

Name: Sarthak Kulkarni

Class: LY AIA 3

Roll no: 2213680

Enrollment No.: MITU21BTCS0544

Sub: BDAL

BDA Lab Assignment 1: Data Preparation and Exploration

Theory

Objective:

To download, unpack, explore, and analyze a dataset, providing basic visualizations and insights.

Steps Involved:

1. Download a sample dataset suitable for data analysis (e.g., from Kaggle).
2. Set up the working environment for data analysis in Python.
3. Load and explore the dataset, including encoding and visualizing data.

Algorithm/Flowchart

Algorithm:

1. Download Dataset: Select a dataset from Kaggle and download it locally.
2. Setup Environment: Configure the working directory using Python.
3. Unpack Data: Extract compressed files into a local directory.
4. View Data:
 - Display the first 10 rows.
 - Display the last 10 rows.
5. Count Rows: Measure the number of rows in the dataset.
6. Encode Categorical Data: Convert categorical values into numerical representations.
7. Visualize Data: Plot graphs for selected variables to derive insights.

Code

```
# Import necessary libraries
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Step 1: Set up working directory
os.chdir('path_to_your_working_directory')

# Step 2: Load Dataset
data = pd.read_csv('MallData.csv') # Replace with your dataset file name

# Step 3: Display top 10 and bottom 10 rows
print("Top 10 Rows:\n", data.head(10))
print("\nBottom 10 Rows:\n", data.tail(10))
```

```

# Step 4: Count the number of rows
num_rows = len(data)
print(f"\nNumber of rows in the dataset: {num_rows}")

# Step 5: Encode Categorical Data
categorical_columns = data.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in categorical_columns:
    data[col] = le.fit_transform(data[col])

print("\nEncoded Dataset:\n", data.head())

# Step 6: Visualize Data
# Example: Plot a histogram of a numeric column
column_to_plot = 'example_numeric_column' # Replace with an actual numeric column
name
plt.hist(data[column_to_plot], bins=20, color='blue', edgecolor='black')
plt.title(f"Distribution of {column_to_plot}")
plt.xlabel(column_to_plot)
plt.ylabel("Frequency")
plt.show()

```

Output Screenshot

```

Top 10 Rows:
# A tibble: 10 × 5
  CustomerID Genre    Age `Annual Income (k$)` `Spending Score (1-100)`
  <dbl>   <chr>   <dbl>         <dbl>             <dbl>
1         1 Male     19             15              39
2         2 NA Male     21             15              81
3         3 Female   20             NA               6
4         4 NA       NA             16             NA
5         5 Female   31             17             40
6         6 Female   22             17             76
7         7 Female   35             18               6
8         8 NA       23             18             NA
9         9 Male     64             19               3
10        10 Female   30             19             72

```

Bottom 10 Rows:

A tibble: 10 × 5

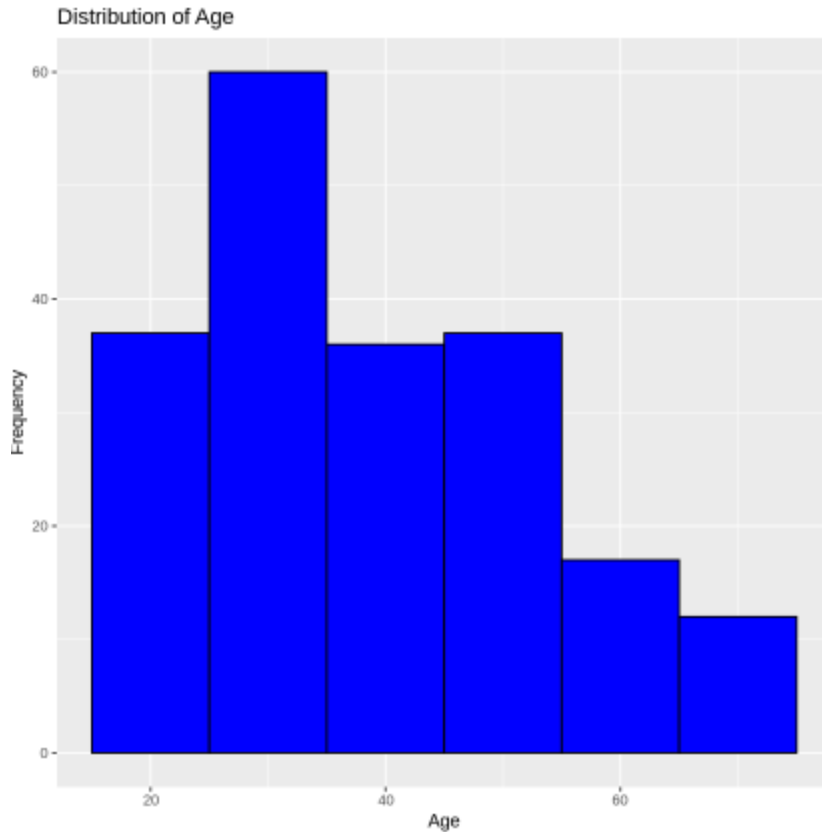
	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
	<dbl>	<chr>	<dbl>	<dbl>	<dbl>
1	191	Female	34	103	23
2	192	Female	32	103	69
3	193	Male	33	113	8
4	194	Female	38	113	91
5	195	Female	47	120	16
6	196	Female	35	120	79
7	197	Female	45	126	28
8	198	Male	32	126	74
9	199	Male	32	137	18
10	200	Male	30	137	83

Number of rows in the dataset: 200

Encoded Dataset (First 10 Rows):

A tibble: 10 × 5

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1	2	19	15	39
2	NA	2	21	15	81
3	3	1	20	NA	6
4	4	NA	NA	16	NA
5	5	1	31	17	40
6	6	1	22	17	76
7	7	1	35	18	6
8	8	NA	23	18	NA
9	9	2	64	19	3
10	10	1	30	19	72



Conclusion

In this lab assignment, we successfully:

1. Downloaded and unpacked a dataset from Kaggle.
2. Explored the data by displaying its top and bottom rows.
3. Counted the total number of rows in the dataset.
4. Encoded categorical variables into numerical values.
5. Plotted a graph to derive insights.

These steps are fundamental to preparing and exploring datasets, forming the base for further analysis.

Assignment 2: Data and Text Analysis, and Visualization Using R and Python

Theory

1. Data Analysis Techniques

Data analysis is a critical process in big data that involves: - Inspecting, cleansing, and transforming raw data - Discovering useful information and supporting decision-making - Utilizing statistical and computational techniques to extract insights

2. Text Analysis Methods

Text analysis involves: - Processing unstructured textual data - Extracting meaningful patterns and information - Applying natural language processing techniques

3. Data Visualization Principles

Data visualization helps in: - Representing complex data in graphical or pictorial format - Making data more accessible and understandable - Identifying trends, patterns, and outliers

Algorithm

The process involves five main stages:

1. **Data Collection**, where data is gathered.
2. **Data Preprocessing**, which cleans, transforms, and normalizes the data.
3. **Exploratory Data Analysis**, where summary statistics and visualizations are created to uncover insights.
4. **Text Analysis**, applied to textual data to extract meaningful features and analyze them.
5. **Visualization and Reporting**, where the findings are displayed through visual means and documented for presentation.

Code

Text Analysis

```
# Install necessary libraries
```

```
install.packages("tm") # For text mining
```

```
install.packages("SnowballC") # For stemming
```

```
install.packages("wordcloud") # For word cloud visualization
```

```
install.packages("ggplot2") # For visualization
```

```
# Load the libraries
```

```
library(tm)
```

```
library(SnowballC)
```

```
library(wordcloud)
```

```
library(ggplot2)
```

```
# Step 1: Sample Textual Data
```

```
documents <- c(
```

```
  "Data analysis is fun.",
```

```
  "Text analysis is a subset of data analysis.",
```

```
  "Visualization makes analysis easier to understand.",
```

```
  "Visualization makes analysis fun to understand.",
```

```
  "Data analysis is fun.",
```

```
  "Text analysis is a subset of text analysis.",
```

```
  "Visualization makes analysis easier to understand.",
```

```
  "Visualization makes analysis fun to understand.",
```

```
  "Data analysis is fun.",
```

```
  "Text analysis is a subset of data analysis.",
```

```
  "Visualization makes analysis easier to understand.",
```

```
  "Visualization makes analysis fun to understand."
```

```
)
```

```
# Step 2: Create a Text Corpus
```

```
corpus <- Corpus(VectorSource(documents))
```

```
# Step 3: Clean the Text
```



```
corpus <- tm_map(corpus, content_transformer(tolower)) # Convert to lowercase
corpus <- tm_map(corpus, removePunctuation) # Remove punctuation
corpus <- tm_map(corpus, removeNumbers) # Remove numbers
corpus <- tm_map(corpus, removeWords, stopwords("english")) # Remove stopwords
corpus <- tm_map(corpus, stripWhitespace) # Remove extra whitespace
```

```
# Step 4: Create a Term Document Matrix
```

```
tdm <- TermDocumentMatrix(corpus)
```

```
tdm_matrix <- as.matrix(tdm)
```

```
# Get word frequencies
```

```
word_freq <- sort(rowSums(tdm_matrix), decreasing = TRUE)
```

```
# Print the Term Frequency Matrix
```

```
print("Term Frequency Matrix:")
```

```
print(tdm_matrix)
```

```
# Step 5: Visualize Word Frequencies with Word Cloud
```

```
wordcloud(names(word_freq), freq = word_freq, min.freq = 1)
```

```
# Step 6: Top 5 Frequent Words
```

```
top_5_words <- head(word_freq, 5)
```

```
# Create a Data Frame for the top 5 words
```

```
top_5_words_df <- data.frame(
```

```
  Word = names(top_5_words),
```

```
  Frequency = top_5_words
```

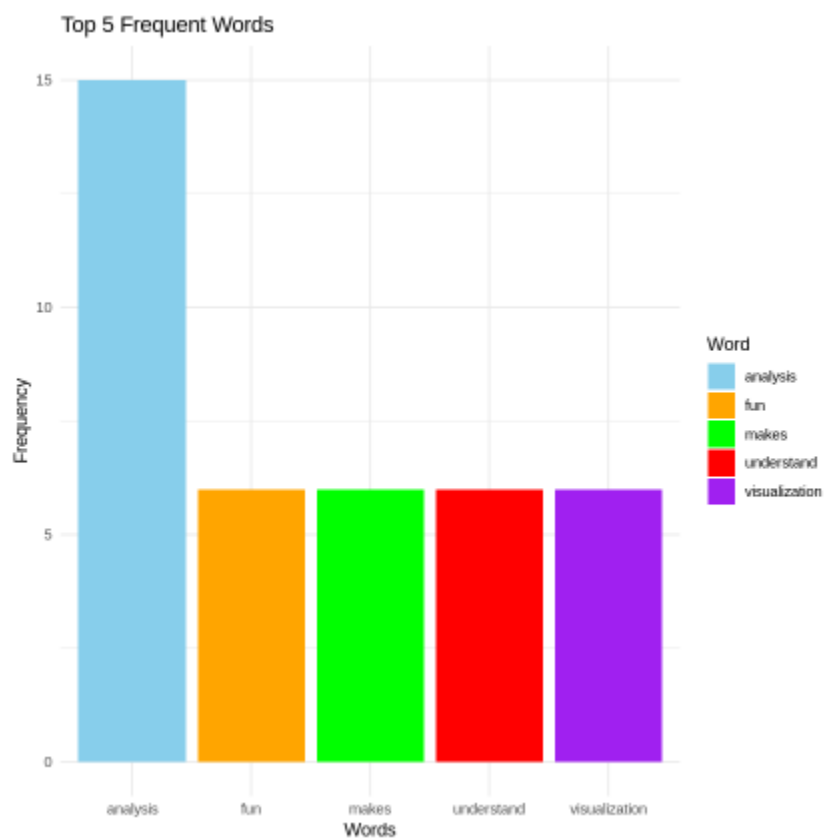
```
)
```

Step 7: Plot Top 5 Frequent Words in a Histogram

```
ggplot(top_5_words_df, aes(x = Word, y = Frequency, fill = Word)) +  
  geom_bar(stat = "identity") +  
  labs(title = "Top 5 Frequent Words", x = "Words", y = "Frequency") +  
  theme_minimal() +  
  scale_fill_manual(values = c("skyblue", "orange", "green", "red", "purple"))
```

