<i>ROWINDEX</i>
<i>ROWCOUNT</i>
S country
S 2021_last_updated
S 2020_population
S area
S density_sq_km
S growth_rate
D world_%
I rank | All | |
| Flow Variable List
knime.workspace | Function | <pre>capitalise(str) capitalise(str, chars) compare(str1, str2) count(str, toCount) count(str, toCount, modifiers) countChars(str, chars) countChars(str, chars, modifiers) indexOf(str, toSearch) indexOf(str, toSearch, modifiers) indexOf(str, toSearch, start) indexOf(str, toSearch, start, modifiers)</pre> |
| | Expression | <pre>1 removeChars(\$density_sq_km\$, "/sq_km")</pre> |
| | | <input type="radio"/> Append Column: <input type="text"/>
<input checked="" type="radio"/> Replace Column: <input type="text" value="S density_sq_km"/> |
| | | <input type="checkbox"/> Insert Missing As Null
<input checked="" type="checkbox"/> Syntax check on close |

String to Number

Parsing options

Type:

 Number (double) ▾

Decimal separator:

Thousands separator:

 Accept type suffix, e.g. 'd', 'D', 'f', 'F' Manual Selection Wildcard/Regex Selection

Exclude

 country Enforce exclusion

Include

 2021_last_updated
 2020_population
 area
 density_sq_km
 growth_rate Enforce inclusion

Sorter

[Sorting Filter](#)[Advanced Settings](#)[Flow Variables](#)[Memory Policy](#)**Sort by**A text input field containing the letter 'D' followed by the text '2020_population'. A small downward-pointing arrow is located at the right end of the field. Ascending Descending**Next by**A text input field containing the letter 'D' followed by the text '2021_last_updated'. A small downward-pointing arrow is located at the right end of the field. Ascending Descending

A small gray square icon with a white plus sign in the center, followed by the text 'Add Rule'.

Rule Engine

Rule Editor

Flow Variables

Memory Policy

| Column List | Category | Description |
|---------------------|----------|-------------|
| ROWID | All | |
| ROWINDEX | | |
| ROWCOUNT | | |
| S country | | |
| D 2021_last_updated | | |
| D 2020_population | | |
| D area | | |
| D density_sq_km | | |
| D growth_rate | | |
| D world_% | | |
| I rank | | |

Flow Variable List

- knime.workspace

| Category | Description |
|----------|---|
| Function | ? < ?
? <= ?
? = ?
? > ?
? >= ?
? AND ?
? IN ?
? LIKE ?
? MATCHES ?
? OR ?
? XOR ?
FALSE
MISSING ?
NOT ? |

Expression

```
? 4 // TRUE => "default outcome"
S 5 $growth_rate$ > 2 => "Very High growth rate"
S 6 $growth_rate$ < 2 AND $growth_rate$ > 1 => "High growth rate"
S 7 $growth_rate$ < 1 AND $growth_rate$ > 0 => "Mediocre growth rate"
S 8 $growth_rate$ < 0 => "Negative growth rate"
```

Append Column: S

Replace Column:

Colour Manager

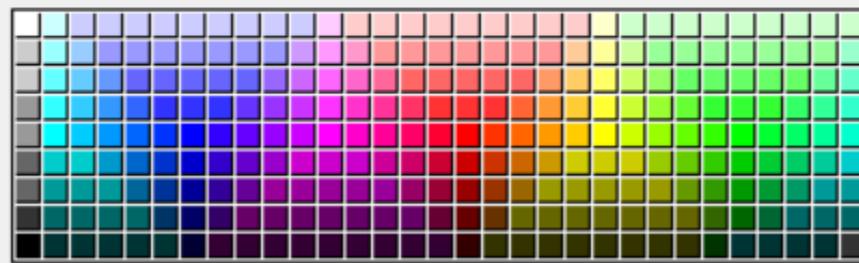
Select one Column

S | growth_rate_description

 Nominal Range

- High growth rate
- Mediocre growth rate
- Negative growth rate
- Very High growth rate

Preview

[Palettes](#) [Swatches](#) [HSV](#) [HSL](#) [RGB](#) [CMYK](#) [Alpha](#)

Recent:



Partitioning

Choose size of first partition

 Absolute Relative[%] Take from top Linear sampling Draw randomly Stratified sampling Use random seed

k-Means

Clusters

Number of clusters:  

Centroid initialization:

 First k rows Random initialization Use static random seed

0

Number of Iterations

Max. number of iterations: 

Column Selection

Exclude



- D area
- D density_sq_km
- D growth_rate
- D world_%
- I rank

Include



- D 2021_last_updated
- D 2020_population

 Always include all columns

Hilite Mapping

 Enable Hilite Mapping

Flow Variables **Memory Policy**

Select memory policy for data outport(s)

Cache tables in memory.
 Write tables to disc.

Pie/Donut Chart

General Settings

Generate image

Category Column ▾

Aggregation Method

Occurrence Count Sum Average

Report on missing values

Include 'Missing values' category

Frequency Column ▾

Process table in memory

Create image at outport

Maximum number of rows:

2,500

Selection column name:

Selected (Scatter Plot)

Choose column for x axis

S country

Choose column for y axis

D growth_rate

Report on missing values

Sunburst Chart

General Settings

 Generate imageMaximum number of rows Manual Selection Wildcard/Regex Selection

No columns in this list

 Enforce exclusion

S country
S growth_rate_description
S Cluster

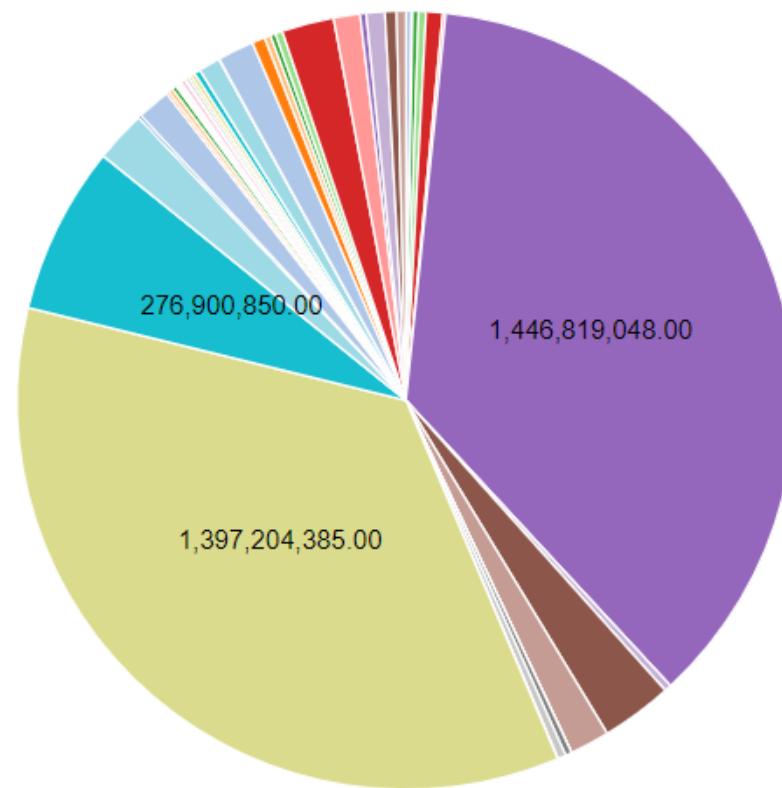
 Enforce inclusionValue Column Filter out small nodesThreshold for filtering (radians)

OUTPUT:

