Data

Integration

Project

**Introduction**

Integrating and analysing data help companies to identify patterns, trends and business opportunities that can shape a companies strategy. This project in hand integrates and analyses a data set by using different tools and testing their capabilities. Apache spark has been one of the tools used in this project to analyse the data and create a strategic plan for the company.By leveraging customer demographics and purchasing behavior, the goal is to uncover insights that can enhance product offerings and drive sales strategies.By leveraging customer demographics and purchasing behavior, the goal is to uncover insights that can enhance product offerings and drive sales strategies.By leveraging customer demographics and purchasing behavior, the goal is to uncover insights that can enhance product offerings and drive sales strategies.

By leveraging customer info, demographics and purchasing behavior, the researcher is uncovering insights that can enhance product offerings and incease sales strategies. The goal for analysing this data is to better understand the customers behaviour and their wants. The data from the selected data set has been cleaned, analysed and integrated to money processes, so it will be easy for the company to work and understand the filter dataset.

**Objective**

The primary objective of this project is to use Apache Spark and its data processing and integration capabilities for the analysation of retail dataset. By performing comprehensive data preprocessing and integration tasks the goal is to uncover and understand valuable data and look into customer demographics and purchasing behavior.

**Important commands:**

pip install pandas numpy matplotlib seaborn scikit-learn pyspark

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

**Choosing a Dataset**

The data selected below is collected from online dataset website Kaggle and is the raw form of data ready to be cleaned and processed using Apache spark.

This dataset for this project contains essential information about customers, including CustomerID, Gender, Age, Annual Income, and Spending Score.

The attributes in the data said is essential for this project, that's why it's selected.

DATASET with all rows and columns

pd.set\_option('display.max\_rows', None)

pd.set\_option('display.max\_columns', None)