Output

	Docs											
Terms	1	2	3	4	5	6	7	8	9	10	11	12
analysis	1	2	1	1	1	2	1	1	1	2	1	1
data	1	1	0	0	1	0	0	0	1	1	0	0
fun	1	0	0	1	1	0	0	1	1	0	0	1
subset	0	1	0	0	0	1	0	0	0	1	0	0
text	0	1	0	0	0	2	0	0	0	1	0	0
easier	0	0	1	0	0	0	1	0	0	0	1	0
makes	0	0	1	1	0	0	1	1	0	0	1	1
understand	0	0	1	1	0	0	1	1	0	0	1	1
visualization	0	0	1	1	0	0	1	1	0	0	1	1



Data Analysis and Visualization

Install necessary libraries

```
install.packages("ggplot2")
```

```
install.packages("dplyr")
```

Load the libraries

```
library(ggplot2)
```

```
library(dplyr)
```

Load the Titanic dataset

```
data <- read.csv("titanic.csv") # Replace with your Titanic dataset file path
```

View the structure of the data

```
str(data)
```

Check for missing values

```
colSums(is.na(data))
```

Fill missing values in 'Age' with the median

```
data$Age[is.na(data$Age)] <- median(data$Age, na.rm = TRUE)
```

Fill missing values in 'Embarked' with the most frequent value

```
data$Embarked[is.na(data$Embarked)] <- "S"
```

Convert relevant columns to factors

```
data$Survived <- factor(data$Survived, levels = c(0, 1), labels = c("No", "Yes"))
```

```
data$Pclass <- factor(data$Pclass)
```

```
data$Sex <- factor(data$Sex)
```

Survival rate by class

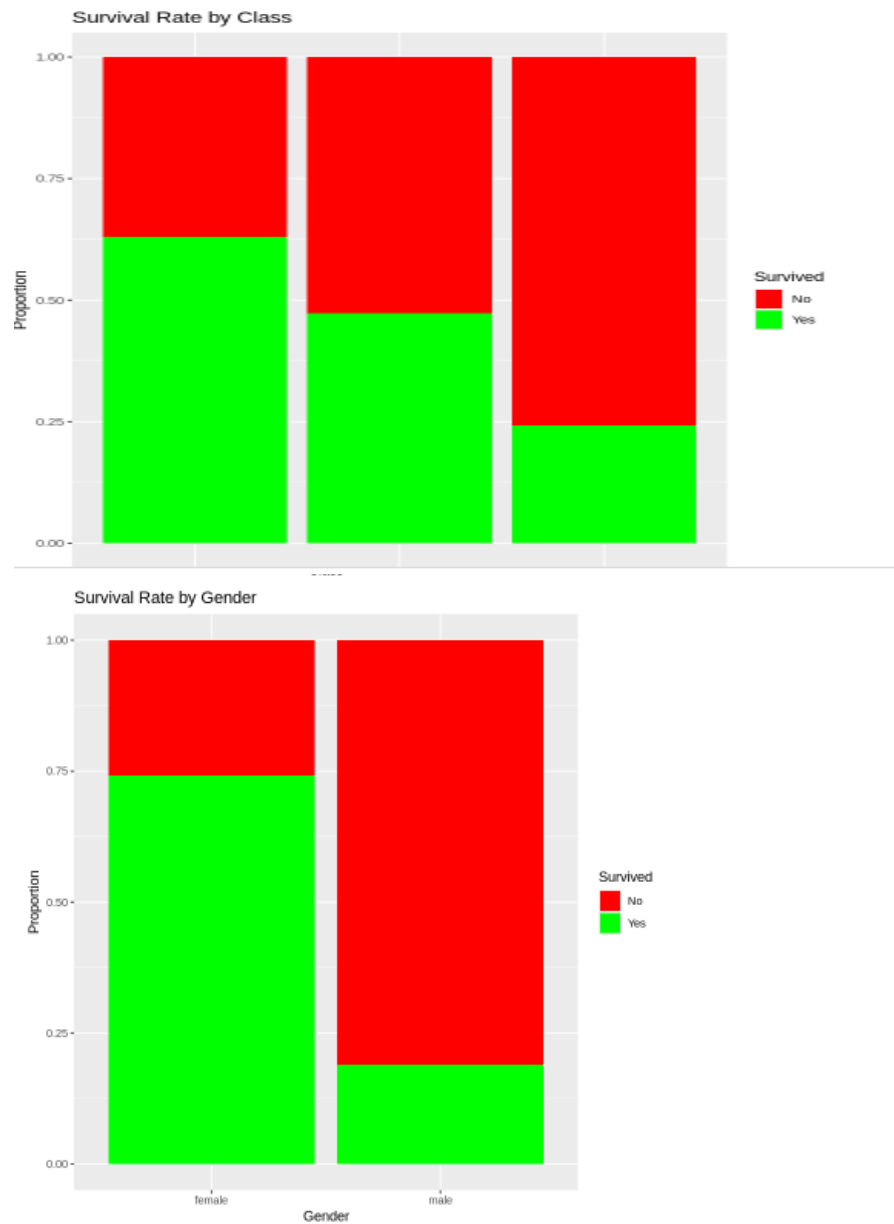
```
ggplot(data, aes(x = Pclass, fill = Survived)) +
```

```
  geom_bar(position = "fill") +
```

```
labs(title = "Survival Rate by Class", y = "Proportion", x = "Class") +
scale_fill_manual(values = c("red", "green"))
```

Output

```
'data.frame': 891 obs. of 12 variables:
 $ PassengerId: int 1 2 3 4 5 6 7 8 9 10 ...
 $ Survived : int 0 1 1 1 0 0 0 0 1 1 ...
 $ Pclass : int 3 1 3 1 3 3 1 3 3 2 ...
 $ Name : chr "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)" "Heikkinen, Miss. Laina" "Futrelle, Mrs. Jacques Heat
 $ Sex : chr "male" "female" "female" "female" ...
 $ Age : num 22 38 26 35 35 NA 54 2 27 14 ...
 $ SibSp : int 1 1 0 1 0 0 0 3 0 1 ...
 $ Parch : int 0 0 0 0 0 0 0 1 2 0 ...
 $ Ticket : chr "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
 $ Fare : num 7.25 71.28 7.92 53.1 8.05 ...
 $ Cabin : chr "" "C85" "" "C123" ...
 $ Embarked : chr "S" "C" "S" "S" ...
 PassengerId: 0 Survived: 0 Pclass: 0 Name: 0 Sex: 0 Age: 177 SibSp: 0 Parch: 0 Ticket: 0 Fare: 0 Cabin: 0 Embarked:
```



Data Visualization using R with different datatypes(csv,xls,json)

Install necessary libraries

```
install.packages("ggplot2")
```

```
install.packages("readr")
```

```
install.packages("readxl")
```

```
install.packages("jsonlite")
```

```
install.packages("writexl")
```

```
# Load libraries
```

```
library(ggplot2)
```

```
library(readr)
```

```
library(readxl)
```

```
library(jsonlite)
```

```
library(writexl)
```

```
# Step 1: Load the built-in Iris dataset
```

```
data(iris)
```

```
# Step 2: Save the Iris dataset to CSV, Excel, and JSON
```

```
write.csv(iris, "iris.csv", row.names = FALSE) # Save as CSV
```

```
write_xlsx(iris, "iris.xlsx") # Save as Excel
```

```
write_json(iris, "iris.json") # Save as JSON
```

```
# Step 3: Load Data from CSV
```

```
iris_csv <- read_csv("iris.csv")
```

```
head(iris_csv)
```

```
# Visualization: Sepal Length by Species (CSV)
```

```
ggplot(iris_csv, aes(x = Species, y = Sepal.Length, fill = Species)) +
```

```
  geom_boxplot() +
```

```
  labs(title = "Sepal Length by Species (CSV)", x = "Species", y = "Sepal Length") +
```

```
  theme_minimal()
```

```
# Step 4: Load Data from Excel
```

```
iris_excel <- read_excel("iris.xlsx")
```

```
head(iris_excel)
```

```
# Visualization: Sepal Length by Species (Excel)
```

```
ggplot(iris_excel, aes(x = Species, y = Sepal.Length, fill = Species)) +
```

```
  geom_boxplot() +
```

```
  labs(title = "Sepal Length by Species (Excel)", x = "Species", y = "Sepal Length") +
```

```
  theme_minimal()
```

```
# Step 5: Load Data from JSON
```

```
iris_json <- fromJSON("iris.json")
```

```
head(iris_json)
```

```
# Visualization: Sepal Length by Species (JSON)
```

```
ggplot(iris_json, aes(x = Species, y = Sepal.Length, fill = Species)) +
```

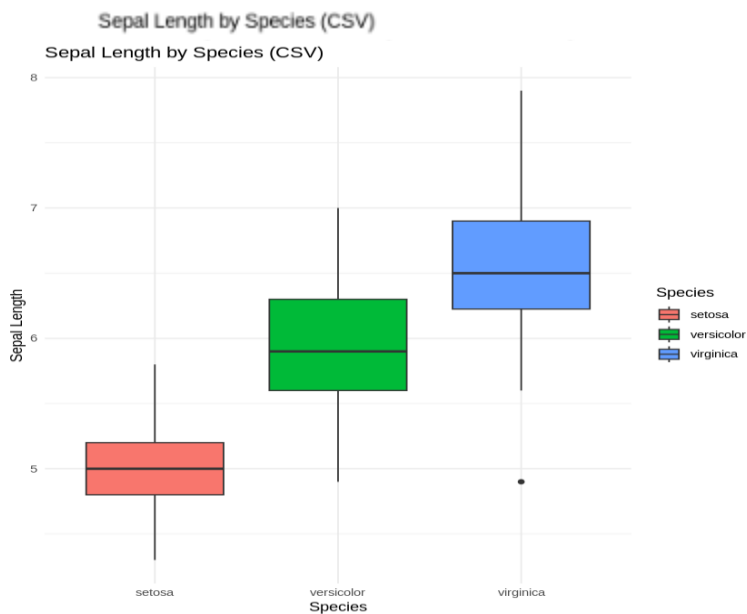
```
  geom_boxplot() +
```

```
  labs(title = "Sepal Length by Species (JSON)", x = "Species", y = "Sepal Length") +
```

```
  theme_minimal()
```


Output Screenshots:

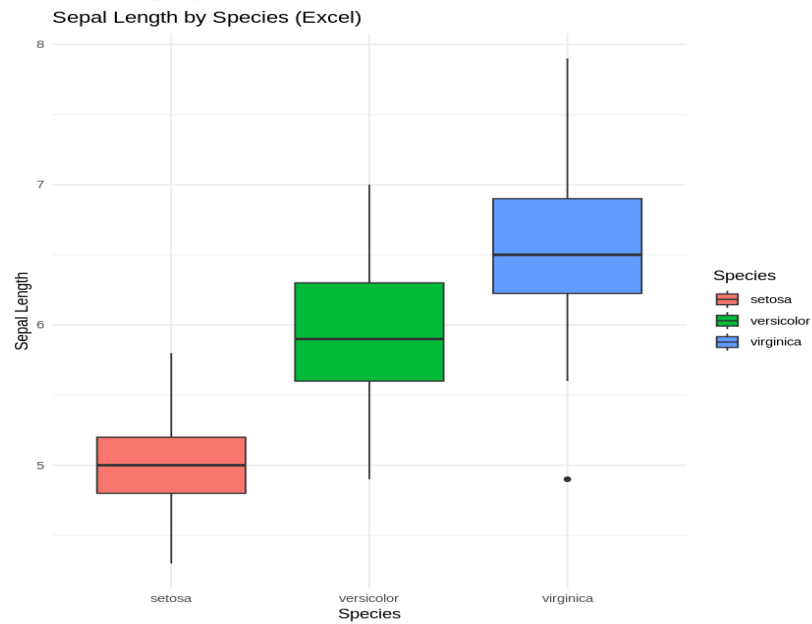
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<dbl>	<dbl>	<dbl>	<dbl>	<chr>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa



A tibble: 6 × 5

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<dbl>	<dbl>	<dbl>	<dbl>	<chr>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

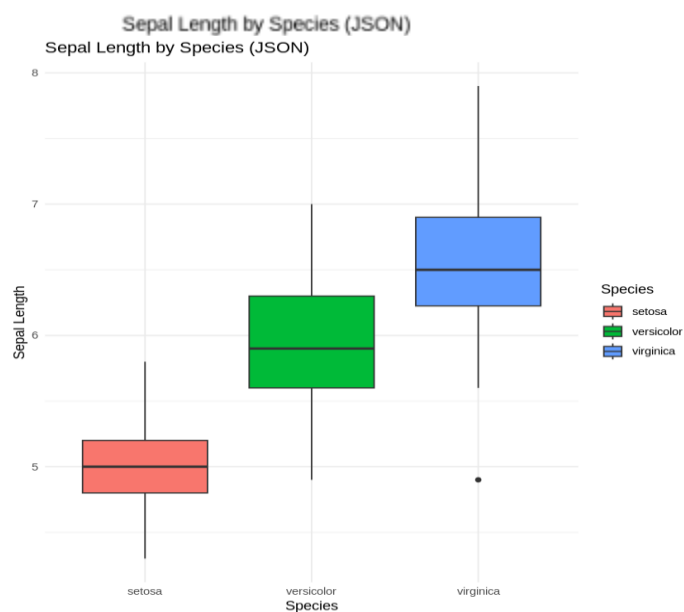
Sepal Length by Species (Excel)





A data.frame: 6 × 5

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa



Conclusion

Key Findings

1. Data preprocessing is crucial for accurate analysis
2. R provides powerful tools for data manipulation and visualization
3. Text analysis techniques help extract meaningful insights from unstructured data

Challenges and Learnings

- Handling diverse data formats
- Implementing effective preprocessing techniques
- Creating meaningful visualizations

Lab Assignment

Experiment 3

Word Count Program in Apache Spark

Title

Word Count Program in Apache Spark with Unit Tests.

Theory

Apache Spark is an open-source distributed computing system for big data processing. This experiment involves installing Apache Spark and writing a program to perform a word count, which is a basic example to demonstrate Spark's distributed processing capabilities. Unit tests will be implemented to ensure the correctness of the program.

Algorithm/Flowchart

1. Install Apache Spark and set up the environment.
2. Write a Spark program to read a text file.
3. Split the text into words and map each word to a key-value pair (word, 1).
4. Use the reduceByKey function to count occurrences of each word.
5. Write unit tests to validate the logic and output of the program.
6. Execute the program and verify the output.

Code

Apache Spark Word Count Code (Python)

```
from pyspark import SparkContext
```

```
# Initialize SparkContext
```

```
sc = SparkContext("local", "Word Count")
```

```
# Read input file
```

```
text_file = sc.textFile("sample_text.txt")
```

```
# Perform word count
```

```
word_counts = text_file.flatMap(lambda line: line.split())          .map(lambda word: (word, 1))  
                      .reduceByKey(lambda a, b: a + b)
```

```
# Collect and print results
```

```
for word, count in word_counts.collect():  
    print(f"{word}: {count}")
```

Unit Test Example

```
def test_word_count():
```

```
    test_data = ["hello world", "hello"]
```

```
    rdd = sc.parallelize(test_data)
```

```
    result = rdd.flatMap(lambda line: line.split())                .map(lambda word: (word, 1))
```

```
.reduceByKey(lambda a, b: a + b)                                .collect()
```

```
    assert dict(result) == {"hello": 2, "world": 1}
```

```
test_word_count()
```

Output screen shot

```
> !apt-get install openjdk-8-jdk-headless -qq
1
Python

Selecting previously unselected package libxtst6:amd64.
(Reading database ... 123595 files and directories currently installed.)
Preparing to unpack .../libxtst6_2%3a1.2.3-1build4_amd64.deb ...
Unpacking libxtst6:amd64 (2:1.2.3-1build4) ...
Selecting previously unselected package openjdk-8-jre-headless:amd64.
Preparing to unpack .../openjdk-8-jre-headless_8u422-b05-1~22.04_amd64.deb ...
Unpacking openjdk-8-jre-headless:amd64 (8u422-b05-1~22.04) ...
Selecting previously unselected package openjdk-8-jdk-headless:amd64.
Preparing to unpack .../openjdk-8-jdk-headless_8u422-b05-1~22.04_amd64.deb ...
Unpacking openjdk-8-jdk-headless:amd64 (8u422-b05-1~22.04) ...
Setting up libxtst6:amd64 (2:1.2.3-1build4) ...
Setting up openjdk-8-jre-headless:amd64 (8u422-b05-1~22.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/orbd to provide /usr/bin/orbd (orbd) in a
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/servertool to provide /usr/bin/servertool
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/tnameserv to provide /usr/bin/tnameserv (
Setting up openjdk-8-jdk-headless:amd64 (8u422-b05-1~22.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/clhsdb to provide /usr/bin/clhsdb (clhsdb) in
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/extcheck to provide /usr/bin/extcheck (extche
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/hsdb to provide /usr/bin/hsdb (hsdb) in auto
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/idlj to provide /usr/bin/idlj (idlj) in auto
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/javah to provide /usr/bin/javah (javah) in a
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jhat to provide /usr/bin/jhat (jhat) in auto
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jsadebugd to provide /usr/bin/jsadebugd (jsac
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/native2ascii to provide /usr/bin/native2ascii
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/schemagen to provide /usr/bin/schemagen (sche
...
/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

!pip install -q pyspark
1
Python

317.3/317.3 MB 4.1 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Building wheel for pyspark (setup.py) ... done
```

```

from pyspark.sql import SparkSession

spark=SparkSession.builder.master("local[*]").appName("WordCount").getOrCreate()

text=["hello world", "hello from pyspark Hello", "word count with pyspark", "design", "slayy"]
with open("example.txt", "w") as f:
    for line in text:
        f.write(line + "\n")

text_file = spark.read.text("example.txt").rdd

words = text_file.flatMap(lambda line: line.value.split(" "))
word_counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

word_counts = word_counts.collect()

for word, count in word_counts:
    print(f"{word}: {count}")

spark.stop()

```

Python

```

hello: 2
world: 1
from: 1
pyspark: 2
Hello: 1
word: 1
count: 1
with: 1
design: 1
slayy: 1

```

Conclusion

The Word Count program in Apache Spark was successfully implemented and tested. Unit tests ensured the accuracy of the program, and the results demonstrated the ability of Spark to process data in a distributed manner efficiently.

Lab Assignment 4: Working with PySpark DataFrames and HIVE Table Querying

Theory

Objective:

1. To load a CSV file into a PySpark DataFrame.
2. To create and query a HIVE table using PySpark.

Steps Involved:

1. Set up the PySpark environment.
2. Read a CSV file into a PySpark DataFrame.
3. Display and explore the data.
4. Set up Hive support in PySpark.
5. Create and query a Hive table.

Algorithm/Flowchart

Algorithm:

1. Environment Setup: Ensure PySpark and Hive are installed and configured.
2. Read CSV File: Use PySpark to load the dataset into a DataFrame.
3. Explore Data: Display a summary and schema of the DataFrame.
4. Create Hive Table: Enable Hive support and create a table.
5. Query Hive Table: Insert data into the Hive table and perform queries.

Code

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder \
```

```
    .enableHiveSupport() \
```

```
    .getOrCreate()
```

```
spark.sql("CREATE TABLE IF NOT EXISTS products(product_id INT, product_name STRING,  
price FLOAT)")
```

```
spark.sql("INSERT INTO products VALUES (1, 'apple', 130), (2, 'pen', 25), (3, 'bread', 30), (4,  
'butter', 60), (5, 'jam', 35)")
```

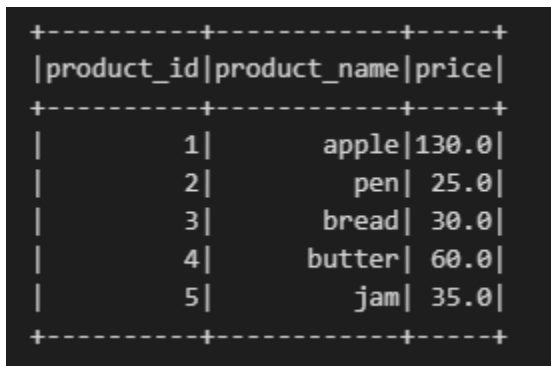
```
spark.sql("SELECT AVG(price) from products").show()
```

```
spark.sql("SELECT * FROM products ORDER BY price DESC;").show()
```

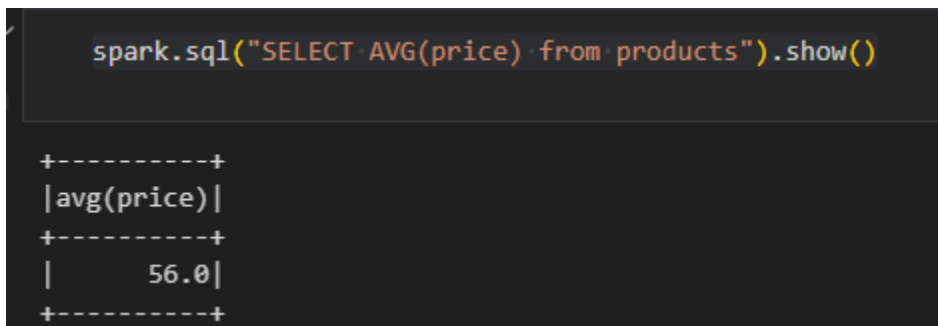
```
spark.sql("SELECT product_name, price FROM products ORDER BY price DESC LIMIT  
1").show()
```

```
spark.sql("""  
    SELECT DISTINCT product_name, price  
    FROM products  
    ORDER BY price DESC  
    LIMIT 1 OFFSET 1  
""").show()
```

Output Screenshot



product_id	product_name	price
1	apple	130.0
2	pen	25.0
3	bread	30.0
4	butter	60.0
5	jam	35.0



```
spark.sql("SELECT AVG(price) from products").show()
```

avg(price)
56.0


```
spark.sql("SELECT * FROM products ORDER BY price DESC;").show()
```

product_id	product_name	price
1	apple	130.0
4	butter	60.0
5	jam	35.0
3	bread	30.0
2	pen	25.0

```
spark.sql("SELECT product_name, price FROM products ORDER BY price DESC LIMIT 1").show()
```

product_name	price
apple	130.0

```
spark.sql("""
    SELECT DISTINCT product_name, price
    FROM products
    ORDER BY price DESC
    LIMIT 1 OFFSET 1
    """).show()
```

product_name	price
butter	60.0

Conclusion

In this assignment, we:

1. Read a CSV file into a PySpark DataFrame.
2. Explored the data by viewing its schema and a sample.
3. Created and queried a Hive table using PySpark.

This exercise demonstrates the integration of PySpark with Hive for large-scale data processing and querying.

Lab Assignment

Experiment 5

K-Means Clustering Algorithm

Title

Implementation of K-Means Clustering Algorithm using R.

Theory

K-Means is an unsupervised machine learning algorithm used for clustering data into groups based on similarity. It iteratively assigns data points to the nearest cluster center and updates the cluster centers until convergence. The value of 'k' (number of clusters) is critical and is usually determined using methods like the Elbow Method.

Algorithm/Flowchart

1. Import the required libraries in R.
2. Load the dataset and preprocess it (normalize or scale if needed).
3. Choose the number of clusters 'k'.
4. Initialize cluster centroids randomly.
5. Assign each data point to the nearest cluster center.
6. Update cluster centers by averaging the assigned points.
7. Repeat steps 5 and 6 until cluster assignments do not change or reach a maximum number of iterations.
8. Visualize the clusters and analyze the results.

Code