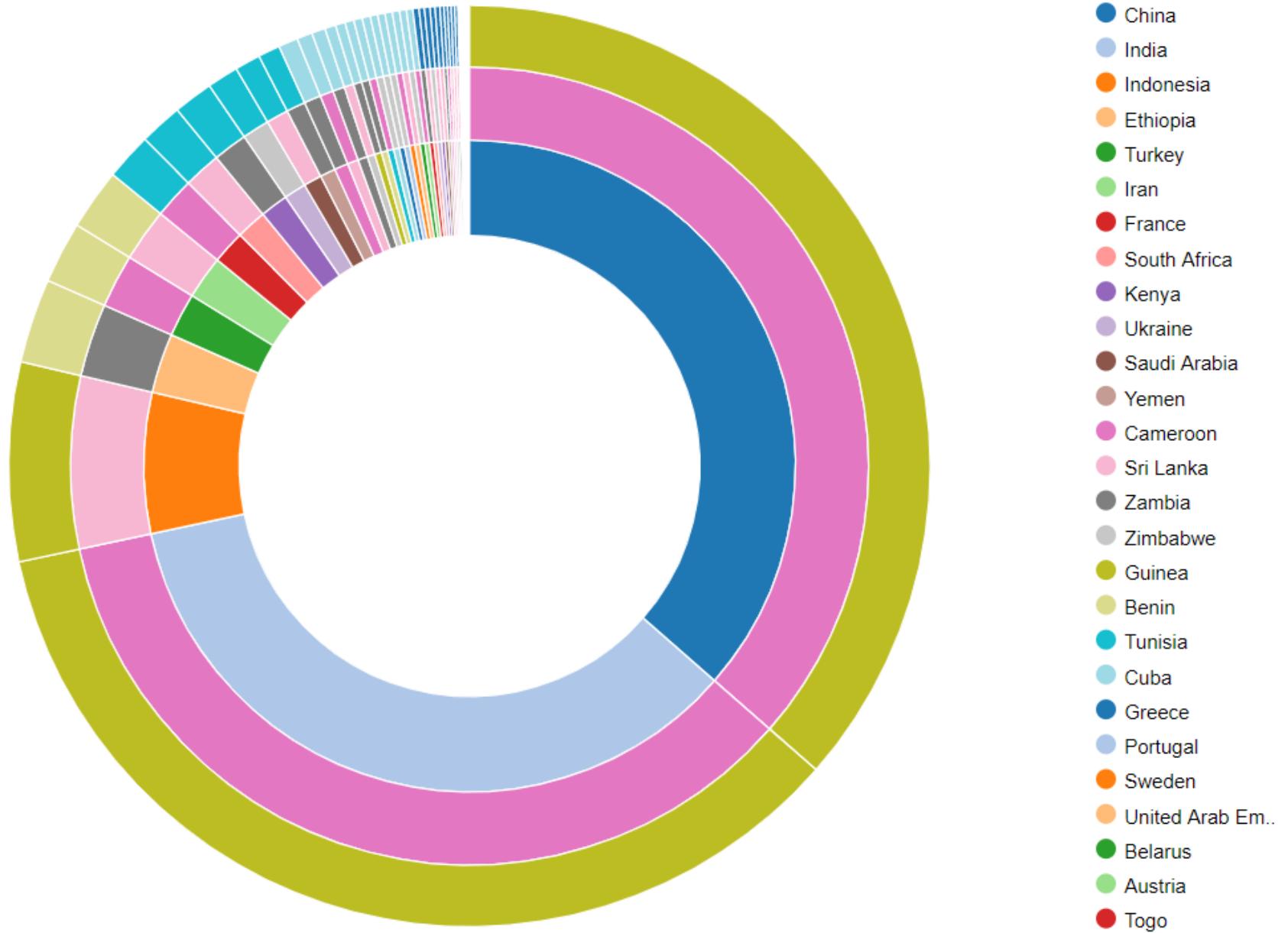
Pie Chart



| | | | | | | |
|--------------------------|----------|------------|----------|--------------|------------------|----------------------|
| Andorra | Austria | Bahrain | Barbados | Belarus | Benin | Cameroon |
| Central African Republic | China | Cuba | Ethiopia | France | French Polynesia | Gibraltar |
| Greece | Guinea | Guyana | India | Indonesia | Iran | Ireland |
| Kenya | Kuwait | Laos | Lebanon | Lithuania | Malta | Mauritius |
| Monaco | Mongolia | Montserrat | Namibia | Nauru | Nicaragua | North Macedonia |
| Oman | Panama | Paraguay | Portugal | Saudi Arabia | Solomon Islands | South Africa |
| Sri Lanka | Sweden | Togo | Tunisia | Turkey | Ukraine | United Arab Emirates |
| Yemen | Zambia | Zimbabwe | | | | |



Sunburst Chart



RESULT:

The CSV data from the file was preprocessed and displayed using various graphical outputs.

Chapter 8

Machine Learning – Regression Model Using Designer Studio

Azure ML Designer – Regression model

Reference: <https://learn.microsoft.com/en-us/training/>

Train a linear regression model that predicts car prices using the Azure Machine Learning designer.

Create the pipeline

Note

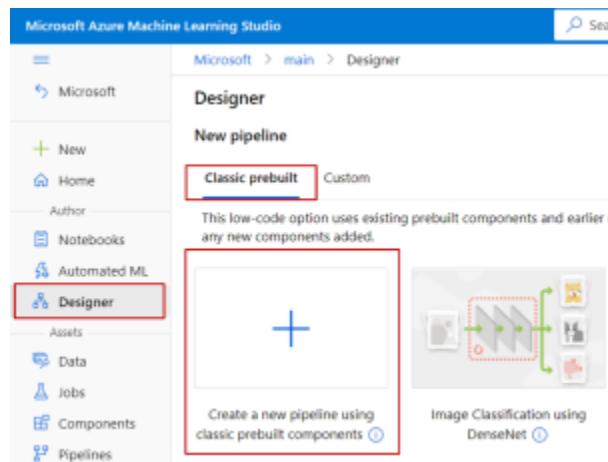
Designer supports two type of components, classic prebuilt components and custom components. These two types of components are not compatible.

Classic prebuilt components provides prebuilt components majorly for data processing and traditional machine learning tasks like regression and classification. This type of component continues to be supported but will not have any new components added.

Custom components allow you to provide your own code as a component. It supports sharing across workspaces and seamless authoring across Studio, CLI, and SDK interfaces.

This article applies to classic prebuilt components.

1. Sign in to ml.azure.com, and select the workspace you want to work with.
2. Select **Designer-> Classic prebuilt**



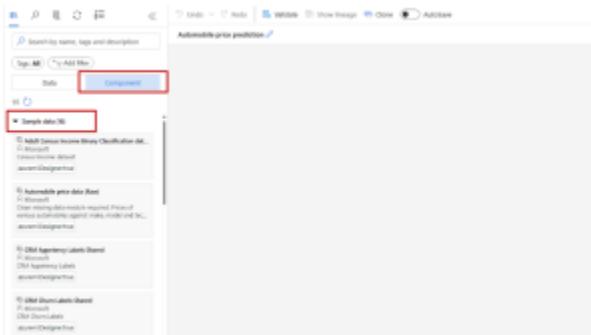
3. Select **Create a new pipeline using classic prebuilt components**.

4. Click the pencil icon beside the automatically generated pipeline draft name, rename it to *Automobile price prediction*. The name doesn't need to be unique.

Import data

There are several sample datasets included in the designer for you to experiment with. For this tutorial, use **Automobile price data (Raw)**.

1. To the left of the pipeline canvas is a palette of datasets and components. Select **Component-> Sample data**.
2. Select the dataset **Automobile price data (Raw)**, and drag it onto the canvas.



Visualize the data

You can visualize the data to understand the dataset that you'll use.

1. Right-click the **Automobile price data (Raw)** and select **Preview Data**.
2. Select the different columns in the data window to view information about each one.

Each row represents an automobile, and the variables associated with each automobile appear as columns. There are 205 rows and 26 columns in this dataset.

Prepare data

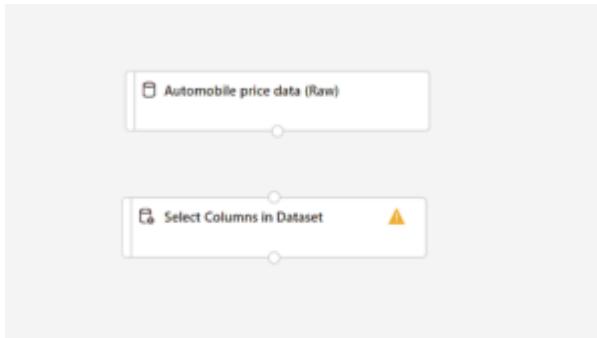
Datasets typically require some preprocessing before analysis. You might have noticed some missing values when you inspected the dataset. These missing values must be cleaned so that the model can analyze the data correctly.

Remove a column

When you train a model, you have to do something about the data that's missing. In this dataset, the **normalized-losses** column is missing many values, so you'll exclude that column from the model altogether.

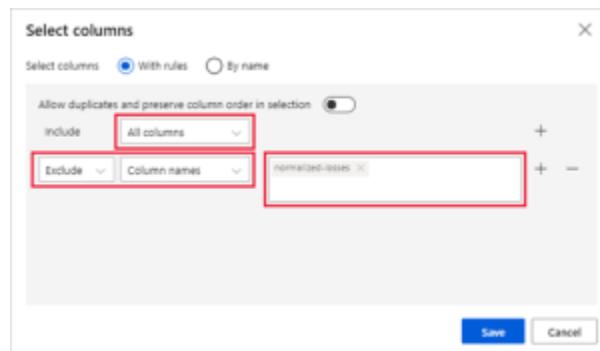
1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Select Columns in Dataset** component.
2. Drag the **Select Columns in Dataset** component onto the canvas. Drop the component below the dataset component.