data.head(200)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | **Gender** | **Age** | **Annual Income (k$)** | **Spending Score (1-100)** | **Age Category** |
| **1** | Male | 19 | 15.0 | 39 | Young |
| **2** | Male | 21 | 15.0 | 81 | Young |
| **3** | Female | 20 | 16.0 | 6 | Young |
| **4** | Female | 23 | 16.0 | 77 | Young |
| **5** | Female | 31 | 17.0 | 40 | Old |
| **6** | Female | 22 | 17.0 | 76 | Young |
| **7** | Female | 35 | 18.0 | 6 | Old |
| **8** | Female | 23 | 18.0 | 94 | Young |
| **9** | Male | 64 | 19.0 | 3 | Old |
| **10** | Female | 30 | 19.0 | 72 | Old |
| **11** | Male | 67 | 19.0 | 14 | Old |
| **12** | Female | 35 | 19.0 | 99 | Old |
| **13** | Female | 58 | 20.0 | 15 | Old |
| **14** | Female | 24 | 20.0 | 77 | Young |
| **15** | Male | 37 | 20.0 | 13 | Old |
| **16** | Male | 22 | 20.0 | 79 | Young |
| **17** | Female | 35 | 21.0 | 35 | Old |
| **18** | Male | 20 | 21.0 | 66 | Young |
| **19** | Male | 52 | 23.0 | 29 | Old |
| **20** | Female | 35 | 23.0 | 98 | Old |
| **21** | Male | 35 | 24.0 | 35 | Old |
| **22** | Male | 25 | 24.0 | 73 | Young |
| **23** | Female | 46 | 25.0 | 5 | Old |
| **24** | Male | 31 | 25.0 | 73 | Old |
| **25** | Female | 54 | 28.0 | 14 | Old |
| **26** | Male | 29 | 28.0 | 82 | Young |
| **27** | Female | 45 | 28.0 | 32 | Old |
| **28** | Male | 35 | 28.0 | 61 | Old |
| **29** | Female | 40 | 29.0 | 31 | Old |
| **30** | Female | 23 | 29.0 | 87 | Young |
| **31** | Male | 60 | 30.0 | 4 | Old |
| **32** | Female | 21 | 30.0 | 73 | Young |
| **33** | Male | 53 | 33.0 | 4 | Old |
| **34** | Male | 18 | 33.0 | 92 | Young |
| **35** | Female | 49 | 33.0 | 14 | Old |
| **36** | Female | 21 | 33.0 | 81 | Young |
| **37** | Female | 42 | 34.0 | 17 | Old |
| **38** | Female | 30 | 34.0 | 73 | Old |
| **39** | Female | 36 | 37.0 | 26 | Old |
| **40** | Female | 20 | 37.0 | 75 | Young |
| **41** | Female | 65 | 38.0 | 35 | Old |
| **42** | Male | 24 | 38.0 | 92 | Young |
| **43** | Male | 48 | 39.0 | 36 | Old |
| **44** | Female | 31 | 39.0 | 61 | Old |
| **45** | Female | 49 | 39.0 | 28 | Old |
| **46** | Female | 24 | 39.0 | 65 | Young |
| **47** | Female | 50 | 40.0 | 55 | Old |
| **48** | Female | 27 | 40.0 | 47 | Young |
| **49** | Female | 29 | 40.0 | 42 | Young |
| **50** | Female | 31 | 40.0 | 42 | Old |
| **51** | Female | 49 | 42.0 | 52 | Old |
| **52** | Male | 33 | 42.0 | 60 | Old |
| **53** | Female | 31 | 43.0 | 54 | Old |
| **54** | Male | 59 | 43.0 | 60 | Old |
| **55** | Female | 50 | 43.0 | 45 | Old |
| **56** | Male | 47 | 43.0 | 41 | Old |
| **57** | Female | 51 | 44.0 | 50 | Old |
| **58** | Male | 69 | 44.0 | 46 | Old |
| **59** | Female | 27 | 46.0 | 51 | Young |
| **60** | Male | 53 | 46.0 | 46 | Old |
| **61** | Male | 70 | 46.0 | 56 | Old |
| **62** | Male | 19 | 46.0 | 55 | Young |
| **63** | Female | 67 | 47.0 | 52 | Old |
| **64** | Female | 54 | 47.0 | 59 | Old |
| **65** | Male | 63 | 48.0 | 51 | Old |
| **66** | Male | 18 | 48.0 | 59 | Young |
| **67** | Female | 43 | 48.0 | 50 | Old |
| **68** | Female | 68 | 48.0 | 48 | Old |
| **69** | Male | 19 | 48.0 | 59 | Young |
| **70** | Female | 32 | 48.0 | 47 | Old |
| **71** | Male | 70 | 49.0 | 55 | Old |
| **72** | Female | 47 | 49.0 | 42 | Old |
| **73** | Female | 60 | 50.0 | 49 | Old |
| **74** | Female | 60 | 50.0 | 56 | Old |
| **75** | Male | 59 | 54.0 | 47 | Old |
| **76** | Male | 26 | 54.0 | 54 | Young |
| **77** | Female | 45 | 54.0 | 53 | Old |
| **78** | Male | 40 | 54.0 | 48 | Old |
| **79** | Female | 23 | 54.0 | 52 | Young |
| **80** | Female | 49 | 54.0 | 42 | Old |
| **81** | Male | 57 | 54.0 | 51 | Old |
| **82** | Male | 38 | 54.0 | 55 | Old |
| **83** | Male | 67 | 54.0 | 41 | Old |
| **84** | Female | 46 | 54.0 | 44 | Old |
| **85** | Female | 21 | 54.0 | 57 | Young |
| **86** | Male | 48 | 54.0 | 46 | Old |
| **87** | Female | 55 | 57.0 | 58 | Old |
| **88** | Female | 22 | 57.0 | 55 | Young |
| **89** | Female | 34 | 58.0 | 60 | Old |
| **90** | Female | 50 | 58.0 | 46 | Old |
| **91** | Female | 68 | 59.0 | 55 | Old |
| **92** | Male | 18 | 59.0 | 41 | Young |
| **93** | Male | 48 | 60.0 | 49 | Old |
| **94** | Female | 40 | 60.0 | 40 | Old |
| **95** | Female | 32 | 60.0 | 42 | Old |
| **96** | Male | 24 | 60.0 | 52 | Young |
| **97** | Female | 47 | 60.0 | 47 | Old |
| **98** | Female | 27 | 60.0 | 50 | Young |
| **99** | Male | 48 | 61.0 | 42 | Old |