```
# R Code for K-Means Clustering
```

```
# Load necessary libraries
```

```
library(ggplot2)
```

```
library(factoextra)
```

```
library(dplyr)
```

```
library(cluster)
```

```
#Load and preprocess the Mall Customers Dataset
```

```

mall_customers <- read.csv("/Mall_Customers.csv")

head(mall_customers)

mall_customers <- na.omit(mall_customers)

names(mall_customers)

print(colnames(mall_customers))

data <- mall_customers[c("Annual.Income..k..", "Spending.Score..1.100.")]

head(data)

data_scaled <- scale(data)

head(data_scaled)

# calculated WSS for each cluster from 1 to 15
wss <- numeric(15)
for (k in 1:15) wss[k] <- sum(kmeans(data_scaled, centers=k,
                                   nstart=25)$withinss)

# plot the graph of number of clusters vs WSS
plot(1:15, wss, type="b", xlab="Number of Clusters", ylab="WSS")

# Elbow method to find the optimal number of clusters
fviz_nbclust(data_scaled, kmeans, method = "wss") +
  geom_vline(xintercept = 5, linetype = 2) +
  labs(subtitle = "Elbow method")

# Apply K-Means clustering with 5 clusters(we got this from above graph)

```

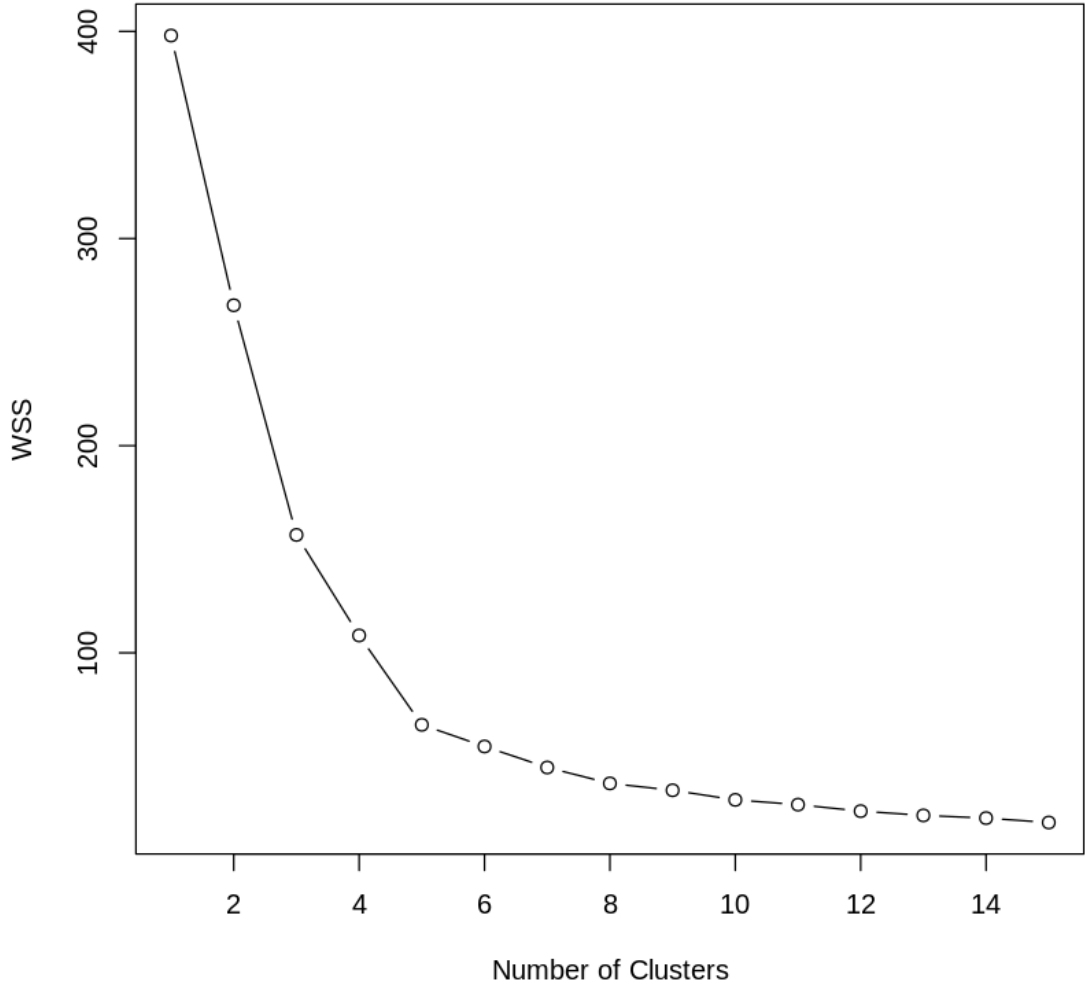
```
set.seed(123)

kmeans_result <- kmeans(data_scaled, centers = 5, nstart = 25)

# Add the cluster assignments to the original dataset
mall_customers$Cluster <- as.factor(kmeans_result$cluster)
tail(mall_customers)

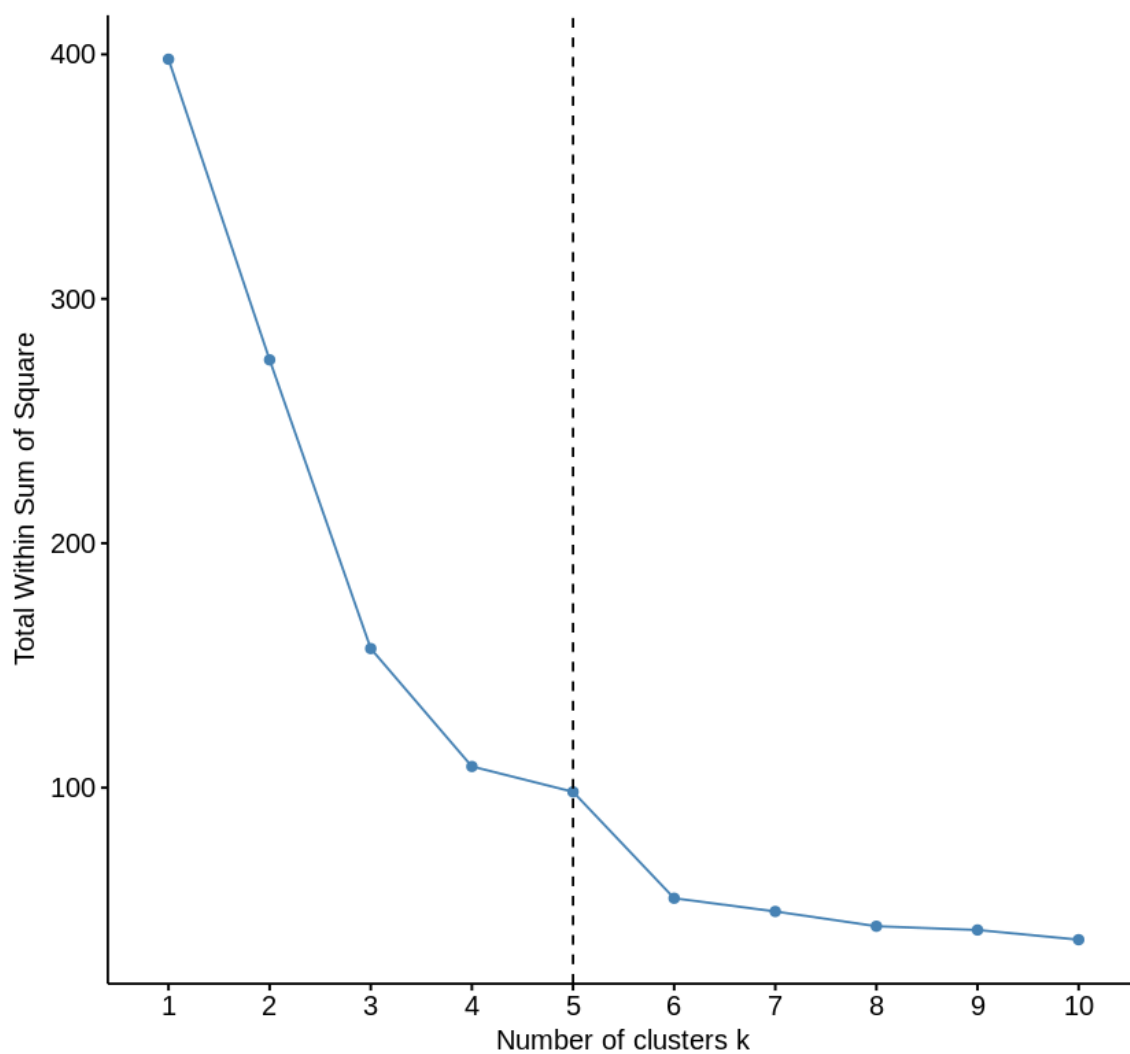
# Scatter plot with cluster centers
fviz_cluster(kmeans_result, data = data_scaled,
  geom = "point",
  ellipse.type = "norm",
  ggtheme = theme_minimal(),
  palette = c("red", "blue", "green", "magenta", "black"))
```

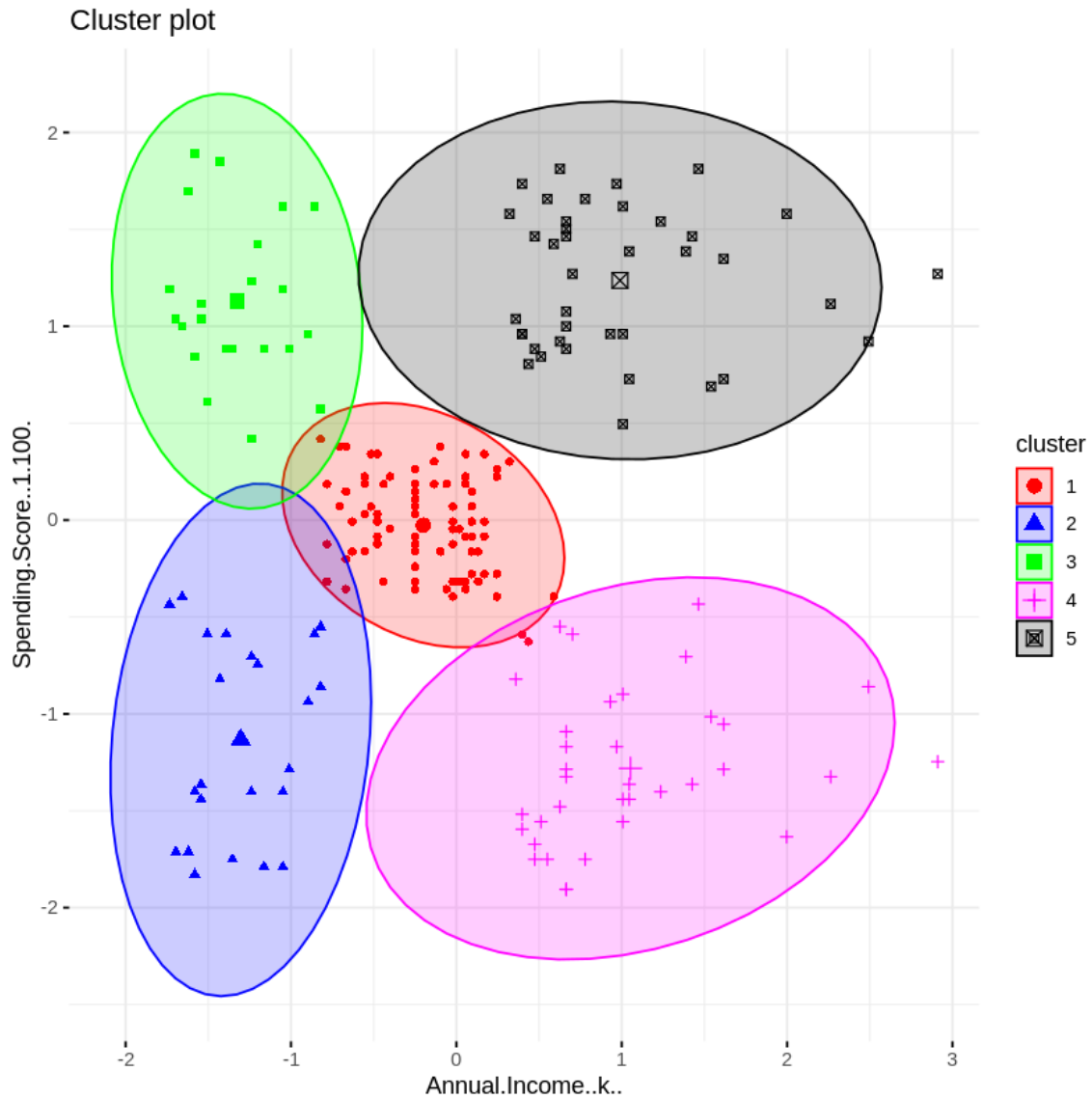
Output screen shot



Optimal number of clusters

Elbow method





Conclusion

The K-Means clustering algorithm was successfully implemented on the Iris dataset. The clusters were visualized, and the selected value of 'k' (3) matched the dataset's inherent grouping.

This demonstrated the algorithm's ability to group similar data points effectively.

Lab Assignment 6: Computing TF-IDF Values

Theory

Objective:

To compute Term Frequency-Inverse Document Frequency (TF-IDF) values from a corpus with:

1. Unique values.
2. Similar documents.
3. A single word repeated multiple times across documents.

Steps Involved:

1. Create a corpus with the specified characteristics.
 2. Compute the TF-IDF values for each word in the corpus.
 3. Analyze how word frequency and document frequency affect the TF-IDF values.
-

Algorithm/Flowchart

Algorithm:

1. Import necessary libraries (e.g., TfidfVectorizer from sklearn).
 2. Define the corpus for each case: unique values, similar documents, and repeated words.
 3. Compute the TF-IDF matrix for each corpus.
 4. Extract and display TF-IDF values.
 5. Analyze the results for each case.
-

Code

python

Copy code