3. Connect the **Automobile price data (Raw)** dataset to the **Select Columns in Dataset** component. Drag from the dataset's output port, which is the small circle at the bottom of the dataset on the canvas, to the input port of **Select Columns in Dataset**, which is the small circle at the top of the component.



You create a flow of data through your pipeline when you connect the output port of one component to an input port of another.

4. Select the **Select Columns in Dataset** component.
5. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Select Columns in Dataset** component to open the details pane.
6. Select **Edit column** to the right of the pane.
7. Expand the **Column names** drop down next to **Include**, and select **All columns**.
8. Select the **+** to add a new rule.
9. From the drop-down menus, select **Exclude** and **Column names**.
10. Enter *normalized-losses* in the text box.
11. In the lower right, select **Save** to close the column selector.



12. In the **Select Columns in Dataset** component details pane, expand **Node info**.
13. Select the **Comment** text box and enter *Exclude normalized losses*.

Comments will appear on the graph to help you organize your pipeline.

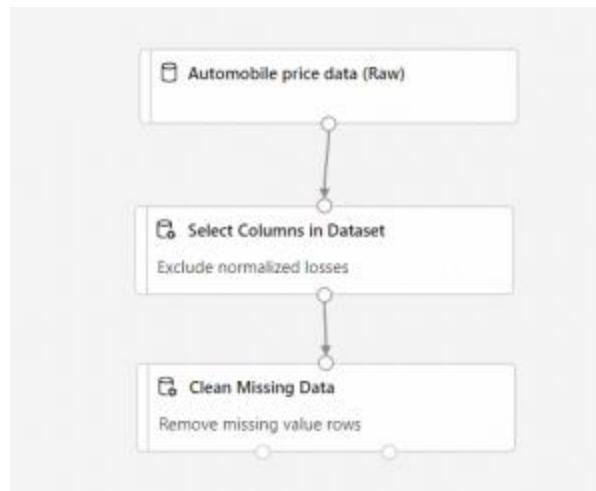
Clean missing data

Your dataset still has missing values after you remove the **normalized-losses** column. You can remove the remaining missing data by using the **Clean Missing Data** component.

In the datasets and component palette to the left of the canvas, click **Component** and search for the **Clean Missing Data** component.

1. Drag the **Clean Missing Data** component to the pipeline canvas. Connect it to the **Select Columns in Dataset** component.
2. Select the **Clean Missing Data** component.
3. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Clean Missing Data** component to open the details pane.
4. Select **Edit column** to the right of the pane.
5. In the **Columns to be cleaned** window that appears, expand the drop-down menu next to **Include**. Select **All columns**.
6. Select **Save**.
7. In the **Clean Missing Data** component details pane, under **Cleaning mode**, select **Remove entire row**.
8. In the **Clean Missing Data** component details pane, expand **Node info**.
9. Select the **Comment** text box and enter *Remove missing value rows*.

Your pipeline should now look something like this:



Train a machine learning model

Now that you have the components in place to process the data, you can set up the training components.

Because you want to predict price, which is a number, you can use a regression algorithm. For this example, you use a linear regression model.

Split the data

Splitting data is a common task in machine learning. You'll split your data into two separate datasets. One dataset trains the model and the other will test how well the model performed.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Split Data** component.
2. Drag the **Split Data** component to the pipeline canvas.
3. Connect the left port of the **Clean Missing Data** component to the **Split Data** component.

Important

Make sure that the left output port of **Clean Missing Data** connects to **Split Data**. The left port contains the cleaned data. The right port contains the discarded data.

4. Select the **Split Data** component.
5. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Split Data** component to open the details pane.
6. In the **Split Data** details pane, set the **Fraction of rows in the first output dataset** to 0.7.

This option splits 70 percent of the data to train the model and 30 percent for testing it. The 70 percent dataset will be accessible through the left output port. The remaining data is available through the right output port.

7. In the **Split Data** details pane, expand **Node info**.
8. Select the **Comment** text box and enter *Split the dataset into training set (0.7) and test set (0.3)*.

Train the model

Train the model by giving it a dataset that includes the price. The algorithm constructs a model that explains the relationship between the features and the price as presented by the training data.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Linear Regression** component.
2. Drag the **Linear Regression** component to the pipeline canvas.
3. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Train Model** component.
4. Drag the **Train Model** component to the pipeline canvas.
5. Connect the output of the **Linear Regression** component to the left input of the **Train Model** component.
6. Connect the training data output (left port) of the **Split Data** component to the right input of the **Train Model** component.

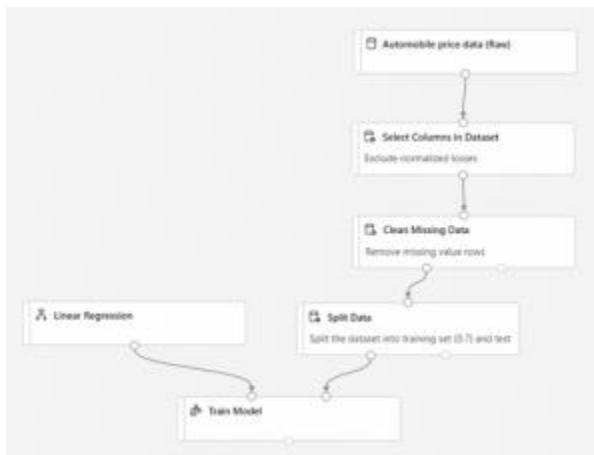
Important

Make sure that the left output port of **Split Data** connects to **Train Model**. The left port contains the training set. The right port contains the test set.



7. Select the **Train Model** component.
8. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Train Model** component to open the details pane.
9. Select **Edit column** to the right of the pane.
10. In the **Label column** window that appears, expand the drop-down menu and select **Column names**.
11. In the text box, enter *price* to specify the value that your model is going to predict.

Your pipeline should look like this:



Add the Score Model component

After you train your model by using 70 percent of the data, you can use it to score the other 30 percent to see how well your model functions.

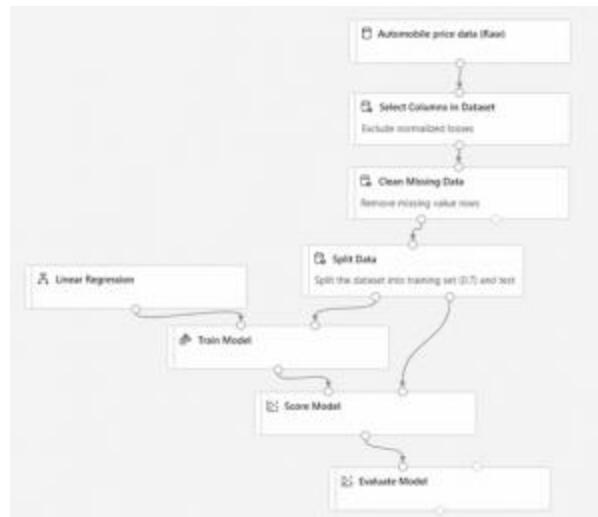
1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Score Model** component.
2. Drag the **Score Model** component to the pipeline canvas.
3. Connect the output of the **Train Model** component to the left input port of **Score Model**. Connect the test data output (right port) of the **Split Data** component to the right input port of **Score Model**.

Add the Evaluate Model component

Use the **Evaluate Model** component to evaluate how well your model scored the test dataset.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Evaluate Model** component.
2. Drag the **Evaluate Model** component to the pipeline canvas.
3. Connect the output of the **Score Model** component to the left input of **Evaluate Model**.

The final pipeline should look something like this:



Submit pipeline

1. Select **Configure & Submit** on the right top corner to submit the pipeline.
2. Then you'll see a step-by-step wizard, follow the wizard to submit the pipeline job.

Configure & Submit

Set up pipeline job

Basics

Experiment name Select existing Create new
Existing experiment *
Job display name create-pipeline-using-designer-dsc
Job description Pipeline created on 29/03/2024
Job tags Name: Value: Add

Review + Submit Back Next Close

In *Basics* step, you can configure the experiment, job display name, job description etc.

In *Inputs & Outputs* step, you can assign value to the Inputs/Outputs that are promoted to pipeline level. In this example it will be empty because we didn't promote any input/output to pipeline level.

In *Runtime settings*, you can configure the default datastore and default compute to the pipeline. It's the default datastore/compute to all components in the pipeline. However, if you set a different compute or datastore for a component explicitly, the system respects the component level setting. Otherwise, it uses the default.

The *Review + Submit* step is the last step to review all settings before submit. The wizard will remember your last configuration if you ever submit the pipeline.

After submitting the pipeline job, there will be a message on the top with a link to the job detail. You can select this link to review the job details.



View scored labels

In the job detail page, you can check the pipeline job status, results and logs.



After the job completes, you can view the results of the pipeline job. First, look at the predictions generated by the regression model.