| **100** | Male | 20 | 61.0 | 49 | Young |
| **101** | Female | 23 | 62.0 | 41 | Young |
| **102** | Female | 49 | 62.0 | 48 | Old |
| **103** | Male | 67 | 62.0 | 59 | Old |
| **104** | Male | 26 | 62.0 | 55 | Young |
| **105** | Male | 49 | 62.0 | 56 | Old |
| **106** | Female | 21 | 62.0 | 42 | Young |
| **107** | Female | 66 | 63.0 | 50 | Old |
| **108** | Male | 54 | 63.0 | 46 | Old |
| **109** | Male | 68 | 63.0 | 43 | Old |
| **110** | Male | 66 | 63.0 | 48 | Old |
| **111** | Male | 65 | 63.0 | 52 | Old |
| **112** | Female | 19 | 63.0 | 54 | Young |
| **113** | Female | 38 | 64.0 | 42 | Old |
| **114** | Male | 19 | 64.0 | 46 | Young |
| **115** | Female | 18 | 65.0 | 48 | Young |
| **116** | Female | 19 | 65.0 | 50 | Young |
| **117** | Female | 63 | 65.0 | 43 | Old |
| **118** | Female | 49 | 65.0 | 59 | Old |
| **119** | Female | 51 | 67.0 | 43 | Old |
| **120** | Female | 50 | 67.0 | 57 | Old |
| **121** | Male | 27 | 67.0 | 56 | Young |
| **122** | Female | 38 | 67.0 | 40 | Old |
| **123** | Female | 40 | 69.0 | 58 | Old |
| **124** | Male | 39 | 69.0 | 91 | Old |
| **125** | Female | 23 | 70.0 | 29 | Young |
| **126** | Female | 31 | 70.0 | 77 | Old |
| **127** | Male | 43 | 71.0 | 35 | Old |
| **128** | Male | 40 | 71.0 | 95 | Old |
| **129** | Male | 59 | 71.0 | 11 | Old |
| **130** | Male | 38 | 71.0 | 75 | Old |
| **131** | Male | 47 | 71.0 | 9 | Old |
| **132** | Male | 39 | 71.0 | 75 | Old |
| **133** | Female | 25 | 72.0 | 34 | Young |
| **134** | Female | 31 | 72.0 | 71 | Old |
| **135** | Male | 20 | 73.0 | 5 | Young |
| **136** | Female | 29 | 73.0 | 88 | Young |
| **137** | Female | 44 | 73.0 | 7 | Old |
| **138** | Male | 32 | 73.0 | 73 | Old |
| **139** | Male | 19 | 74.0 | 10 | Young |
| **140** | Female | 35 | 74.0 | 72 | Old |
| **141** | Female | 57 | 75.0 | 5 | Old |
| **142** | Male | 32 | 75.0 | 93 | Old |
| **143** | Female | 28 | 76.0 | 40 | Young |
| **144** | Female | 32 | 76.0 | 87 | Old |
| **145** | Male | 25 | 77.0 | 12 | Young |
| **146** | Male | 28 | 77.0 | 97 | Young |
| **147** | Male | 48 | 77.0 | 36 | Old |
| **148** | Female | 32 | 77.0 | 74 | Old |
| **149** | Female | 34 | 78.0 | 22 | Old |
| **150** | Male | 34 | 78.0 | 90 | Old |
| **151** | Male | 43 | 78.0 | 17 | Old |
| **152** | Male | 39 | 78.0 | 88 | Old |
| **153** | Female | 44 | 78.0 | 20 | Old |
| **154** | Female | 38 | 78.0 | 76 | Old |
| **155** | Female | 47 | 78.0 | 16 | Old |
| **156** | Female | 27 | 78.0 | 89 | Young |
| **157** | Male | 37 | 78.0 | 1 | Old |
| **158** | Female | 30 | 78.0 | 78 | Old |
| **159** | Male | 34 | 78.0 | 1 | Old |
| **160** | Female | 30 | 78.0 | 73 | Old |
| **161** | Female | 56 | 79.0 | 35 | Old |
| **162** | Female | 29 | 79.0 | 83 | Young |
| **163** | Male | 19 | 81.0 | 5 | Young |
| **164** | Female | 31 | 81.0 | 93 | Old |
| **165** | Male | 50 | 85.0 | 26 | Old |
| **166** | Female | 36 | 85.0 | 75 | Old |
| **167** | Male | 42 | 86.0 | 20 | Old |
| **168** | Female | 33 | 86.0 | 95 | Old |
| **169** | Female | 36 | 87.0 | 27 | Old |
| **170** | Male | 32 | 87.0 | 63 | Old |
| **171** | Male | 40 | 87.0 | 13 | Old |
| **172** | Male | 28 | 87.0 | 75 | Young |
| **173** | Male | 36 | 87.0 | 10 | Old |
| **174** | Male | 36 | 87.0 | 92 | Old |
| **175** | Female | 52 | 88.0 | 13 | Old |
| **176** | Female | 30 | 88.0 | 86 | Old |
| **177** | Male | 58 | 88.0 | 15 | Old |
| **178** | Male | 27 | 88.0 | 69 | Young |
| **179** | Male | 59 | 93.0 | 14 | Old |
| **180** | Male | 35 | 93.0 | 90 | Old |
| **181** | Female | 37 | 97.0 | 32 | Old |
| **182** | Female | 32 | 97.0 | 86 | Old |
| **183** | Male | 46 | 98.0 | 15 | Old |
| **184** | Female | 29 | 98.0 | 88 | Young |
| **185** | Female | 41 | 99.0 | 39 | Old |
| **186** | Male | 30 | 99.0 | 97 | Old |
| **187** | Female | 54 | 101.0 | 24 | Old |
| **188** | Male | 28 | 101.0 | 68 | Young |
| **189** | Female | 41 | 103.0 | 17 | Old |
| **190** | Female | 36 | 103.0 | 85 | Old |
| **191** | Female | 34 | 103.0 | 23 | Old |
| **192** | Female | 32 | 103.0 | 69 | Old |
| **193** | Male | 33 | 113.0 | 8 | Old |
| **194** | Female | 38 | 113.0 | 91 | Old |
| **195** | Female | 47 | 120.0 | 16 | Old |
| **196** | Female | 35 | 120.0 | 79 | Old |
| **197** | Female | 45 | 126.0 | 28 | Old |
| **198** | Male | 32 | 126.0 | 74 | Old |
| **199** | Male | 32 | 137.0 | 18 | Old |
| **200** | Male | 30 | 137.0 | 83 | Old |