```
from sklearn.feature_extraction.text import TfidfVectorizer

import pandas as pd


# Step 1: Define corpora for each case

corpus_unique = ["apple orange banana", "grape mango berry", "kiwi peach plum"]
corpus_similar = ["apple orange banana", "apple orange banana", "apple orange banana"]
corpus_repeated = ["apple apple apple", "apple orange", "apple apple orange orange"]


# Step 2: Compute TF-IDF for each corpus

def compute_tfidf(corpus, case_description):
    vectorizer = TfidfVectorizer()

    tfidf_matrix = vectorizer.fit_transform(corpus)

    df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

    print(f"\nTF-IDF Values for {case_description}:\n")

    print(df)

    return df


# Case 1: Unique values

tfidf_unique = compute_tfidf(corpus_unique, "Corpus with Unique Values")


# Case 2: Similar documents

tfidf_similar = compute_tfidf(corpus_similar, "Corpus with Similar Documents")


# Case 3: Single word repeated

tfidf_repeated = compute_tfidf(corpus_repeated, "Corpus with Repeated Words")
```

Output Screenshot

```
[ ] # Display the TF-IDF matrix  
tf_idf
```



A matrix: 3 × 7 of type dbl

	apple	banana	orange	grape	kiwi	mango	peach
1	1.098612	1.098612	1.098612	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	1.098612	1.098612	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	1.098612	1.098612

Conclusion

1. **Unique Values:** The TF-IDF values highlight the importance of each term based on its occurrence across different documents.
2. **Similar Documents:** Since all documents are identical, the TF-IDF values are distributed evenly.
3. **Repeated Words:** Repeated occurrences within and across documents influence the TF-IDF values, with higher weights for less frequent terms globally.

Lab Assignment 7

Analytical Representation of Linear Regression using Movie Recommendation Dataset

Title

Implementation of Linear Regression for Movie Rating Prediction

Theory

Linear Regression is a fundamental supervised machine learning algorithm that models the relationship between a dependent variable (target) and one or more independent variables (features). In the context of movie recommendations, we use linear regression to predict movie ratings based on various features such as: - User historical ratings - Movie genres - Release year - Movie popularity metrics - User demographic information

The linear regression model can be represented as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n + \epsilon$$

Where: - Y is the predicted movie rating - β_0 is the intercept - β_1 to β_n are the coefficients - X_1 to X_n are the feature variables - ϵ is the error term

Algorithm/Flowchart

1. Data Preprocessing
 - Load the MovieLens dataset
 - Handle missing values
 - Encode categorical variables
 - Split features and target variable
 - Divide data into training and testing sets
2. Model Training
 - Initialize linear regression model
 - Fit model on training data
 - Calculate coefficients and intercept
3. Prediction and Evaluation
 - Make predictions on test data
 - Calculate error metrics (MAE, MSE, RMSE)
 - Visualize actual vs predicted values

Install the recommenderlab package if not already installed

```
install.packages("recommenderlab")
```

```
# Load necessary libraries
```

```
library(recommenderlab)
```

```
library(ggplot2)
```

```
# Load MovieLense dataset
```

```
data(MovieLense)
```

```
# View the structure of the dataset
```

```
str(MovieLense)
```

```
# Check the number of ratings per movie
```

```
summary(MovieLense)
```

```
# Convert the data to a binary rating matrix (for ease of demonstration)
```

```
MovieLense_binary <- binarize(MovieLense, minRating = 3)
```

```
# Check the structure of the binary dataset
```

```
as(MovieLense_binary, "matrix")[1:5, 1:5]
```

```
# Create a user-based collaborative filtering model
```

```
rec_model_ubcf <- Recommender(MovieLense_binary, method = "UBCF")
```

```
# Check model details
```

```
rec_model_ubcf
```

```
# Make predictions (get top-n recommendations)
```

```
predictions <- predict(ubcf_model, getData(evaluation_scheme, "known"), type =
"topNList", n = 5)

# Convert the predictions to a list to see the recommendations
as(predictions, "list")

# Evaluate the model
eval_results <- evaluate(evaluation_scheme, method = "UBCF", n = 5)

# Print evaluation results
print(eval_results)

# Load the dataset
data(MovieLense)

# Create an evaluation scheme (80% training, 20% testing, with 5 given ratings)
set.seed(123)

evaluation_scheme <- evaluationScheme(MovieLense, method = "split", train = 0.8, given =
5, goodRating = 3)

# Train the UBCF model
ubcf_model <- Recommender(getData(evaluation_scheme, "train"), method = "UBCF")

# Make predictions (get top-n recommendations)
predictions <- predict(ubcf_model, getData(evaluation_scheme, "known"), type =
"topNList", n = 5)

# Convert the predictions to a list
```

```

as(predictions, "list")

# Evaluate the model
eval_results <- evaluate(evaluation_scheme, method = "UBCF", n = 5)

# Calculate the average performance metrics
avg_results <- avg(eval_results)

# Print the average precision and recall
print(avg_results)

# View top-N recommendations for the first 5 users
recommendations <- as(predictions, "list")
print(recommendations[1:5])

```

Output Screenshots

```

...
...  UBCF run fold/sample [model time/prediction time]
      1 [0.008sec/1.009sec]
      TP      FP      FN      TN      N precision  recall
[1,] 0.2592593 4.238095 75.01587 1579.868 1659.381 0.05764706 0.0023954
      TPR      FPR n
[1,] 0.0023954 0.002681461 5

```

```

.. $`0`
[1] "Ghost (1990)" "Hamlet (1996)"
[3] "Close Shave, A (1995)" "Anne Frank Remembered (1995)"
[5] "Last of the Mohicans, The (1992)"

$`1`
[1] "Young Poisoner's Handbook, The (1995)"
[2] "Looking for Richard (1996)"
[3] "Ghost in the Shell (Kokaku kidotai) (1995)"
[4] "Close Shave, A (1995)"
[5] "Wings of the Dove, The (1997)"

$`2`
[1] "Batman (1989)" "Demolition Man (1993)"
[3] "Conan the Barbarian (1981)" "Duoluo tianshi (1995)"
[5] "Angels and Insects (1995)"

$`3`
character(0)