1. Right-click the **Score Model** component, and select **Preview data > Scored dataset** to view its output.

Here you can see the predicted prices and the actual prices from the testing data.

Evaluate models

Use the **Evaluate Model** to see how well the trained model performed on the test dataset.

1. Right-click the **Evaluate Model** component and select **Preview data > Evaluation results** to view its output.

The following statistics are shown for your model:

- **Mean Absolute Error (MAE)**: The average of absolute errors. An error is the difference between the predicted value and the actual value.
- **Root Mean Squared Error (RMSE)**: The square root of the average of squared errors of predictions made on the test dataset.
- **Relative Absolute Error**: The average of absolute errors relative to the absolute difference between actual values and the average of all actual values.
- **Relative Squared Error**: The average of squared errors relative to the squared difference between the actual values and the average of all actual values.
- **Coefficient of Determination**: Also known as the R squared value, this statistical metric indicates how well a model fits the data.

For each of the error statistics, smaller is better. A smaller value indicates that the predictions are closer to the actual values. For the coefficient of determination, the closer its value is to one (1.0), the better the predictions.

Chapter 9

Multiple regression models to predict car prices

Train & compare multiple regression models to predict car prices with Azure Machine Learning designer

Learn how to build a machine learning pipeline without writing a single line of code using the designer. This sample trains and compares multiple regression models to predict a car's price based on its technical features. We'll provide the rationale for the choices made in this pipeline so you can tackle your own machine learning problems.

If you're just getting started with machine learning, take a look at the [basic version](#) of this pipeline.

Here's the completed graph for this pipeline:



Pipeline summary

Use following steps to build the machine learning pipeline:

1. Get the data.
2. Pre-process the data.
3. Train the model.
4. Test, evaluate, and compare the models.

Get the data

This sample uses the **Automobile price data (Raw)** dataset, which is from the UCI Machine Learning Repository. This dataset contains 26 columns that contain information about automobiles, including make, model, price, vehicle features (like the number of cylinders), MPG, and an insurance risk score.

Pre-process the data

The main data preparation tasks include data cleaning, integration, transformation, reduction, and discretization or quantization. In the designer, you can find modules to perform these operations and other data pre-processing tasks in the **Data Transformation** group in the left panel.

Use the **Select Columns in Dataset** module to exclude normalized-losses that have many missing values. We then use **Clean Missing Data** to remove the rows that have missing values. This helps to create a clean set of training data.



Train the model

Machine learning problems vary. Common machine learning tasks include classification, clustering, regression, and recommender systems, each of which might require a different algorithm. Your choice of algorithm often depends on the requirements of the use case. After you pick an algorithm, you need to tune its parameters to train a more accurate model. You then need to evaluate all models based on metrics like accuracy, intelligibility, and efficiency.

Because the goal of this pipeline is to predict automobile prices, and because the label column (price) contains real numbers, a regression model is a good choice.

To compare the performance of different algorithms, we use two nonlinear algorithms, **Boosted Decision Tree Regression** and **Decision Forest Regression**, to build models. Both algorithms have parameters that you can change, but this sample uses the default values for this pipeline.

Use the **Split Data** module to randomly divide the input data so that the training dataset contains 70% of the original data and the testing dataset contains 30% of the original data.

Test, evaluate, and compare the models

You use two different sets of randomly chosen data to train and then test the model, as described in the previous section. Split the dataset and use different datasets to train and test the model to make the evaluation of the model more objective.

After the model is trained, use the **Score Model** and **Evaluate Model** modules to generate predicted results and evaluate the models. **Score Model** generates predictions for the test dataset by using the trained model. Then pass the scores to **Evaluate Model** to generate evaluation metrics.

Here are the results:

| Mean_Absolute_Error | Root_Mean_Squared_Error | Relative_Squared_Error | Relative_Absolute_Error | Coefficient_of_Determination |
|---------------------|-------------------------|------------------------|-------------------------|------------------------------|
| 2029.4089 | 3468.292402 | 0.172933 | 0.326473 | 0.827067 |
| 2186.403547 | 4150.979807 | 0.245531 | 0.39172 | 0.754669 |

These results show that the model built with **Boosted Decision Tree Regression** has a lower root mean squared error than the model built on **Decision Forest Regression**.

Chapter 10

Text Classification pipeline in Azure Machine Learning designer

Build a classifier to predict company category using Azure Machine Learning designer.

Reference: <https://learn.microsoft.com/en-us/training/>

This demonstrates how to use text analytics modules to build a text classification pipeline in Azure Machine Learning designer.

The goal of text classification is to assign some piece of text to one or more predefined classes or categories. The piece of text could be a document, news article, search query, email, tweet, support tickets, customer feedback, user product review etc. Applications of text classification include categorizing newspaper articles and news wire contents into topics, organizing web pages into hierarchical categories, filtering spam email, sentiment analysis, predicting user intent from search queries, routing support tickets, and analyzing customer feedback.

This pipeline trains a **multiclass logistic regression classifier** to predict the company category with **Wikipedia SP 500 dataset derived from Wikipedia**.

The fundamental steps of a training machine learning model with text data are:

1. Get the data
2. Pre-process the text data
3. Feature Engineering

Convert text feature into the numerical feature with feature extracting module such as feature hashing, extract n-gram feature from the text data.

1. Train the model
2. Score dataset
3. Evaluate the model

Here's the final, completed graph of the pipeline we'll be working on. We'll provide the rationale for all the modules so you can make similar decisions on your own.



Data

In this pipeline, we use the **Wikipedia SP 500** dataset. The dataset is derived from Wikipedia (<https://www.wikipedia.org/>) based on articles of each S&P 500 company. Before uploading to Azure Machine Learning designer, the dataset was processed as follows:

- Extract text content for each specific company
- Remove wiki formatting
- Remove non-alphanumeric characters
- Convert all text to lowercase
- Known company categories were added

Articles could not be found for some companies, so the number of records is less than 500.

Pre-process the text data

We use the **Preprocess Text** module to preprocess the text data, including detect the sentences, tokenize sentences and so on. You would find all supported options in the [Preprocess Text](#) article. After pre-processing text data, we use the **Split Data** module to randomly divide the input data so that the training dataset contains 50% of the original data and the testing dataset contains 50% of the original data.

Feature Engineering

In this sample, we will use two methods performing feature engineering.

Feature Hashing

We used the [Feature Hashing](#) module to convert the plain text of the articles to integers and used the integer values as input features to the model.

The **Feature Hashing** module can be used to convert variable-length text documents to equal-length numeric feature vectors, using the 32-bit murmurhash v3 hashing method provided by the Vowpal Wabbit library. The objective of using feature hashing is dimensionality reduction; also feature hashing makes the lookup of feature weights faster at classification time because it uses hash value comparison instead of string comparison.

In the sample pipeline, we set the number of hashing bits to 14 and set the number of n-grams to 2. With these settings, the hash table can hold 2^{14} entries, in which each hashing feature represents one or more n-gram features and its value represents the occurrence frequency of that n-gram in the text instance. For many problems, a hash table of this size is more than adequate, but in some cases, more space might be needed to avoid collisions. Evaluate the performance of your machine learning solution using different number of bits.

Extract N-Gram Feature from Text

An n-gram is a contiguous sequence of n terms from a given sequence of text. An n-gram of size 1 is referred to as a unigram; an n-gram of size 2 is a bigram; an n-gram of size 3 is a trigram. N-grams of larger sizes are sometimes referred to by the value of n, for instance, “four-gram”, “five-gram”, and so on.

We used [Extract N-Gram Feature from Text](#) module as another solution for feature engineering. This module first extracts the set of n-grams, in addition to the n-grams, the number of documents where each n-gram appears in the text is counted(DF). In this sample, TF-IDF metric is used to calculate feature values. Then, it converts unstructured text data into equal-length numeric feature vectors where each feature represents the TF-IDF of an n-gram in a text instance.

After converting text data into numeric feature vectors, A **Select Column** module is used to remove the text data from the dataset.

Train the model

Your choice of algorithm often depends on the requirements of the use case. Because the goal of this pipeline is to predict the category of company, a multi-class classifier model is a good choice. Considering that the number of features is large and these features are sparse, we use **Multiclass Logistic Regression** model for this pipeline.

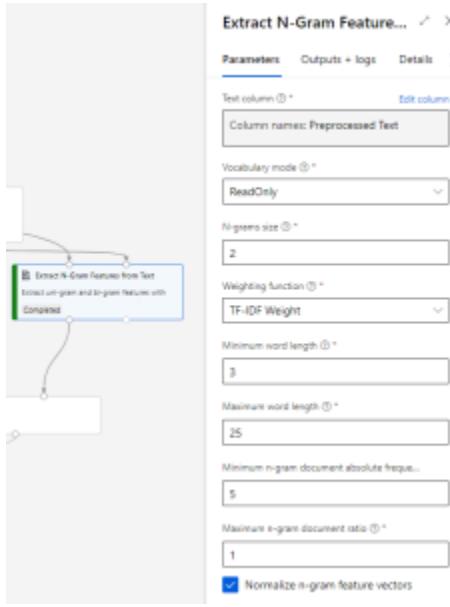
Test, evaluate, and compare

We split the dataset and use different datasets to train and test the model to make the evaluation of the model more objective.