Data cleaning and catagorisation

After loading the data In Jupiter’s notebook, the CSV file is read and all the valuable content are stored. The next command given, here was to read the columns and rows of the dataset. Then we make sure that the data set doesn’t have any missing values. So all the columns or rows with missing values will be dropped then re-filter the data based on quantity and unit price. Put that just for assurance we convert invoice data to date time format using PD.to\_datetime(). This make sure that if invoicedate column is not found a prints a message. To make sure the total price data is available, we write a command that multiplies quantity and unit price into a new column call total price.

In order to make the process easier we create new datasets where the genders are in separate data tables, so there is a table for female gender and there is a table for male gender.

data = pd.read\_csv('data int.csv')

print("Original Columns:")

print(data.columns)

print("\nFirst few rows:")

print(data.head(200))

data.dropna(inplace=True)

if 'Quantity' in data.columns and 'UnitPrice' in data.columns:

data = data[(data['Quantity'] > 0) & (data['UnitPrice'] > 0)]

else:

print("Column 'Quantity' or 'UnitPrice' not found or NA values dropped all rows.")

if 'InvoiceDate' in data.columns:

data['InvoiceDate'] = pd.to\_datetime(data['InvoiceDate'])

else:

print("Column 'InvoiceDate' not found.")

if 'Quantity' in data.columns and 'UnitPrice' in data.columns:

data['TotalPrice'] = data['Quantity'] \* data['UnitPrice']

else:

print("Column 'Quantity' or 'UnitPrice' not found.")

if 'Gender' in data.columns:

female\_data = data[data['Gender'] == 'Female']

male\_data = data[data['Gender'] == 'Male']

else:

print("Column 'Gender' not found.")

print("\nFemale Data:")

print(female\_data.head(200))

print("\nMale Data:")

print(male\_data.head(200))

Original Columns:

Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',

'Spending Score (1-100)'],

dtype='object')

First few rows:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15 39

1 2 Male 21 15 81

2 3 Female 20 16 6

3 4 Female 23 16 77

4 5 Female 31 17 40

.. ... ... ... ... ...

195 196 Female 35 120 79

196 197 Female 45 126 28

197 198 Male 32 126 74

198 199 Male 32 137 18

199 200 Male 30 137 83

[200 rows x 5 columns]

Column 'Quantity' or 'UnitPrice' not found or NA values dropped all rows.

Column 'InvoiceDate' not found.

Column 'Quantity' or 'UnitPrice' not found.

Female Data:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

2 3 Female 20 16 6

3 4 Female 23 16 77

4 5 Female 31 17 40

5 6 Female 22 17 76

6 7 Female 35 18 6

.. ... ... ... ... ...

191 192 Female 32 103 69

193 194 Female 38 113 91

194 195 Female 47 120 16

195 196 Female 35 120 79

196 197 Female 45 126 28

[112 rows x 5 columns]

Male Data:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15 39

1 2 Male 21 15 81

8 9 Male 64 19 3

10 11 Male 67 19 14

14 15 Male 37 20 13

.. ... ... ... ... ...

187 188 Male 28 101 68

192 193 Male 33 113 8

197 198 Male 32 126 74

198 199 Male 32 137 18

199 200 Male 30 137 83

[88 rows x 5 columns]

Data Analysis and Visualization of Customer Spending Scores by Age

The code written by essaer here uses the pandas and matplotlib libraries to create and plot sample customer data. These sample customer details can be used to understand the financial aspects of the data set. The writer first creates two dataframes with sample data for female and male customers, including their age and spending score. The code then groups this data by age and calculates the average spending score for each age group. Then the writer plots these averages on a graph, with separate lines for female and male customers, and includes lines indicating the overall mean spending score for each gender. The ring graph gives a data visualisation off the expanding trend record for the female and male customers.

import pandas as pd

import matplotlib.pyplot as plt

female\_data = pd.DataFrame({

'CustomerID': [3, 4, 5, 6, 7],

'Gender': ['Female', 'Female', 'Female', 'Female', 'Female'],

'Age': [20, 23, 31, 22, 35],

'Annual Income (k$)': [16, 16, 17, 17, 18],

'Spending Score (1-100)': [6, 77, 40, 76, 6]

})

male\_data = pd.DataFrame({

'CustomerID': [1, 2, 9, 11, 15],

'Gender': ['Male', 'Male', 'Male', 'Male', 'Male'],

'Age': [19, 21, 64, 67, 37],

'Annual Income (k$)': [15, 15, 19, 19, 20],

'Spending Score (1-100)': [39, 81, 3, 14, 13]

})

female\_age\_groups = female\_data.groupby('Age')['Spending Score (1-100)'].mean()

male\_age\_groups = male\_data.groupby('Age')['Spending Score (1-100)'].mean()

plt.figure(figsize=(10, 6))

plt.plot(female\_age\_groups.index, female\_age\_groups.values, marker='o', linestyle='-', color='pink', label='Female')

plt.axhline(y=female\_age\_groups.mean(), color='red', linestyle='--', label=f'Female Mean Score: {female\_age\_groups.mean():.2f}')

plt.plot(male\_age\_groups.index, male\_age\_groups.values, marker='o', linestyle='-', color='blue', label='Male')

plt.axhline(y=male\_age\_groups.mean(), color='orange', linestyle='--', label=f'Male Mean Score: {male\_age\_groups.mean():.2f}')

plt.title('Average Spending Score Trend by Age')

plt.xlabel('Age')

plt.ylabel('Average Spending Score')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**A graph with a line and a line