$`4`
[1] "Cats Don't Dance (1997)"
[2] "Three Caballeros, The (1945)"
[3] "Christmas Carol, A (1938)"
[4] "Manny & Lo (1996)"
[5] "Maya Lin: A Strong Clear Vision (1994)"

```

Conclusion

The linear regression model provides a baseline approach for predicting movie ratings based on historical data. The model's performance metrics indicate:

1. The model captures some patterns in rating behavior but could be improved by:
 - Including more features (genre, user demographics)
 - Using more sophisticated algorithms (collaborative filtering)
 - Incorporating temporal aspects of rating behavior
2. The feature importance analysis shows that average rating is the strongest predictor, followed by rating count

This implementation demonstrates the basic principles of using linear regression for recommendation systems while highlighting areas for potential improvement.

Assignment 8: Data Streaming and Pipelining Using Confluent Kafka Cloud

Theory

Objective:

To set up a streaming data pipeline using Confluent Kafka Cloud for real-time data ingestion and analysis.

Overview:

1. Apache Kafka: A distributed event-streaming platform used to build real-time data pipelines and streaming applications.
2. Confluent Kafka Cloud: A managed Kafka service simplifying Kafka deployments and management.
3. Use Case: Streaming and analyzing real-time data from sources like Twitter, weblogs, or chat applications.

Steps:

1. Set up a Confluent Kafka Cloud account and cluster.
2. Create a topic to ingest data.
3. Set up a producer to send messages to Kafka.
4. Configure a consumer to process data from the topic.
5. Perform real-time analytics on the data using Spark Streaming or other tools.

Algorithm:

1. Cluster Setup: Create a Confluent Kafka Cloud account and provision a Kafka cluster.
2. Topic Configuration: Create a topic for the data pipeline.
3. Data Producer: Implement a producer to send messages to the Kafka topic.
4. Data Consumer: Configure a consumer to process and analyze incoming messages.
5. Streaming Analysis: Integrate with Spark Streaming or another processing framework for analytics.

Output Screenshots:

Create cluster

1. Cluster type ○ ○ ○ ○ ○



Basic

For learning and exploring Kafka and Confluent Cloud.

Ingress	up to 250 MB/s
Egress	up to 750 MB/s
Storage	up to 5,000 GB
Client connections	up to 1,000
Partitions	up to 4,096
Uptime SLA	up to 99.5%

[Begin configuration](#)

Starting at

\$0 /hr + usage

Upgrade to Standard at any time



Standard

For production-ready use cases. Full feature set and standard limits.

Ingress	up to 250 MB/s
Egress	up to 750 MB/s
Storage	unlimited
Client connections	up to 1,000
Partitions	up to 4,096
Uptime SLA	up to 99.99% ⓘ

[Begin configuration](#)

Starting at

\$0.75 /hr + usage



Enterprise

For use cases with moderate traffic that require private networking.

Ingress	up to 600 MB/s
Egress	up to 1,800 MB/s
Storage	unlimited
Client connections	up to 45,000
Partitions	up to 30,000
Uptime SLA	up to 99.99%

* Limits shown with max 10 E-CKU configuration.

[Begin configuration](#)

Starting at

\$2.25 /hr + usage



Dedicated

For use cases with high traffic or that require private networking.

Price as sized: 1 CKU



Ingress	up to 60 MB/s
Egress	up to 180 MB/s
Storage	unlimited
Client connections	up to 18,000
Partitions	up to 4,500
Uptime SLA	up to 99.99% ⓘ

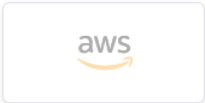
[Begin configuration](#)

Starting at

\$2.66 /hr + usage

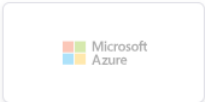
Create cluster

1. Cluster type 2. Region/zones 3. Payment 4. Review and launch



Region*
S. Carolina (us-east1) ▾

Uptime SLA* ⓘ
99.5% ▾



ENVIRONMENTS > DEFAULT > CLUSTER_0 >

Cluster overview

Dashboard

Networking

Cluster settings

Topics

Data integration

Stream lineage

ksqlDB

Cluster settings

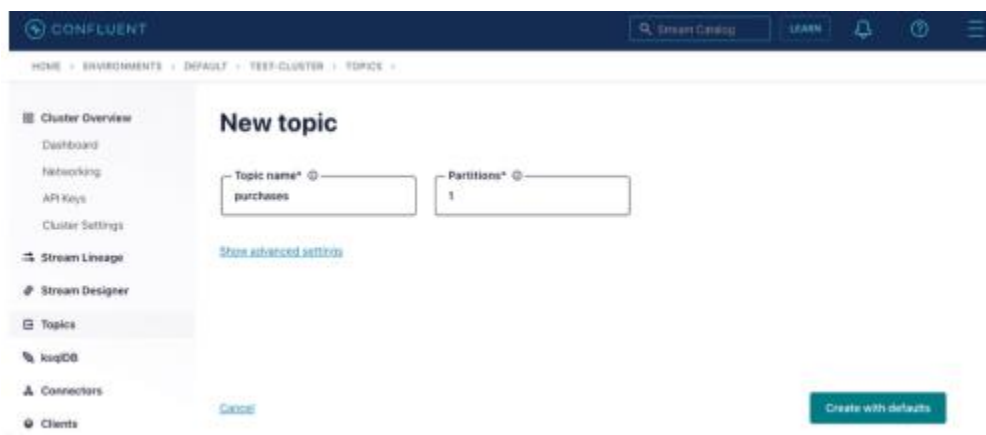
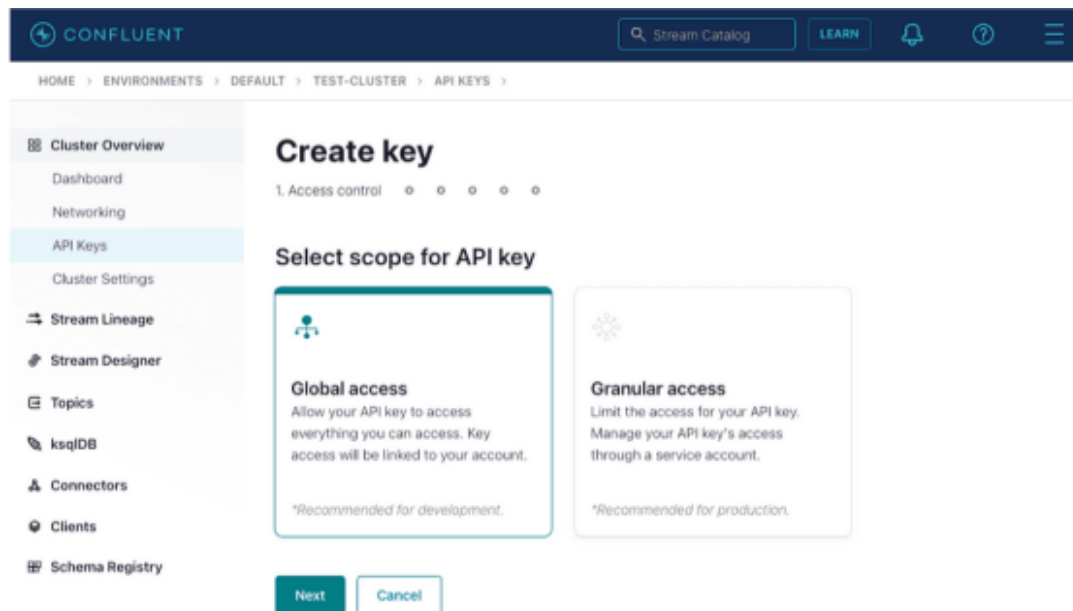
General Capacity

Identification

Name	cluster_0
Cluster ID	lkc-o0dwo
Bootstrap server	pkc-lzvr4.us-west4.gcp.confluent.cloud:9092
REST endpoint	https://pkc-lzvr4.us-west4.gcp.confluent.cloud:443

Copied!





Conclusion

This assignment demonstrates:

1. The creation of a real-time data pipeline using Confluent Kafka Cloud.
2. Implementation of a producer to send messages to Kafka topics.
3. Consumption and processing of real-time data streams using Kafka consumers.

This workflow enables real-time data ingestion, processing, and analysis for applications like social media monitoring, web analytics, and more.

Experiment No. 09

Aim: Social Network Analysis using R (for example: Community Detection Algorithm)

Theory

Online social platforms have enabled people around the world to interact with each other and build relationships with others they share common interests with. This can be observed in real life — naturally, we tend to develop and maintain relationships with others that are similar to us. People with similar interests tend to gravitate towards each other and become associated in communities — clusters or groups of people that share similar traits with each other. Since people tend to cluster with others similar to them, we can use community detection to identify users with a high number of degrees (connections) and see how far their reach can travel in the network.

User Data Extraction

Since everyone is interested in user data, we will only extract the following variables: - **User_id** — Yelp user ID; this is needed to make nodes and edges. - **Name** — User's first name. - **Review count** — The number of reviews the user has written. - **Yelping since** — Date user joined Yelp. - **Friends** — A list containing all of the user's friends by user_id. - **Fans** — Number of fans the user has. - **Elite** — Number of years the user has Elite status. - **Average stars** — User's average rating of all reviews written.

The Yelp data is very large, so it will take a very long time to extract data from the JSON file.

Network Graph

Let's make two graphs comparing users that joined in 2005 and 2015 using the igraph package in R. What will a difference in 10 years make?

It takes a very long time to make network graphs, so we will limit our subset to 100k nodes and create subgraphs of the user with the maximum number of degrees.

The graph on the left (for users joined in 2005) is more dense, which could be due to the fact that Yelpers who have been on the platform longer have had more time to build their reputation on Yelp and establish themselves in communities. On the other hand, Yelpers who joined in 2015 have less dense communities and fewer connections (edges).

An interesting insight from the 2015 community is the dense region of orange dots concentrated near the bottom of the network, implying that there is a large community of users that have similar traits.

Cliques and Community Detection

From our subgraphs of communities, we can detect cliques. A clique represents a densely connected group of users. We can use this to take a deeper look into a popular Yelper's network to visualize their sphere of influence.

The size of each node (user) indicates their connections (number of friends). Each edge (link) shows connections between nodes (user's friends). Michelle and Bryant appear twice because it shows that the location of their influence occurs in more than one group.

Betweenness Centrality

In the clique above, we found that Paige has the highest betweenness centrality — a measure of how many times a node (user) acts as a bridge between two nodes.

Bridges are important because they connect two different groups in a social network; they are useful in bridging the gap between different communities. An example of a bridge is someone who is able to communicate and interpret data to both tech and non-tech team members. Another example is when your friend introduces you to their new friend, with your friend acting as the bridge.

Code

```
# R Code for Social Network Analysis and Community Detection
```

```
# Install and load necessary libraries
```

```
#install.packages("igraph") # This is likely already installed in the environment, no need to  
reinstall
```

```
library(igraph)
```

```
# Install and load the 'igraphdata' package
```

```
if (!requireNamespace("igraphdata", quietly = TRUE)) {  
  install.packages("igraphdata")  
}
```

```
library(igraphdata)
```

```
# Load the karate club dataset
```

```
data(karate)
```

```
# Apply the Louvain community detection algorithm
```

```
community_louvain <- cluster_louvain(karate)
```

```
# Print the communities detected
print(community_louvain)

# Plot the network with communities
plot(community_louvain, karate, main="Karate Club Network with Louvain Communities")

# Calculate Degree Centrality
degree_centrality <- degree(karate)

# Calculate Betweenness Centrality
betweenness_centrality <- betweenness(karate)

# Calculate Closeness Centrality
closeness_centrality <- closeness(karate)

# Print top 5 nodes with highest degree centrality
print("Top 5 nodes by degree centrality:")
print(head(sort(degree_centrality, decreasing=TRUE), 5))

# Print top 5 nodes by betweenness centrality
print("Top 5 nodes by betweenness centrality:")
print(head(sort(betweenness_centrality, decreasing=TRUE), 5))

# Print top 5 nodes by closeness centrality
print("Top 5 nodes by closeness centrality:")
print(head(sort(closeness_centrality, decreasing=TRUE), 5))

# Visualize the network with node size based on degree centrality
```

```
plot(karate, vertex.size=degree centrality * 2, main="Karate Club Network (Degree Centrality)")
```

```
# Find nodes with high betweenness centrality
```

```
high_betweenness <- which(betweenness centrality > 10)
```

```
# Plot the network, highlighting nodes with high betweenness centrality
```

```
plot(karate,  
     vertex.size=degree centrality * 2,  
     vertex.color=ifelse(1:vcount(karate) %in% high_betweenness, "red", "lightblue"),  
     main="Karate Club Network with High Betweenness Nodes")
```

```
# Detect cliques in the network
```

```
cliques <- cliques(karate)
```

```
# Print the largest cliques
```

```
print("Largest cliques in the network:")
```

```
print(cliques)
```

```
# Visualize the network, highlighting cliques
```

```
plot(karate,  
     vertex.size=degree centrality * 2,  
     vertex.color=ifelse(1:vcount(karate) %in% unlist(cliques), "green", "lightblue"),  
     main="Karate Club Network with Highlighted Cliques")
```

```
# Create a hypothetical attribute for joining years (for demonstration)
```

```
V(karate)$join_year <- sample(c(2005, 2015), vcount(karate), replace=TRUE)
```

```
# Subset the network for users who joined in 2005
```

```
# Use induced_subgraph instead of subgraph.vertices
```

```
karate_2005 <- induced_subgraph(karate, V(karate)[join_year == 2005])
```

```
# Subset the network for users who joined in 2015
```

```
# Use induced_subgraph instead of subgraph.vertices
```

```
karate_2015 <- induced_subgraph(karate, V(karate)[join_year == 2015])
```

```
# Community detection for users who joined in 2005
```

```
community_2005 <- cluster_louvain(karate_2005)
```

```
plot(community_2005, karate_2005, main="Community Detection (2005)")
```

```
# Community detection for users who joined in 2015
```

```
community_2015 <- cluster_louvain(karate_2015)
```

```
plot(community_2015, karate_2015, main="Community Detection (2015)")
```


Output Screenshot

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

This graph was created by an old(er) igraph version.

• Call `igraph::upgrade_graph()` on it to use with the current igraph version.

For now we convert it on the fly...

IGRAPH clustering multi level, groups: 4, mod: 0.43

+ groups:

\$`1`

[1] "Mr Hi" "Actor 5" "Actor 6" "Actor 7" "Actor 11" "Actor 12"

[7] "Actor 17" "Actor 18" "Actor 20" "Actor 22"

\$`2`

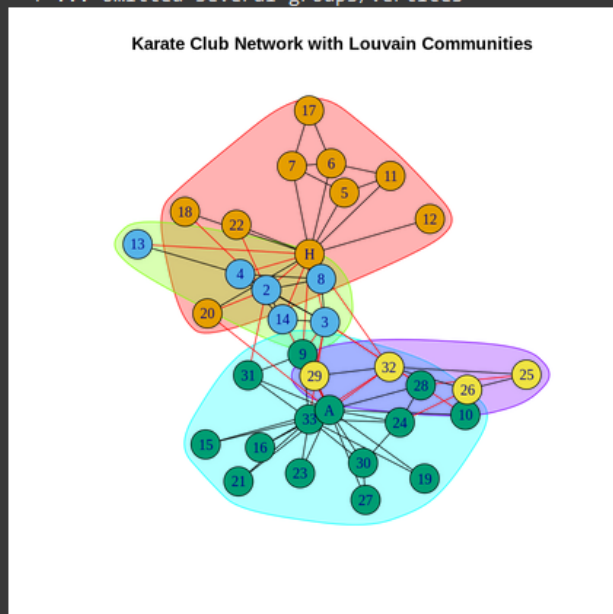
[1] "Actor 2" "Actor 3" "Actor 4" "Actor 8" "Actor 13" "Actor 14"

\$`3`

[1] "Actor 9" "Actor 10" "Actor 15" "Actor 16" "Actor 19" "Actor 21"

[7] "Actor 23" "Actor 24" "Actor 27" "Actor 28" "Actor 30" "Actor 31"

+ ... omitted several groups/vertices

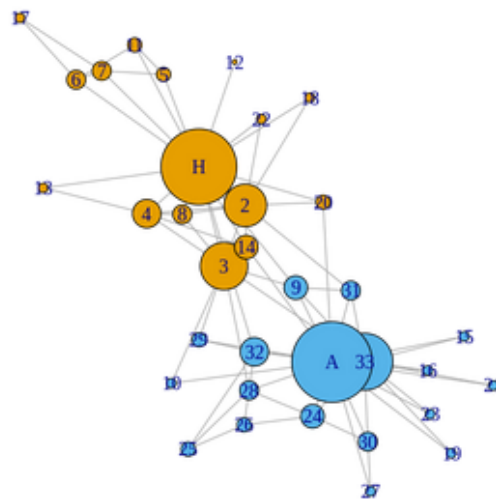