After the model is trained, we would use the **Score Model** and **Evaluate Model** modules to generate predicted results and evaluate the models. However, before using the **Score Model** module, performing feature engineering as what we have done during training is required.

For **Feature Hashing** module, it is easy to perform feature engineer on scoring flow as training flow. Use **Feature Hashing** module directly to process the input text data.

For **Extract N-Gram Feature from Text** module, we would connect the **Result Vocabulary output** from the training dataflow to the **Input Vocabulary** on the scoring dataflow, and set the **Vocabulary mode** parameter to **ReadOnly**.



After finishing the engineering step, **Score Model** could be used to generate predictions for the test dataset by using the trained model. To check the result, select the output port of **Score Model** and then select **Visualize**.

We then pass the scores to the **Evaluate Model** module to generate evaluation metrics. **Evaluate Model** has two input ports, so that we could evaluate and compare scored datasets that are generated with different methods. In this sample, we compare the performance of the result generated with feature hashing method and n-gram method. To check the result, select the output port of the **Evaluate Model** and then select **Visualize**.

Build inference pipeline to deploy a real-time endpoint

After submitting the training pipeline above successfully, you can register the output of the circled module as dataset.



To register dataset, you need to select **Extract N-Gram Feature from Text** module and switch to **Outputs+logs** tab in the right pane. Click on **Register dataset** and fill in the pop-up window.

Extract N-Gram Features from Text

Parameters Outputs + logs Details Metrics >

Data outputs Hide data outputs ^

Result vocabulary   

Results dataset   

Other outputs

- ▽ azureml-logs
 - 55_azureml-execution-tvmps_366a5021c2e008e...
 - 65_job_prep-tvmps_366a5021c2e008efc9a8ef0ff...
 - 70_driver_log.txt
 - 75_job_post-tvmps_366a5021c2e008efc9a8ef0ff...

After register dataset successfully in the training pipeline, you can create real-time inference pipeline. You need to adjust your inference pipeline manually to the following graph:



Then submit the inference pipeline, and deploy a real-time endpoint.

Chapter 11

Azure Machine Learning pipeline for image classification

Azure Machine Learning pipeline for image classification

Reference: <https://learn.microsoft.com/en-us/training/>

The example trains a small [Keras](#) convolutional neural network to classify images in the [Fashion MNIST](#) dataset.

This is based on the image-classification.ipynb notebook found in the python-sdk/tutorial/using-pipelines directory of the [Azure Machine Learning Examples](#) repository. The source code for the steps themselves is in the keras-mnist-fashion subdirectory.

Import types

Import all the Azure Machine Learning types that you'll need for this tutorial:

PythonCopy

```
import os
```

```
import azureml.core
```

```
from azureml.core import (
```

```
    Workspace,
```

```
    Experiment,
```

```
    Dataset,
```

```
    Datastore,
```

```
    ComputeTarget,
```

```
    Environment,
```

```
    ScriptRunConfig
```

```
)
```

```
from azureml.data import OutputFileDatasetConfig
```

```
from azureml.core.compute import AmlCompute
```

```
from azureml.core.compute_target import ComputeTargetException
```

```
from azureml.pipeline.steps import PythonScriptStep
```

```
from azureml.pipeline.core import Pipeline  
  
# check core SDK version number  
  
print("Azure Machine Learning SDK Version: ", azureml.core.VERSION)
```

The Azure Machine Learning SDK version should be 1.37 or greater. If it isn't, upgrade with pip install –upgrade azureml-core.

Configure workspace

Create a workspace object from the existing Azure Machine Learning workspace.

PythonCopy

```
workspace = Workspace.from_config()
```

Important

This code snippet expects the workspace configuration to be saved in the current directory or its parent. For more information on creating a workspace, see [Create workspace resources](#). For more information on saving the configuration to file, see [Create a workspace configuration file](#).

Create the infrastructure for your pipeline

Create an Experiment object to hold the results of your pipeline runs:

PythonCopy

```
exp = Experiment(workspace=workspace, name="keras-mnist-fashion")
```

Create a ComputeTarget that represents the machine resource on which your pipeline will run. The simple neural network used in this tutorial trains in just a few minutes even on a CPU-based machine. If you wish to use a GPU for training, set use_gpu to True. Provisioning a compute target generally takes about five minutes.

PythonCopy

```
use_gpu = False  
  
# choose a name for your cluster  
  
cluster_name = "gpu-cluster" if use_gpu else "cpu-cluster"  
  
found = False  
  
# Check if this compute target already exists in the workspace.  
  
cts = workspace.compute_targets  
  
if cluster_name in cts and cts[cluster_name].type == "AmlCompute":  
  
    found = True
```

```

print("Found existing compute target.")

compute_target = cts[cluster_name]

if not found:

    print("Creating a new compute target...")

    compute_config = AmlCompute.provisioning_configuration(
        vm_size= "STANDARD_NC6" if use_gpu else "STANDARD_D2_V2"
        # vm_priority = 'lowpriority', # optional
        max_nodes=4,
    )

# Create the cluster.

compute_target = ComputeTarget.create(workspace, cluster_name, compute_config)

# Can poll for a minimum number of nodes and for a specific timeout.

# If no min_node_count is provided, it will use the scale settings for the cluster.

compute_target.wait_for_completion(
    show_output=True, min_node_count=None, timeout_in_minutes=10
)

# For a more detailed view of current AmlCompute status, use get_status().print(compute_target.get_status().serialize())

```

Note

GPU availability depends on the quota of your Azure subscription and upon Azure capacity. See [Manage and increase quotas for resources with Azure Machine Learning](#).

Create a dataset for the Azure-stored data

Fashion-MNIST is a dataset of fashion images divided into 10 classes. Each image is a 28×28 grayscale image and there are 60,000 training and 10,000 test images. As an image classification problem, Fashion-MNIST is harder than the classic MNIST handwritten digit database. It's distributed in the same compressed binary form as the original [handwritten digit database](#) .

To create a Dataset that references the Web-based data, run:

PythonCopy

```
data_urls = ["https://data4mldemo6150520719.blob.core.windows.net/demo/mnist-fashion"]
```

```
fashion_ds = Dataset.File.from_files(data_urls)

# list the files referenced by fashion_ds

print(fashion_ds.to_path())
```

This code completes quickly. The underlying data remains in the Azure storage resource specified in the data_urls array.

Create the data-preparation pipeline step

The first step in this pipeline will convert the compressed data files of fashion_ds into a dataset in your own workspace consisting of CSV files ready for use in training. Once registered with the workspace, your collaborators can access this data for their own analysis, training, and so on

PythonCopy

```
datastore = workspace.get_default_datastore()

prepared_fashion_ds = OutputFileDatasetConfig(
    destination=(datastore, "outputdataset/{run-id}"))

.register_on_complete(name="prepared_fashion_ds")
```

The above code specifies a dataset that is based on the output of a pipeline step. The underlying processed files will be put in the workspace's default datastore's blob storage at the path specified in destination. The dataset will be registered in the workspace with the name prepared_fashion_ds.

Create the pipeline step's source

The code that you've executed so far has created and controlled Azure resources. Now it's time to write code that does the first step in the domain.

If you're following along with the example in the [Azure Machine Learning Examples repo](#), the source file is already available as keras-mnist-fashion/prepare.py.

If you're working from scratch, create a subdirectory called keras-mnist-fashion/. Create a new file, add the following code to it, and name the file prepare.py.

PythonCopy

```
# prepare.py

# Converts MNIST-formatted files at the passed-in input path to a passed-in output path

import os

import sys

# Conversion routine for MNIST binary format

def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
```

```
I = open(labelf, "rb")
o = open(outf, "w")
f.read(16)
I.read(8)
images = []
for i in range(n):
    image = [ord(I.read(1))]
    for j in range(28 * 28):
        image.append(ord(f.read(1)))
    images.append(image)
for image in images:
    o.write(",".join(str(pix) for pix in image) + "\n")
f.close()
o.close()
I.close()
```

```
# The MNIST-formatted source
mounted_input_path = sys.argv[1]
# The output directory at which the outputs will be written
mounted_output_path = sys.argv[2]
# Create the output directory
os.makedirs(mounted_output_path, exist_ok=True)
# Convert the training data
convert(
    os.path.join(mounted_input_path, "mnist-fashion/train-images-idx3-ubyte"),
```

```
os.path.join(mounted_input_path, "mnist-fashion/train-labels-idx1-ubyte"),
os.path.join(mounted_output_path, "mnist_train.csv"),
60000,
)
# Convert the test data
convert(
os.path.join(mounted_input_path, "mnist-fashion/t10k-images-idx3-ubyte"),
os.path.join(mounted_input_path, "mnist-fashion/t10k-labels-idx1-ubyte"),
os.path.join(mounted_output_path, "mnist_test.csv"),
10000,
)
```

The code in `prepare.py` takes two command-line arguments: the first is assigned to `mounted_input_path` and the second to `mounted_output_path`. If that subdirectory doesn't exist, the call to `os.makedirs` creates it. Then, the program converts the training and testing data and outputs the comma-separated files to the `mounted_output_path`.