Description automatically generated with medium confidence**

Comparing Spending Scores with Trend Lines

The writer here inputs a code to creates bar charts to visualize the average spending scores by age for male and female customers using sample data. He calculates the mean spending scores for different age groups, plots these averages as bars, and adds horizontal lines to indicate the overall mean spending scores for each gender. Additionally, it includes trend lines to show how spending scores change with age. The resulting plots, displayed side by side, provide a clear comparison of spending behaviors between male and female customers across different ages.

from scipy.stats import linregress

female\_data = pd.DataFrame({

'CustomerID': [3, 4, 5, 6, 7],

'Gender': ['Female', 'Female', 'Female', 'Female', 'Female'],

'Age': [20, 23, 31, 22, 35],

'Annual Income (k$)': [16, 16, 17, 17, 18],

'Spending Score (1-100)': [6, 77, 40, 76, 6]

})

male\_data = pd.DataFrame({

'CustomerID': [1, 2, 9, 11, 15],

'Gender': ['Male', 'Male', 'Male', 'Male', 'Male'],

'Age': [19, 21, 64, 67, 37],

'Annual Income (k$)': [15, 15, 19, 19, 20],

'Spending Score (1-100)': [39, 81, 3, 14, 13]

})

female\_age\_groups = female\_data.groupby('Age')['Spending Score (1-100)'].mean()

male\_age\_groups = male\_data.groupby('Age')['Spending Score (1-100)'].mean()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

female\_age\_groups.plot(kind='bar', color='pink', label='Average Spending Score')

plt.axhline(y=female\_age\_groups.mean(), color='red', linestyle='--', label=f'Female Mean Score: {female\_age\_groups.mean():.2f}')

x\_female = female\_age\_groups.index

y\_female = female\_age\_groups.values

slope\_female, intercept\_female, \_, \_, \_ = linregress(x\_female, y\_female)

plt.plot(x\_female, slope\_female \* x\_female + intercept\_female, color='purple', linestyle='-', label='Trend Line (Female)')

plt.title('Average Spending Score by Age (Female)')

plt.xlabel('Age')

plt.ylabel('Average Spending Score')

plt.legend()

plt.subplot(1, 2, 2)

male\_age\_groups.plot(kind='bar', color='blue', label='Average Spending Score')

plt.axhline(y=male\_age\_groups.mean(), color='orange', linestyle='--', label=f'Male Mean Score: {male\_age\_groups.mean():.2f}')

x\_male = male\_age\_groups.index

y\_male = male\_age\_groups.values

slope\_male, intercept\_male, \_, \_, \_ = linregress(x\_male, y\_male)

plt.plot(x\_male, slope\_male \* x\_male + intercept\_male, color='green', linestyle='-', label='Trend Line (Male)')

plt.title('Average Spending Score by Age (Male)')

plt.xlabel('Age')

plt.ylabel('Average Spending Score')

plt.legend()

plt.tight\_layout()

plt.show()

**A comparison of a graph

Description automatically generated**

**Apache Spark Implementation**

**First, the writer will download Apache spark and all its components into Jupiter notebook in order to do the next step of the assignment, which is applying Apache spark on the dateset in hand to find the financial trend.**

**pip install pyspark**

This code below initializes a Spark session using PySpark, which is important for running Spark and its applications. The session below is named "Customer Segmentation," and it allows you to work with Spark's data processing capabilities, enabling tasks like data analysis, manipulation, and machine learning on large datasets.

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Customer Segmentation") \

.getOrCreate()

Before the next step once more, we open the dataset and make sure that is correct. Also, to check the columns and their names.

print("Original Columns:")

print(data.columns)

print("\nFirst few rows:")

print(data.head(200))

Original Columns:

Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',

'Spending Score (1-100)'],

dtype='object')

First few rows:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15 39

1 2 Male 21 15 81

2 3 Female 20 16 6

3 4 Female 23 16 77

4 5 Female 31 17 40

.. ... ... ... ... ...

195 196 Female 35 120 79

196 197 Female 45 126 28

197 198 Male 32 126 74

198 199 Male 32 137 18

199 200 Male 30 137 83

[200 rows x 5 columns]

**Remove missing values**

data.dropna(inplace=True)

CHECK:

The writer uses the code below to checks if the columns Quantity and UnitPrice exist in a dataset after removing any rows with missing values. So it does basically two functions. Fristly If both columns are found, it filters the dataset to keep only rows where Quantity and UnitPrice are positive. If either column isn't found or all rows are removed due to missing values, it prints a message accordingly.

**if 'Quantity' in data.columns and 'UnitPrice' in data.columns:**

**data = data[(data['Quantity'] > 0) & (data['UnitPrice'] > 0)]**

**else:**

**print("Column 'Quantity' or 'UnitPrice' not found or NA values dropped all rows.")**

**Result:** Column 'Quantity' or 'UnitPrice' not found or NA values dropped all rows.

**Again, re-check the missing values via code: the result below shows there are no missing values.**

**print("\nMissing values in each column:")**

**print(data.isna().sum())**

Missing values in each column:

CustomerID 0

Gender 0

Age 0

Annual Income (k$) 0

Spending Score (1-100) 0

dtype: int64

**Data Transformation and Cleaning Process**

The writer wrote a code that transforms the data and make a clean dataset.