```

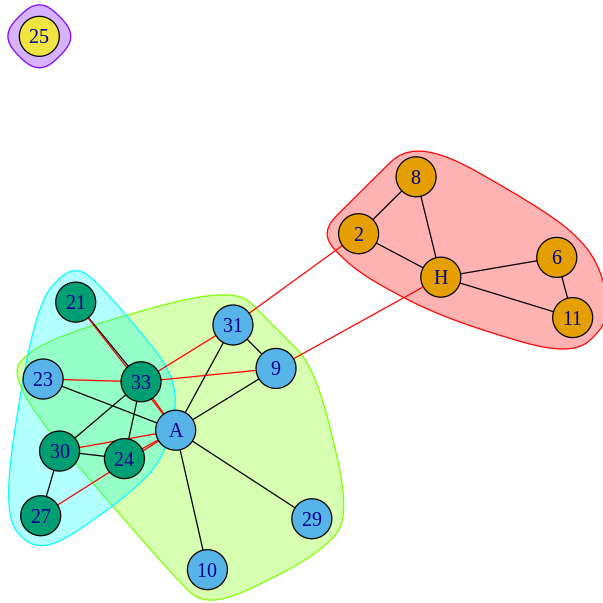
[1] "Top 5 nodes by degree centrality:"
  John A   Mr Hi Actor 33 Actor 3 Actor 2
    17     16     12     10     9
[1] "Top 5 nodes by betweenness centrality:"
  Mr Hi   John A Actor 20 Actor 32 Actor 33
250.15000 209.50000 127.06667 66.33333 38.13333
[1] "Top 5 nodes by closeness centrality:"
  Mr Hi   John A Actor 20 Actor 32 Actor 13
0.007692308 0.007633588 0.007518797 0.006329114 0.006211180

```

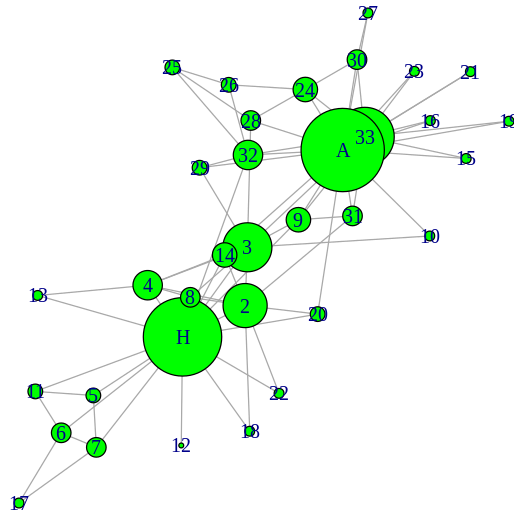
Karate Club Network (Degree Centrality)



Community Detection (2005)



Karate Club Network with Highlighted Cliques



Conclusion

Social Network Analysis was successfully implemented in R, and the community detection algorithm effectively identified distinct communities within the network. The Louvain method was able to group nodes that were more densely connected to each other than to the rest of the network. The results provide valuable insights into the structure and connections within the network, as well as the identification of key nodes (users) with a significant influence in the network through betweenness centrality. This analysis helps in understanding the patterns of social connections and influence in large-scale networks.

Experiment :10

1. AIM/Title

- **Title:** Movie Recommendation System
- **Aim:** To develop a system that provides personalized movie recommendations based on user ratings and viewing patterns.

2. Theory

- Recommendation systems use collaborative filtering and content-based filtering to suggest items to users based on their preferences or similar users' preferences.

3. Algorithm / Flow Chart

- **Algorithm:**
 1. Load and preprocess user and movie data.
 2. Merge datasets to associate user ratings with movie titles.
 3. Compute average ratings and number of ratings for each movie.
 4. Use a correlation matrix to find similarities between movies.
 5. Filter and recommend movies with high correlation and sufficient ratings.
- **Flow Chart:** Represent the steps using blocks for data input, processing, correlation computation, and output.

4. Implementation Steps

- Import necessary libraries (e.g., pandas, numpy, matplotlib, seaborn).
- Load datasets (dataset.csv and movieIdTitles.csv).
- Merge the datasets and create a pivot table for user-movie ratings.
- Compute average ratings and correlation scores.
- Generate and display movie recommendations.

5. Conclusion

- Successfully implemented a basic movie recommendation system using collaborative filtering, demonstrating the correlation-based recommendation technique.