Specify the pipeline step

Back in the Python environment you're using to specify the pipeline, run this code to create a `PythonScriptStep` for your preparation code:

```
PythonCopy
script_folder = "./keras-mnist-fashion"
prep_step = PythonScriptStep(
    name="prepare step",
    script_name="prepare.py",
    # On the compute target, mount fashion_ds dataset as input, prepared_fashion_ds as output
    arguments=[fashion_ds.as_named_input("fashion_ds").as_mount(), prepared_fashion_ds],
    source_directory=script_folder,
    compute_target=compute_target,
    allow_reuse=True,
```

)

The call to PythonScriptStep specifies that, when the pipeline step is run:

- All the files in the script_folderdirectory are uploaded to the compute_target
- Among those uploaded source files, the file prepare.pywill be run
- The fashion_dsand prepared_fashion_ds datasets will be mounted on the compute_target and appear as directories
- The path to the fashion_dsfies will be the first argument to prepare.py. In prepare.py, this argument is assigned to mounted_input_path
- The path to the prepared_fashion_dswill be the second argument to prepare.py. In prepare.py, this argument is assigned to mounted_output_path
- Because allow_reuseis True, it won't be rerun until its source files or inputs change
- This PythonScriptStepwill be named prepare step

Modularity and reuse are key benefits of pipelines. Azure Machine Learning can automatically determine source code or Dataset changes. The output of a step that isn't affected will be reused without rerunning the steps again if allow_reuse is True. If a step relies on a data source external to Azure Machine Learning that may change (for instance, a URL that contains sales data), set allow_reuse to False and the pipeline step will run every time the pipeline is run.

Create the training step

Once the data has been converted from the compressed format to CSV files, it can be used for training a convolutional neural network.

Create the training step's source

With larger pipelines, it's a good practice to put each step's source code in a separate directory (src/prepare/, src/train/, and so on) but for this tutorial, just use or create the file train.py in the same keras-mnist-fashion/ source directory.

PythonCopy

```
import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

from keras.utils import to_categorical

from keras.callbacks import Callback

import numpy as np

import pandas as pd
```

```
import os

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from azureml.core import Run

# dataset object from the run

run = Run.get_context()

dataset = run.input_datasets["prepared_fashion_ds"]

# split dataset into train and test set

(train_dataset, test_dataset) = dataset.random_split(percentage=0.8, seed=111)

# load dataset into pandas dataframe

data_train = train_dataset.to_pandas_dataframe()

data_test = test_dataset.to_pandas_dataframe()

img_rows, img_cols = 28, 28

input_shape = (img_rows, img_cols, 1)

X = np.array(data_train.iloc[:, 1:])

y = to_categorical(np.array(data_train.iloc[:, 0]))

# here we split validation data to optimiza classifier during training

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=13)

# test data

X_test = np.array(data_test.iloc[:, 1:])

y_test = to_categorical(np.array(data_test.iloc[:, 0]))

X_train = (

    X_train.reshape(X_train.shape[0], img_rows, img_cols, 1).astype("float32") / 255

)

X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1).astype("float32") / 255
```

```
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1).astype("float32") / 255
batch_size = 256
num_classes = 10
epochs = 10
# construct neuron network
model = Sequential()
model.add(
    Conv2D(
        32,
        kernel_size=(3, 3),
        activation="relu",
        kernel_initializer="he_normal",
        input_shape=input_shape,
    )
)
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.3))
```

```
model.add(Dense(num_classes, activation="softmax"))

model.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adam(),
    metrics=["accuracy"],
)

# start an Azure ML run

run = Run.get_context()

class LogRunMetrics(Callback):
    # callback at the end of every epoch

    def on_epoch_end(self, epoch, log):
        # log a value repeated which creates a list
        run.log("Loss", log["loss"])

        run.log("Accuracy", log["accuracy"])

history = model.fit(
    X_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_val, y_val),
    callbacks=[LogRunMetrics()],
)

score = model.evaluate(X_test, y_test, verbose=0)

# log a single value
```

```
run.log("Final test loss", score[0])
print("Test loss:", score[0])
run.log("Final test accuracy", score[1])
print("Test accuracy:", score[1])
plt.figure(figsize=(6, 3))
plt.title("Fashion MNIST with Keras ({} epochs)".format(epochs), fontsize=14)
plt.plot(history.history["accuracy"], "b-", label="Accuracy", lw=4, alpha=0.5)
plt.plot(history.history["loss"], "r-", label="Loss", lw=4, alpha=0.5)
plt.legend(fontsize=12)
plt.grid(True)

# log an image
run.log_image("Loss v.s. Accuracy", plot=plt)

# create a ./outputs/model folder in the compute target
# files saved in the "./outputs" folder are automatically uploaded into run history
os.makedirs("./outputs/model", exist_ok=True)

# serialize NN architecture to JSON
model_json = model.to_json()

# save model JSON
with open("./outputs/model/model.json", "w") as f:
    f.write(model_json)

# save model weights
model.save_weights("./outputs/model/model.h5")
print("model saved in ./outputs/model folder")
```

Most of this code should be familiar to ML developers:

- The data is partitioned into train and validation sets for training, and a separate test subset for final scoring

- The input shape is 28x28x1 (only 1 because the input is grayscale), there will be 256 inputs in a batch, and there are 10 classes
- The number of training epochs will be 10
- The model has three convolutional layers, with max pooling and dropout, followed by a dense layer and softmax head
- The model is fitted for 10 epochs and then evaluated
- The model architecture is written to outputs/model/model.json and the weights to outputs/model/model.h5

Some of the code, though, is specific to Azure Machine Learning. `run = Run.get_context()` retrieves a [Run](#) object, which contains the current service context. The `train.py` source uses this run object to retrieve the input dataset via its name (an alternative to the code in `prepare.py` that retrieved the dataset via the `argv` array of script arguments).

The `run` object is also used to log the training progress at the end of every epoch and, at the end of training, to log the graph of loss and accuracy over time.

Create the training pipeline step

The training step has a slightly more complex configuration than the preparation step. The preparation step used only standard Python libraries. More commonly, you'll need to modify the runtime environment in which your source code runs.

Create a file `conda_dependencies.yml` with the following contents:

ymlCopy

dependencies:

– `python=3.7`

– `pip:`

– `azureml-core`

– `azureml-dataset-runtime`

– `keras==2.4.3`

– `tensorflow==2.4.3`

– `numpy`

– `scikit-learn`

– `pandas`

– `matplotlib`

The `Environment` class represents the runtime environment in which a machine learning task runs. Associate the above specification with the training code with:

PythonCopy

```
keras_env = Environment.from_conda_specification(  
    name="keras-env", file_path=".//conda_dependencies.yml"  
)  
  
train_cfg = ScriptRunConfig(  
    source_directory=script_folder,  
    script="train.py",  
    compute_target=compute_target,  
    environment=keras_env,  
)
```

Creating the training step itself uses code similar to the code used to create the preparation step:

PythonCopy

```
train_step = PythonScriptStep(  
    name="train step",  
    arguments=[  
        prepared_fashion_ds.read_delimited_files().as_input(name="prepared_fashion_ds")  
    ],  
    source_directory=train_cfg.source_directory,  
    script_name=train_cfg.script,  
    runconfig=train_cfg.run_config,  
)
```

Create and run the pipeline

Now that you've specified data inputs and outputs and created your pipeline's steps, you can compose them into a pipeline and run it:

PythonCopy

```
pipeline = Pipeline(workspace, steps=[prep_step, train_step])  
run = exp.submit(pipeline)
```

The Pipeline object you create runs in your workspace and is composed of the preparation and training steps you've specified.

Note

This pipeline has a simple dependency graph: the training step relies on the preparation step and the preparation step relies on the fashion_ds dataset. Production pipelines will often have much more complex dependencies. Steps may rely on multiple upstream steps, a source code change in an early step may have far-reaching consequences, and so on. Azure Machine Learning tracks these concerns for you. You need only pass in the array of steps and Azure Machine Learning takes care of calculating the execution graph.