First, he converted the Gender column to a categorical variable which helps him analyze and plot the data more effectively.

Next, he changed the CustomerID column to be treated as a string. This way, he can use it as an identifier without worrying about any numerical operations that well defect and destroy the data set.

The writer also made sure the numerical columns like Age, Annual Income (k$) and spending score (1-100) have the correct types. Now after that age is an integer annual income is a flaot and spending score is also an integer This ensures our calculations and analyses on these columns are accurate.

After all that, the writer printed the first 200 rows of cleaned dataset to inspect the changes and ensure our data quality.

Finally, he saved this cleaned dataset to a CSV file named 'cleaned\_data.csv', which will be handy for further analysis or machine learning tasks. These steps together ensure the dataset is consistent and ready for deeper exploration.

data['Gender'] = data['Gender'].astype('category')

data['CustomerID'] = data['CustomerID'].astype('str')

data['Age'] = data['Age'].astype('int')

data['Annual Income (k$)'] = data['Annual Income (k$)'].astype('float')

data['Spending Score (1-100)'] = data['Spending Score (1-100)'].astype('int')

print("\nCleaned Data:")

print(data.head(200))

data.to\_csv('cleaned\_data.csv', index=False)

Columns in the dataset:

Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',

'Spending Score (1-100)'],

dtype='object')

First few rows of the dataset:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15 39

1 2 Male 21 15 81

2 3 Female 20 16 6

3 4 Female 23 16 77

4 5 Female 31 17 40

.. ... ... ... ... ...

195 196 Female 35 120 79

196 197 Female 45 126 28

197 198 Male 32 126 74

198 199 Male 32 137 18

199 200 Male 30 137 83

[200 rows x 5 columns]

Cleaned Data:

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15.0 39

1 2 Male 21 15.0 81

2 3 Female 20 16.0 6

3 4 Female 23 16.0 77

4 5 Female 31 17.0 40

.. ... ... ... ... ...

195 196 Female 35 120.0 79

196 197 Female 45 126.0 28

197 198 Male 32 126.0 74

198 199 Male 32 137.0 18

199 200 Male 30 137.0 83

[200 rows x 5 columns]

Now we import OS and check if the data is saved.

import os

data\_path = "cleaned\_data.csv"

if os.path.exists(data\_path):

print("File exists")

else:

print("File does not exist")

Result: File exists

df.printSchema() prints out the schema of the DataFrame df, showing the types of data in each column and any nested structures it may contain.

df.printSchema()

root

|-- CustomerID: integer (nullable = true)

|-- Gender: string (nullable = true)

|-- Age: integer (nullable = true)

|-- Annual Income (k$): double (nullable = true)

|-- Spending Score (1-100): integer (nullable = true)

In here the writer uses df.describe().show() to provides a summary of statistics for the numerical columns in the DataFrame df, showing counts, averages, standard deviations, and the range of values present.

+-------+------------------+------+-----------------+------------------+----------------------+

|summary| CustomerID|Gender| Age|Annual Income (k$)|Spending Score (1-100)|

+-------+------------------+------+-----------------+------------------+----------------------+

| count| 200| 200| 200| 200| 200|

| mean| 100.5| NULL| 38.85| 60.56| 50.2|

| stddev|57.879184513951124| NULL|13.96900733155888| 26.26472116527124| 25.823521668370173|

| min| 1|Female| 18| 15.0| 1|

| max| 200| Male| 70| 137.0| 99|

+-------+------------------+------+-----------------+------------------+----------------------+

**Calculating Null Counts in Spark DataFrame**

This code snippet calculates the number of null values in each column of a Spark DataFrame (df). The writer uses this  function to import col and sum, which are then applied to count null values for each column (c) in df. The results are converted into a Pandas DataFrame (null\_counts\_pd) for clearer visualization by transposing the data. Finally, the null counts are printed to assess data quality and prepare for further analysis.

from pyspark.sql.functions import col, sum

null\_counts = df.select([sum(col(c).isNull().cast("int")).alias(c) for c in df.columns])

null\_counts\_pd = null\_counts.toPandas().transpose()

print("Null Counts:")

print(null\_counts\_pd)

Null Counts:

0

CustomerID 0

Gender 0

Age 0

Annual Income (k$) 0

Spending Score (1-100) 0

**Filtering DataFrame for Age Above 30**

This code filters the DataFrame df to retain rows where the 'Age' column values are greater than 30, displaying the filtered results using the code below.

filtered\_data = df.filter(df['Age'] > 30)

filtered\_data.show()