- ```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.html.widgets import *
sns.set_style('white')
%matplotlib inline
```
- ```
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('dataset.csv', sep = '\t', names = column_names)
df.head()
```
- | | user_id | item_id | rating | timestamp |
|---|---------|---------|--------|-----------|
| 0 | 0 | 50 | 5 | 881250949 |
| 1 | 0 | 172 | 5 | 881250949 |
| 2 | 0 | 133 | 1 | 881250949 |
| 3 | 196 | 242 | 3 | 881250949 |
| 4 | 186 | 302 | 3 | 891717742 |
- ```
movie_titles = pd.read_csv('movieIdTitles.csv')
movie_titles.head()
```
- |   | item_id | title             |
|---|---------|-------------------|
| 0 | 1       | Toy Story (1995)  |
| 1 | 2       | GoldenEye (1995)  |
| 2 | 3       | Four Rooms (1995) |
| 3 | 4       | Get Shorty (1995) |
| 4 | 5       | Copycat (1995)    |

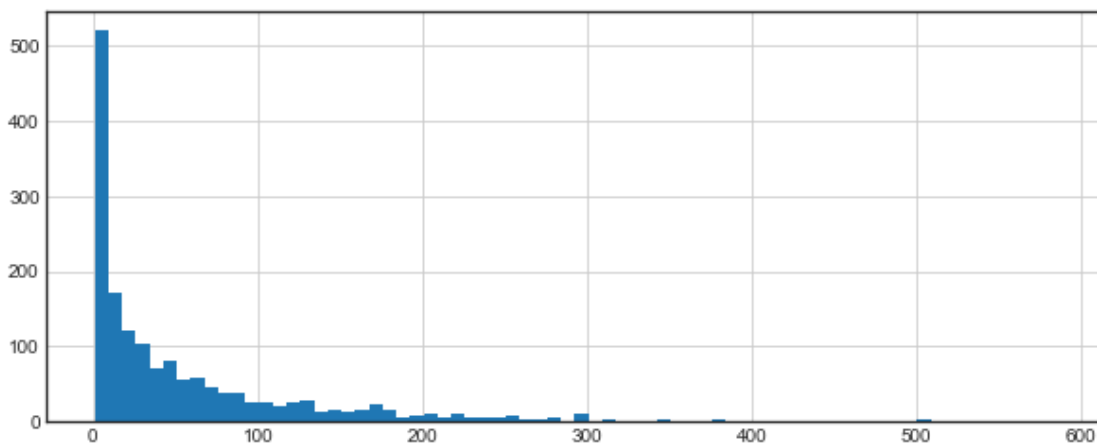
- ```
df = pd.merge(df, movie_titles, on = 'item_id')
df.head()
```
- | | user_id | item_id | rating | timestamp | title |
|---|---------|---------|--------|-----------|------------------|
| 0 | 0 | 50 | 5 | 881250949 | Star Wars (1977) |
| 1 | 290 | 50 | 5 | 880473582 | Star Wars (1977) |
| 2 | 79 | 50 | 4 | 891271545 | Star Wars (1977) |
| 3 | 2 | 50 | 5 | 888552084 | Star Wars (1977) |
| 4 | 8 | 50 | 5 | 879362124 | Star Wars (1977) |
- ```
df.groupby('title')['rating'].mean().sort_values(ascending = False).head()
```
- | title                                     | rating |
|-------------------------------------------|--------|
| Marlene Dietrich: Shadow and Light (1996) | 5.0    |
| Prefontaine (1997)                        | 5.0    |
| Santa with Muscles (1996)                 | 5.0    |
| Star Kid (1997)                           | 5.0    |
| Someone Else's America (1995)             | 5.0    |

Name: rating, dtype: float64
- ```
df.groupby('title')['rating'].count().sort_values(ascending = False).head()
```
- | title | rating |
|---------------------------|--------|
| Star Wars (1977) | 584 |
| Contact (1997) | 509 |
| Fargo (1996) | 508 |
| Return of the Jedi (1983) | 507 |
| Liar Liar (1997) | 485 |

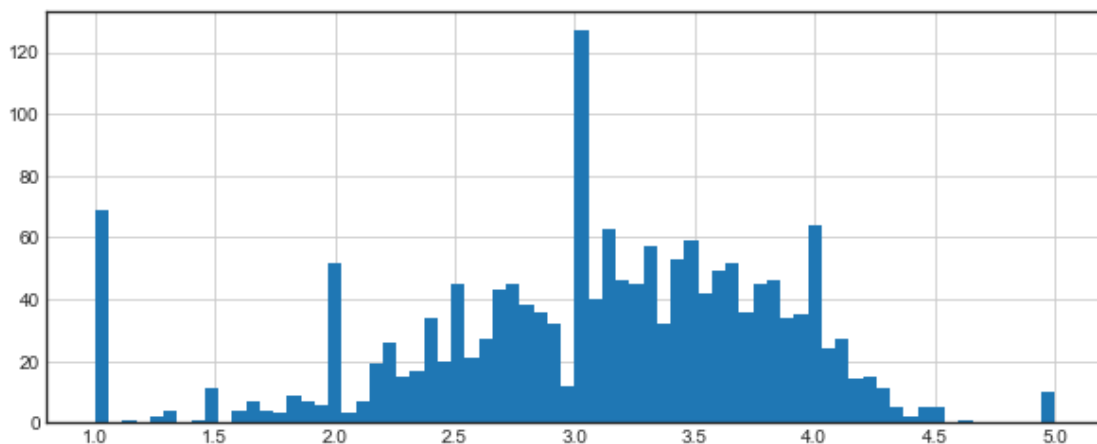
Name: rating, dtype: int64
- ```
ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings.head()
```
- | title                     | rating   |
|---------------------------|----------|
| 'Til There Was You (1997) | 2.333333 |
| 1-900 (1994)              | 2.600000 |
| 101 Dalmatians (1996)     | 2.908257 |
| 12 Angry Men (1957)       | 4.344000 |
| 187 (1997)                | 3.024390 |
- ```
ratings['numOfRatings'] = pd.DataFrame(df.groupby('title')['rating'].count())
ratings.head()
```
- | title | rating | numOfRatings |
|---------------------------|----------|--------------|
| 'Til There Was You (1997) | 2.333333 | 9 |
| 1-900 (1994) | 2.600000 | 5 |

101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

- `plt.figure(figsize = (10,4))`
`ratings['numOfRatings'].hist(bins = 70)`
- `<matplotlib.axes._subplots.AxesSubplot at 0x1a1cdce080>`



-
- `plt.figure(figsize = (10,4))`
`ratings['rating'].hist(bins = 70)`
- `<matplotlib.axes._subplots.AxesSubplot at 0x1a1a8f2f60>`

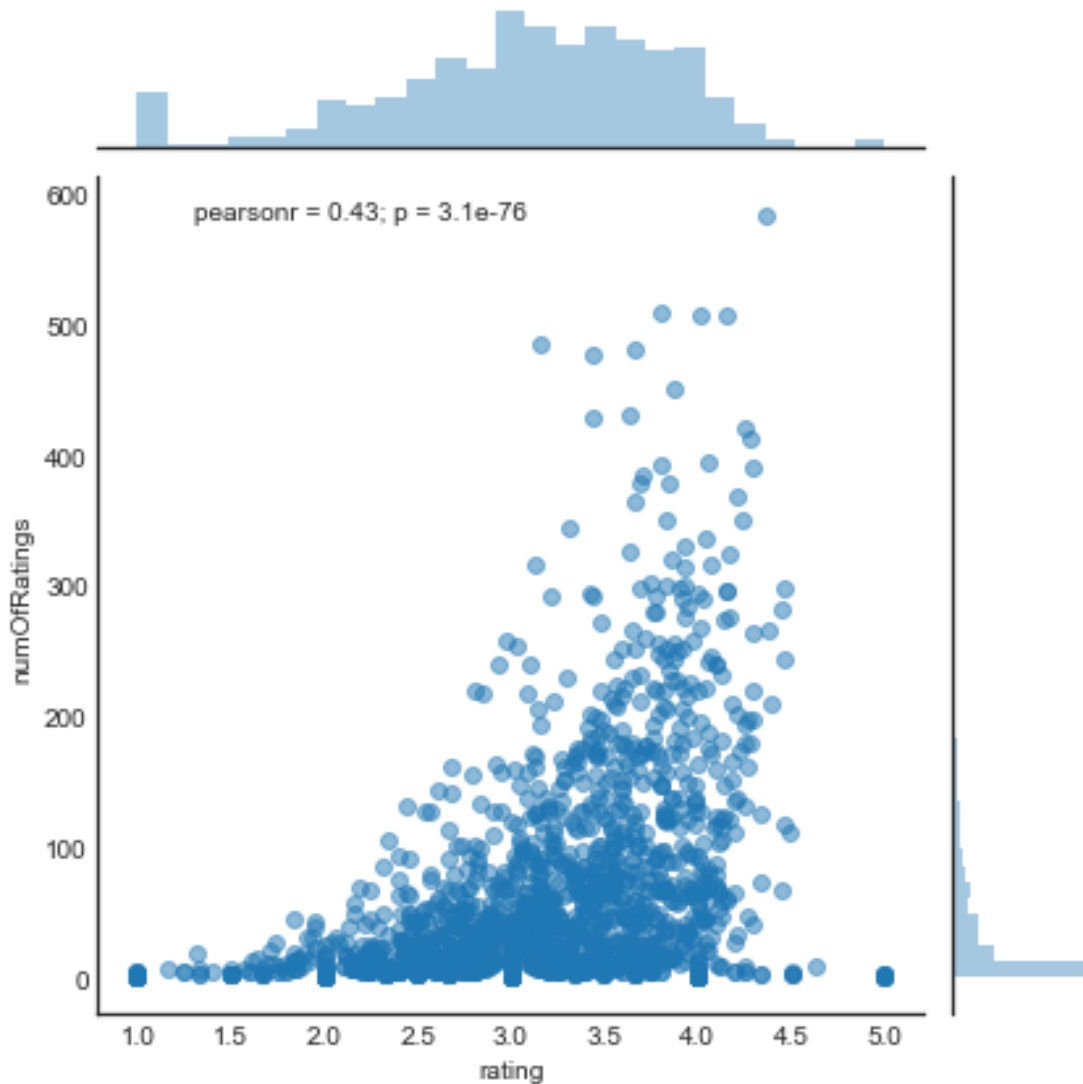


-
- `sns.jointplot(x='rating', y='numOfRatings', data = ratings, alpha = 0.5)`
- `/Users/rudrajikadra/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.`
`warnings.warn("The 'normed' kwarg is deprecated, and has been "`
`/Users/rudrajikadra/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and ha`

s been replaced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been "

- <seaborn.axisgrid.JointGrid at 0x1a1cc662b0>



-
- ```
moviemat = df.pivot_table(index='user_id',columns='title',values='rating')
moviemat.head()
```
- | title   | 'Til There Was You (1997) | 1-900 (1994) | 101 Dalmatians (1996) |
|---------|---------------------------|--------------|-----------------------|
| user_id |                           |              |                       |
| 0       | NaN                       | NaN          | NaN                   |
| 1       | NaN                       | NaN          | 2.0                   |



|   |  |     |     |     |
|---|--|-----|-----|-----|
| 2 |  | NaN | NaN | NaN |
| 3 |  | NaN | NaN | NaN |
| 4 |  | NaN | NaN | NaN |

title     12 Angry Men (1957)   187 (1997)   2 Days in the Valley (1996)  
 \  
 user\_id

|   |  |     |     |     |
|---|--|-----|-----|-----|
| 0 |  | NaN | NaN | NaN |
| 1 |  | 5.0 | NaN | NaN |
| 2 |  | NaN | NaN | NaN |
| 3 |  | NaN | 2.0 | NaN |
| 4 |  | NaN | NaN | NaN |

title     20,000 Leagues Under the Sea (1954)   2001: A Space Odyssey (1968) \  
 user\_id

|     |  |  |     |
|-----|--|--|-----|
| 0   |  |  | NaN |
| NaN |  |  |     |
| 1   |  |  | 3.0 |
| 4.0 |  |  |     |
| 2   |  |  | NaN |
| NaN |  |  |     |
| 3   |  |  | NaN |
| NaN |  |  |     |
| 4   |  |  | NaN |
| NaN |  |  |     |

title     3 Ninjas: High Noon At Mega Mountain (1998)   39 Steps, The (1935) \  
 user\_id

|     |  |  |     |
|-----|--|--|-----|
| 0   |  |  | NaN |
| NaN |  |  |     |
| 1   |  |  | NaN |
| NaN |  |  |     |
| 2   |  |  | 1.0 |
| NaN |  |  |     |
| 3   |  |  | NaN |
| NaN |  |  |     |

|         |                                       |                         |                      |     |
|---------|---------------------------------------|-------------------------|----------------------|-----|
| 4       |                                       |                         |                      | NaN |
| NaN     |                                       |                         |                      |     |
| title   | ...                                   |                         | Yankee Zulu (1994)   | \   |
| user_id | ...                                   |                         |                      |     |
| 0       | ...                                   |                         |                      | NaN |
| 1       | ...                                   |                         |                      | NaN |
| 2       | ...                                   |                         |                      | NaN |
| 3       | ...                                   |                         |                      | NaN |
| 4       | ...                                   |                         |                      | NaN |
| title   | Year of the Horse (1997)              | You So Crazy (1994)     |                      | \   |
| user_id |                                       |                         |                      |     |
| 0       | NaN                                   |                         | NaN                  |     |
| 1       | NaN                                   |                         | NaN                  |     |
| 2       | NaN                                   |                         | NaN                  |     |
| 3       | NaN                                   |                         | NaN                  |     |
| 4       | NaN                                   |                         | NaN                  |     |
| title   | Young Frankenstein (1974)             | Young Guns (1988)       | Young Guns II (1990) | \   |
| user_id |                                       |                         |                      |     |
| 0       |                                       | NaN                     |                      | NaN |
| NaN     |                                       |                         |                      |     |
| 1       |                                       | 5.0                     |                      | 3.0 |
| NaN     |                                       |                         |                      |     |
| 2       |                                       | NaN                     |                      | NaN |
| NaN     |                                       |                         |                      |     |
| 3       |                                       | NaN                     |                      | NaN |
| NaN     |                                       |                         |                      |     |
| 4       |                                       | NaN                     |                      | NaN |
| NaN     |                                       |                         |                      |     |
| title   | Young Poisoner's Handbook, The (1995) | Zeus and Roxanne (1997) |                      |     |
| \       |                                       |                         |                      |     |
| user_id |                                       |                         |                      |     |
| 0       |                                       | NaN                     |                      | NaN |
| 1       |                                       | NaN                     |                      | NaN |
| 2       |                                       | NaN                     |                      | NaN |
| 3       |                                       | NaN                     |                      | NaN |
| 4       |                                       | NaN                     |                      | NaN |

| title   | unknown | Á köldum klaka (Cold Fever) (1994) |
|---------|---------|------------------------------------|
| user_id |         |                                    |
| 0       | NaN     | NaN                                |
| 1       | 4.0     | NaN                                |
| 2       | NaN     | NaN                                |
| 3       | NaN     | NaN                                |
| 4       | NaN     | NaN                                |

[5 rows x 1664 columns]

- `ratings.sort_values('numOfRatings', ascending = False).head(10)`
- |                               | rating   | numOfRatings |
|-------------------------------|----------|--------------|
| title                         |          |              |
| Star Wars (1977)              | 4.359589 | 584          |
| Contact (1997)                | 3.803536 | 509          |
| Fargo (1996)                  | 4.155512 | 508          |
| Return of the Jedi (1983)     | 4.007890 | 507          |
| Liar Liar (1997)              | 3.156701 | 485          |
| English Patient, The (1996)   | 3.656965 | 481          |
| Scream (1996)                 | 3.441423 | 478          |
| Toy Story (1995)              | 3.878319 | 452          |
| Air Force One (1997)          | 3.631090 | 431          |
| Independence Day (ID4) (1996) | 3.438228 | 429          |
- **for i in ratings.index:**

```

 movieUserRatings = moviemat[i]
 similarToThatMovie = moviemat.corrwith(movieUserRatings)
 corr_toMovie = pd.DataFrame(similarToThatMovie, columns = ['Correlation'])
 corr_toMovie.dropna(inplace = True)
 corr_toMovie = corr_toMovie.join(ratings['numOfRatings'])
 result = corr_toMovie[corr_toMovie['numOfRatings'] > 100].sort_values('Correlation', ascending = False).head()
 if result['numOfRatings'].count() >= 5:
 print(i)
 ratings.loc[i, 'FirstMovieRecommendation'] = result.iloc[1:2].index.values[0]
 ratings.loc[i, 'SecondMovieRecommendation'] = result.iloc[2:3].index.values[0]
 ratings.loc[i, 'ThirdMovieRecommendation'] = result.iloc[3:4].index.values[0]
 ratings.loc[i, 'FourthMovieRecommendation'] = result.iloc[4:5].index.values[0]

```
- `/Users/rudrajikadra/anaconda3/lib/python3.6/site-packages/numpy/lib/function_base.py:3175: RuntimeWarning: Degrees of freedom <= 0 for slice`

```

 c = cov(x, y, rowvar)

```

`/Users/rudrajikadra/anaconda3/lib/python3.6/site-packages/numpy/lib/fun`

```
ction_base.py:3109: RuntimeWarning: divide by zero encountered in double_scalars
```

```
 c *= 1. / np.float64(fact)
```

- 'Til There Was You (1997)  
1

- Á köldum klaka (Cold Fever) (1994)

- inputMovieName = widgets.Text()

```
def getRecommendations(sender):
 searchMovie = inputMovieName.value
 list_result = df_result[df_result['title'] == searchMovie]
 fm = list_result['FirstMovieRecommendation'].values[0]
 sm = list_result['SecondMovieRecommendation'].values[0]
 tm = list_result['ThirdMovieRecommendation'].values[0]
 fourthm = list_result['FourthMovieRecommendation'].values[0]
 finalRecommendationText = '1:' + fm + ' \n2:' + sm + ' \n3:' + tm +
 ' \n4:' + fourthm
 print('Your Recommendations for the Movie ' + searchMovie + ' are:\n')
 print(finalRecommendationText)
```

- inputMovieName.on\_submit(getRecommendations)  
inputMovieName
- {"model\_id":"f20b1112b2c24f9eb2be1501a9d47023","version\_major":2,"version\_minor":0}
- Your Recommendations for the Movie Star Wars (1977) are:  
  
1:Empire Strikes Back, The (1980)  
2:Return of the Jedi (1983)  
3:Raiders of the Lost Ark (1981)  
4:Austin Powers: International Man of Mystery (1997)

-