The call to submit the Experiment completes quickly, and produces output similar to:

.NET CLICopy

```
Submitted PipelineRun 5968530a-abcd-1234-9cc1-46168951b5eb
```

Link to Azure Machine Learning Portal: <https://ml.azure.com/runs/abc-xyz...>

You can monitor the pipeline run by opening the link or you can block until it completes by running:

PythonCopy

```
run.wait_for_completion(show_output=True)
```

Important

The first pipeline run takes roughly *15 minutes*. All dependencies must be downloaded, a Docker image is created, and the Python environment is provisioned and created. Running the pipeline again takes significantly less time because those resources are reused instead of created. However, total run time for the pipeline depends on the workload of your scripts and the processes that are running in each pipeline step.

Once the pipeline completes, you can retrieve the metrics you logged in the training step:

PythonCopy

```
run.find_step_run("train step")[0].get_metrics()
```

If you're satisfied with the metrics, you can register the model in your workspace:

PythonCopy

```
run.find_step_run("train step")[0].register_model(  
    model_name="keras-model",  
    model_path="outputs/model/",  
    datasets=[("train test data", fashion_ds)],  
)
```

Chapter 12

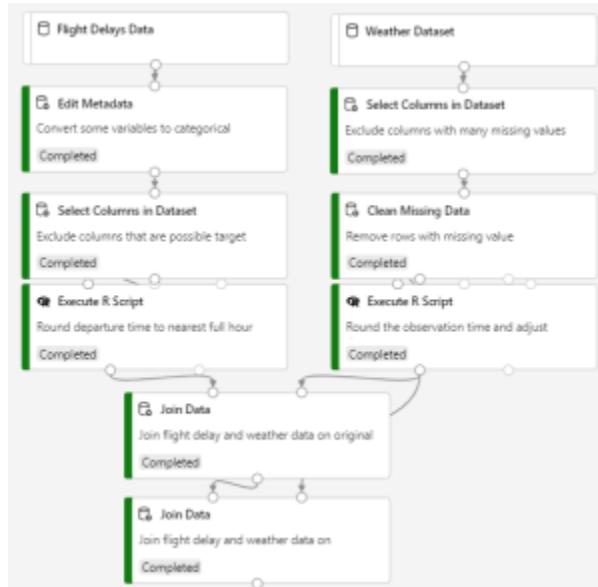
Multi Dataset classifier problem using Designer Studio

Build a classifier & use R to predict flight delays with Azure Machine Learning designer

Reference: <https://learn.microsoft.com/en-us/training/>

This pipeline uses historical flight and weather data to predict if a scheduled passenger flight will be delayed by more than 15 minutes. This problem can be approached as a classification problem, predicting two classes: delayed, or on time.

Here's the final pipeline graph for this sample:



Data

This sample uses the **Flight Delays Data** dataset. It's part of the TranStats data collection from the U.S. Department of Transportation. The dataset contains flight delay information from April to October 2013. The dataset has been pre-processed as follows:

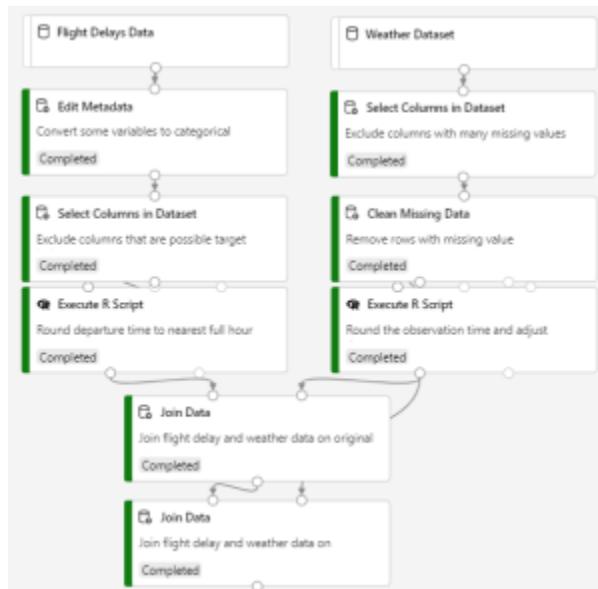
- Filtered to include the 70 busiest airports in the continental United States.
- Relabeled canceled flights as delayed by more than 15 mins.
- Filtered out diverted flights.
- Selected 14 columns.

To supplement the flight data, the **Weather Dataset** is used. The weather data contains hourly, land-based weather observations from NOAA, and represents observations from airport weather stations, covering the same time period as the flights dataset. It has been pre-processed as follows:

- Weather station IDs were mapped to corresponding airport IDs.
- Weather stations not associated with the 70 busiest airports were removed.
- The Date column was split into separate columns: Year, Month, and Day.
- Selected 26 columns.

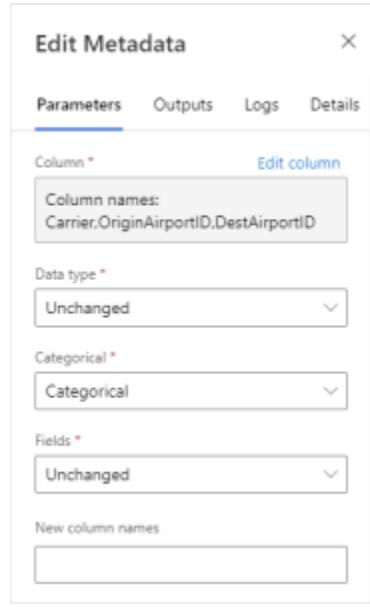
Pre-process the data

A dataset usually requires some pre-processing before it can be analyzed.



Flight data

The columns **Carrier**, **OriginAirportID**, and **DestAirportID** are saved as integers. However, they're categorical attributes, use the **Edit Metadata** module to convert them to categorical.



Then use the **Select Columns** in Dataset module to exclude from the dataset columns that are possible target leakers: **DepDelay**, **DepDel15**, **ArrDelay**, **Canceled**, **Year**.

To join the flight records with the hourly weather records, use the scheduled departure time as one of the join keys. To do the join, the **CSRDepTime** column must be rounded down to the nearest hour, which is done by in the **Execute R Script** module.

Weather data

Columns that have a large proportion of missing values are excluded using the **Project Columns** module. These columns include all string-valued columns: **ValueForWindCharacter**, **WetBulbFarenheit**, **WetBulbCelsius**, **PressureTendency**, **PressureChange**, **SeaLevelPressure**, and **StationPressure**.

The **Clean Missing Data** module is then applied to the remaining columns to remove rows with missing data.

Weather observation times are rounded up to the nearest full hour. Scheduled flight times and the weather observation times are rounded in opposite directions to ensure the model uses only weather before the flight time.

Since weather data is reported in local time, time zone differences are accounted for by subtracting the time zone columns from the scheduled departure time and the weather observation time. These operations are done using the **Execute R Script** module.

Joining Datasets

Flight records are joined with weather data at origin of the flight (**OriginAirportID**) using the **Join Data** module.

Join Data X

Parameters Outputs Logs Details

Join key columns for left dataset * [Edit column](#)

Column names:
Month,DayofMonth,OriginAirportID,CRSDep

Join key columns for right dataset * [Edit column](#)

Column names:
AdjustedMonth,AdjustedDay,AirportID,Adju

Match case

Join type *

Inner Join

Keep right key columns in joined table

Flight records are joined with weather data using the destination of the flight (**DestAirportID**).

Join Data X

Parameters Outputs Logs Details

Join key columns for left dataset * [Edit column](#)

Column names:
Month,DayofMonth,DestAirportID,CRSDepT

Join key columns for right dataset * [Edit column](#)

Column names:
AdjustedMonth,AdjustedDay,AirportID,Adju

Match case

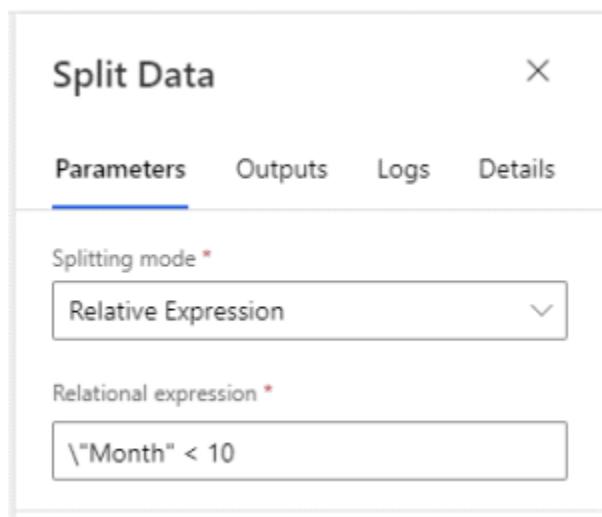
Join type *

Inner Join

Keep right key columns in joined table

Preparing Training and Test Samples

The **Split Data** module splits the data into April through September records for training, and October records for test.