+----------+------+---+------------------+----------------------+

|CustomerID|Gender|Age|Annual Income (k$)|Spending Score (1-100)|

+----------+------+---+------------------+----------------------+

| 5|Female| 31| 17.0| 40|

| 7|Female| 35| 18.0| 6|

| 9| Male| 64| 19.0| 3|

| 11| Male| 67| 19.0| 14|

| 12|Female| 35| 19.0| 99|

| 13|Female| 58| 20.0| 15|

| 15| Male| 37| 20.0| 13|

| 17|Female| 35| 21.0| 35|

| 19| Male| 52| 23.0| 29|

| 20|Female| 35| 23.0| 98|

| 21| Male| 35| 24.0| 35|

| 23|Female| 46| 25.0| 5|

| 24| Male| 31| 25.0| 73|

| 25|Female| 54| 28.0| 14|

| 27|Female| 45| 28.0| 32|

| 28| Male| 35| 28.0| 61|

| 29|Female| 40| 29.0| 31|

| 31| Male| 60| 30.0| 4|

| 33| Male| 53| 33.0| 4|

| 35|Female| 49| 33.0| 14|

+----------+------+---+------------------+----------------------+

only showing top 20 rows

**Grouping and Aggregating Data by Gender**

The code groups the DataFrame df by Gender, calculating the mean of Annual Income and the maximum of Spending Score (1-100) for each group. Results are shown using blow by the writer.

grouped\_data = df.groupBy('Gender').agg({'Annual Income (k$)': 'mean', 'Spending Score (1-100)': 'max'})

grouped\_data.show()

A close-up of a number

Description automatically generated

Then we save the updated DataFrame to a new CSV file.

df.write.csv("processed\_data.csv", header=True)

Then we check if the directory has been created.

if os.path.exists(output\_dir):

print(f"Directory '{output\_dir}' created and CSV files are saved.")

files = os.listdir(output\_dir)

print("Files in the directory:")

for file in files:

print(file)

else:

print(f"Failed to create directory '{output\_dir}'.")

A black text on a white background

Description automatically generated

**Show the contents of the saved DataFrame**

saved\_df.show()

+----------+------+---+------------------+----------------------+------------+

|CustomerID|Gender|Age|Annual Income (k$)|Spending Score (1-100)|Age Category|

+----------+------+---+------------------+----------------------+------------+

| 1| Male| 19| 15.0| 39| Young|

| 2| Male| 21| 15.0| 81| Young|

| 3|Female| 20| 16.0| 6| Young|

| 4|Female| 23| 16.0| 77| Young|

| 5|Female| 31| 17.0| 40| Old|

| 6|Female| 22| 17.0| 76| Young|

| 7|Female| 35| 18.0| 6| Old|

| 8|Female| 23| 18.0| 94| Young|

| 9| Male| 64| 19.0| 3| Old|

| 10|Female| 30| 19.0| 72| Old|

| 11| Male| 67| 19.0| 14| Old|

| 12|Female| 35| 19.0| 99| Old|

| 13|Female| 58| 20.0| 15| Old|

| 14|Female| 24| 20.0| 77| Young|

| 15| Male| 37| 20.0| 13| Old|

| 16| Male| 22| 20.0| 79| Young|

| 17|Female| 35| 21.0| 35| Old|

| 18| Male| 20| 21.0| 66| Young|

| 19| Male| 52| 23.0| 29| Old|

| 20|Female| 35| 23.0| 98| Old|

+----------+------+---+------------------+----------------------+------------+

only showing top 20 rows

**Clustering Customer Segments**

The writer uses machine learning to group customers into clusters based on their data like annual income and spending score. First, he selects these two features and combines them into a single feature vector. Then, a KMeans clustering model is trained with 5 clusters to identify patterns in customer behavior. After making predictions on which cluster each customer belongs to, the results are shown alongside their demographic information. Finally, the clusters are visualized it to understand how customers are segmented based on income and spending habits.

df.describe().show()

gender\_count = df.groupBy("Gender").count()

gender\_count.show()

average\_spending\_score = df.groupBy("Gender").avg("Spending Score (1-100)")

average\_spending\_score.show()

spending\_scores = df.select("Spending Score (1-100)").collect()

spending\_scores = [row["Spending Score (1-100)"] for row in spending\_scores]

plt.hist(spending\_scores, bins=20, edgecolor='k', alpha=0.7)

plt.xlabel('Spending Score (1-100)')

plt.ylabel('Frequency')

plt.title('Distribution of Spending Scores')

plt.show()

average\_income\_by\_age\_category = df.groupBy("Age Category").avg("Annual Income (k$)")

average\_income\_by\_age\_category.show()

A screenshot of a computer

Description automatically generated

A graph of a number of scores

Description automatically generated with medium confidence

**Customer Segmentation with KMeans Clustering**

Now using the next code below the writer segments the customers into groups.It starts by combining the two features into a vector format required for machine learning. Then, a KMeans clustering model with 5 clusters is trained on the data to identify distinct customer segments based on their spending behavior. After making predictions about which cluster each customer belongs to, the results are displayed .clusters are visualized in a mixed plot to visually represent how customers are grouped based on their income and spending habits.

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.clustering import KMeans

assembler = VectorAssembler(

inputCols=["Annual Income (k$)", "Spending Score (1-100)"],

outputCol="features")

data\_with\_features = assembler.transform(df)

kmeans = KMeans(k=5, seed=1)

model = kmeans.fit(data\_with\_features)

predictions = model.transform(data\_with\_features)

predictions.select("CustomerID", "Gender", "Age", "Annual Income (k$)", "Spending Score (1-100)", "prediction").show()

import pandas as pd

predictions\_pd = predictions.select("Annual Income (k$)", "Spending Score (1-100)", "prediction").toPandas()

plt.scatter(predictions\_pd["Annual Income (k$)"], predictions\_pd["Spending Score (1-100)"], c=predictions\_pd["prediction"], cmap='viridis')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Customer Segments')

plt.show()

A table of numbers and letters

Description automatically generated with medium confidence

A chart of a customer segment

Description automatically generated

**Calculating and Visualizing Correlations**

The code below by the writer calculates how related the variables age, annul income and spending score are in the dataset below. It turns these columns into a Pandas DataFrame to make a correlation matrix using codes. The results show as a heatmap using Seaborn, with labels to see how strong and what direction the relationships are between pairs of variables. This graph, titled 'Correlation Matrix', helps see how different parts of customer data are connected.

corr\_matrix = df.select("Age", "Annual Income (k$)", "Spending Score (1-100)").toPandas().corr()

import seaborn as sns

sns.heatmap(corr\_matrix, annot=True, cmap="coolwarm", linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()

A red and blue squares with numbers

Description automatically generated

**Visualizing Data Using Pandas and Matplotlib**

For the next step his code converts a Spark DataFrame (df) into a Pandas DataFrame (df\_pd) for easier visualization using Python. First, he plots the distribution of annual Income using histograms with a kernel density estimate to show its frequency distribution. Second, he creates a scatter plot to explore the relationship between 'Age' and 'Spending Score', with points differentiated by gender. These visualizations help understand the income distribution among customers and explore how age correlates with spending habits across genders.(Their was an error in this code below for one part which the writer could not figure out why):

import pandas as pd

df\_pd = df.toPandas()

plt.figure(figsize=(10, 6))

sns.histplot(df\_pd["Annual Income (k$)"], kde=True)

plt.title('Distribution of Annual Income')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(10, 6))

sns.scatterplot(data=df\_pd, x="Age", y="Spending Score (1-100)", hue="Gender")

plt.title('Age vs Spending Score by Gender')

plt.xlabel('Age')

plt.ylabel('Spending Score (1-100)')

plt.show()

**A graph with a line going up

Description automatically generated**

**A graph of age vs spending score

Description automatically generated**

**Exploring Age and Spending Patterns by Gender**

The final code here creates a scatter plot using Matplotlib and Seaborn to visualize how age influences spending behavior across genders. Each point on the plot represents a customer, with colors distinguishing between genders like blue for males and orange for females. The code also adds trend lines for each gender, showing the general path of the relationship between age and spending score without scatter points. The data scientist here use of the visualisation to show how different age and other aspects change spending pattern. Also he helps the company to figure out which customers from which gender and which age they have to target. This understanding of the market supports targeted marketing efforts and customer segmentation, potentially leading to more effective marketing strategies and higher customer satisfaction and happens.

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(10, 6))

sns.scatterplot(data=df\_pd, x="Age", y="Spending Score (1-100)", hue="Gender", alpha=0.6)

sns.regplot(data=df\_pd[df\_pd["Gender"] == "Male"], x="Age", y="Spending Score (1-100)", scatter=False, label='Male', color='blue')

sns.regplot(data=df\_pd[df\_pd["Gender"] == "Female"], x="Age", y="Spending Score (1-100)", scatter=False, label='Female', color='orange')

plt.title('Age vs Spending Score by Gender')

plt.xlabel('Age')

plt.ylabel('Spending Score (1-100)')

plt.legend(title='Gender')

plt.show()

**A graph showing age vs spending score

Description automatically generated**

**Conclusion**

This project has been written from the viewpoint of a data scientist for a business company. That scientist has taken customer dataset, and via various tools has categorised cleaned and analyze them in order to create a business plan for the company.In conclusion, this project has demonstrated the effectiveness of tools like Apache Spark in handling intricate data integration tasks. Through deriving insights from the integrated dataset, opportunities for improving product offerings and refining marketing strategies were identified. Apache Spark's capabilities enable data-driven decision-making, enhancing customer satisfaction and driving business growth effectively. The results afterwards has given trends and infos about the market, which can help the company.

DATSET SOURCE: https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python/data



Assessment Submission Form

|  |  |
| --- | --- |
| **Student Number**  (If this is group work, please  include the student numbers of all group participants) | GH1024093 |
| **Assessment Title** | Data Integration Project |
| **Module Code** | B142 |
| **Module Title** | Data Integration |
| **Module Tutor** | Mahmoudreza Babaei |
| **Date Submitted** | 03/07/2024 |

**Declaration of Authorship**

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

I fully understand that the unacknowledged inclusion of another person’s writings or ideas or works in this work may be considered plagiarism and that, should a formal investigation process confirms the allegation, I would be subject to the penalties associated with plagiarism, as per GISMA Business School, University of Applied Sciences’ regulations for academic misconduct.

Signed………………Ali Jawed Delawari……………. Date ………03/07/2024………………