Year, month, and timezone columns are removed from the training dataset using the Select Columns module.

Define features

In machine learning, features are individual measurable properties of something you're interested in. Finding a strong set of features requires experimentation and domain knowledge. Some features are better for predicting the target than others. Also, some features may have a strong correlation with other features, and won't add new information to the model. These features can be removed.

To build a model, you can use all the features available, or select a subset of the features.

Choose and apply a learning algorithm

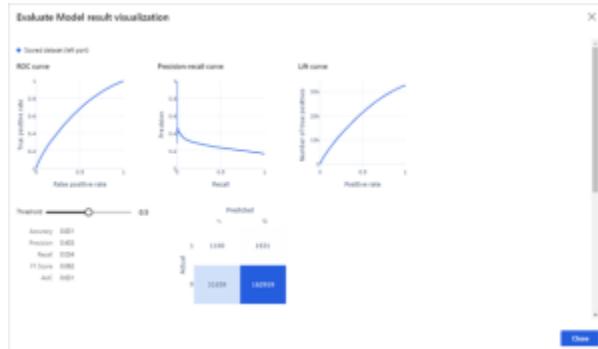
Create a model using the **Two-Class Logistic Regression** module and train it on the training dataset.

The result of the **Train Model** module is a trained classification model that can be used to score new samples to make predictions. Use the test set to generate scores from the trained models. Then use the **Evaluate Model** module to analyze and compare the quality of the models. pipeline After you run the pipeline, you can view the output from the **Score Model** module by clicking the output port and selecting **Visualize**. The output includes the scored labels and the probabilities for the labels.

Finally, to test the quality of the results, add the **Evaluate Model** module to the pipeline canvas, and connect the left input port to the output of the Score Model module. Run the pipeline and view the output of the **Evaluate Model** module, by clicking the output port and selecting **Visualize**.

Evaluate

The logistic regression model has AUC of 0.631 on the test set.



Chapter 13

Python scripts to predict credit risk using designer studio

Build a classifier & use Python scripts to predict credit risk using Azure Machine Learning designer

Reference: <https://learn.microsoft.com/en-us/training/>

This model trains a classifier to predict credit risk using credit application information such as credit history, age, and number of credit cards. However, you can apply the concepts in this article to tackle your own machine learning problems.

Here's the completed graph for this pipeline:

Data



This sample uses the German Credit Card dataset from the UC Irvine repository. It contains 1,000 samples with 20 features and one label. Each sample represents a person. The 20 features include numerical and categorical features. For more information about the dataset, see the [UCI website](#). The last column is the label, which denotes the credit risk and has only two possible values: high credit risk = 2, and low credit risk = 1.

Pipeline summary

In this pipeline, you compare two different approaches for generating models to solve this problem:

- Training with the original dataset.
- Training with a replicated dataset.

With both approaches, you evaluate the models by using the test dataset with replication to ensure that results are aligned with the cost function. Test two classifiers with both approaches: **Two-Class Support Vector Machine** and **Two-Class Boosted Decision Tree**.

The cost of misclassifying a low-risk example as high is 1, and the cost of misclassifying a high-risk example as low is 5. We use an **Execute Python Script** module to account for this misclassification cost.

Data processing

Start by using the **Metadata Editor** module to add column names to replace the default column names with more meaningful names, obtained from the dataset description on the UCI site. Provide the new column names as comma-separated values in the **New column** name field of the **Metadata Editor**.

Next, generate the training and test sets used to develop the risk prediction model. Split the original dataset into training and test sets of the same size by using the **Split Data** module. To create sets of equal size, set the **Fraction of rows in the first output dataset** option to 0.7.

Generate the new dataset

Because the cost of underestimating risk is high, set the cost of misclassification like this:

- For high-risk cases misclassified as low risk: 5
- For low-risk cases misclassified as high risk: 1

To reflect this cost function, generate a new dataset. In the new dataset, each high-risk example is replicated five times, but the number of low-risk examples doesn't change. Split the data into training and test datasets before replication to prevent the same row from being in both sets.

To replicate the high-risk data, put this Python code into an **Execute Python Script** module:

```
import pandas as pd

def azureml_main(dataframe1 = None, dataframe2 = None):
    df_label_1 = dataframe1[dataframe1.iloc[:, 20] == 1]
    df_label_2 = dataframe1[dataframe1.iloc[:, 20] == 2]
    result = df_label_1.append([df_label_2] * 5, ignore_index=True)
    return result,
```

The **Execute Python Script** module replicates both the training and test datasets.

Feature engineering

The **Two-Class Support Vector Machine** algorithm requires normalized data. So use the **Normalize Data** module to normalize the ranges of all numeric features with a tanh transformation. A tanh transformation converts all numeric features to values within a range of 0 and 1 while preserving the overall distribution of values.

The **Two-Class Support Vector Machine** module handles string features, converting them to categorical features and then to binary features with a value of zero or one. So you don't need to normalize these features.

Models

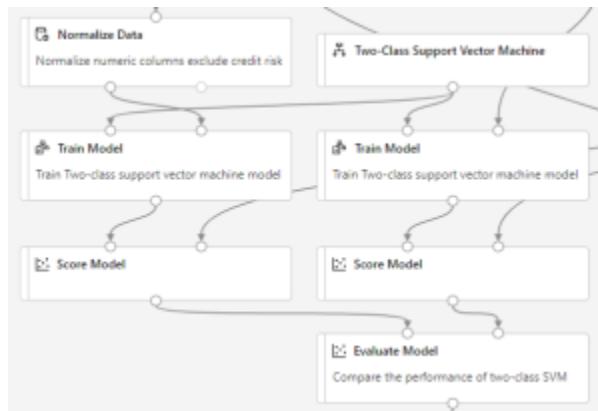
Because you applied two classifiers, **Two-Class Support Vector Machine** (SVM) and **Two-Class Boosted Decision Tree**, and two datasets, you generate a total of four models:

- SVM trained with original data.
- SVM trained with replicated data.
- Boosted Decision Tree trained with original data.
- Boosted Decision Tree trained with replicated data.

This sample uses the standard data science workflow to create, train, and test the models:

1. Initialize the learning algorithms, using **Two-Class Support Vector Machine** and **Two-Class Boosted Decision Tree**.
2. Use **Train Model** to apply the algorithm to the data and create the actual model.
3. Use **Score Model** to produce scores by using the test examples.

The following diagram shows a portion of this pipeline, in which the original and replicated training sets are used to train two different SVM models. **Train Model** is connected to the training set, and **Score Model** is connected to the test set.



In the evaluation stage of the pipeline, you compute the accuracy of each of the four models. For this pipeline, use **Evaluate Model** to compare examples that have the same misclassification cost.

The **Evaluate Model** module can compute the performance metrics for as many as two scored models. So you can use one instance of **Evaluate Model** to evaluate the two SVM models and another instance of **Evaluate Model** to evaluate the two Boosted Decision Tree models.

Notice that the replicated test dataset is used as the input for **Score Model**. In other words, the final accuracy scores include the cost for getting the labels wrong.

Combine multiple results

The **Evaluate Model** module produces a table with a single row that contains various metrics. To create a single set of accuracy results, we first use **Add Rows** to combine the results into a single table. We then use the following Python script in the **Execute Python Script** module to add the model name and training approach for each row in the table of results:

```
import pandas as pd

def azureml_main(dataframe1 = None, dataframe2 = None):
    new_cols = pd.DataFrame(
        columns=[“Algorithm”, “Training”],
        data=[

            [“SVM”, “weighted”],
            [“SVM”, “unweighted”],
            [“Boosted Decision Tree”, “weighted”],
            [“Boosted Decision Tree”, “unweighted”]
        ])
    result = pd.concat([new_cols, dataframe1], axis=1)
    return result,
```

Results

To view the results of the pipeline, you can right-click the Visualize output of the last **Select Columns in Dataset** module.

| Select Columns in Dataset result visualization | | |
|--|------------|----------|
| Rows | Columns | |
| 4 | 3 | |
| Algorithm | Training | Accuracy |
| SVM | weighted | 0.721212 |
| SVM | unweighted | 0.571212 |
| Boosted Decision Tree | weighted | 0.706061 |
| Boosted Decision Tree | unweighted | 0.590909 |

The first column lists the machine learning algorithm used to generate the model.

The second column indicates the type of the training set.

The third column contains the cost-sensitive accuracy value.

From these results, you can see that the best accuracy is provided by the model that was created with **Two-Class Support Vector Machine** and trained on the replicated training dataset.