Higher Nationals

Pearson | BTEC

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

ASSIGNMENT FINAL REPORT

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: HOANG ANH QUY		**Student ID: BS00311**
Class: DA06201		**Assessor name: NGUYEN VAN QUANG**
Issue date: **August 12th, 2024**	**Submission date:** **August 12th, 2024**	**Submitted on: CMS**
	Re-submission date:	**Re-submitted on:**
Programme: BTEC Level 5 HND Diploma in Computing		
Unit number, title, and code: Unit 25: Machine Learning (H/618/7438)		
Assignment number and title: Final report		

Group work information:

Student name: Pham Anh Tuan	Student ID: BS00304
Student name: Hoang Anh Quy	Student ID: BS00311
Student name: Nguyen Hoang Thanh Nam	Student ID: BS00499
Assignment number and title: ASM Part 1 and Part 2	

Grading Grid

Higher Nationals

Pearson | BTEC

P1	P2	P3	P4	P5	P6	P7	P8	M1	M2	M3	M4	D1	D2

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I declare that the work submitted for assessment has been carried out without assistance other than that which is acceptable according to the rules of the specification. I certify I have clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I understand that making a false declaration is a form of malpractice.

Student signature: Hoang Anh Quy Date: August 12th, 2024

Higher Nationals

Pearson | BTEC

Higher Nationals – Summative Assignment Feedback Form

Student Name/ID	Hoang Anh Quy		
Unit Title	**Unit 25: Machine Learning (H/618/7438)**		
Assignment	**Final Report**	**Assessor**	NGUYEN VAN QUANG
Submission Date	August 12th, 2024	**Date Received 1st submission**	August 12th, 2024
Re-submission Date		**Date Received 2nd submission**	

Assessor Feedback:

*Please note that constructive and useful feedback should allow students to understand:

a) **Strengths of performance**
b) **Limitations of performance**
c) **Any improvements needed in future assessments**

Feedback should be against the learning outcomes and assessment criteria to help students understand how these inform the process of judging the overall grade.

Feedback should give full guidance to the students on how they have met the learning outcomes and assessment criteria.

I certify that to the best of my knowledge the evidence submitted for this assignment is the student's own. The student has clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I have not solely used AI to grade the student's work.

Grade:	**Assessor Signature:**	**Date:**

Resubmission Feedback:

*Please note resubmission feedback is focusssed only on the resubmitted work

Grade:	**Assessor Signature:**	**Date:**

Internal Verifier's Comments:

Signature & Date:

* Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Observation record – BTEC Higher Nationals

Student name:	**Hoang Anh Quy**
Qualification:	**BTEC Level 5 HND Diploma in Computing**
Unit number & title:	**Unit 25: Machine Learning (H/618/7438)**

Description of activity undertaken

P3 Investigate a range of machine learning algorithms and how these algorithms solve learning problems. (50%)

P4 Demonstrate the efficiency of these algorithms by implementing them using an appropriate programming language or a machine learning tool. (50%)

M2 Analyse these algorithms using an appropriate example to determine their power. (50%)

D1 Critically evaluate why machine learning is essential to the design of intelligent machines.

PART 2 (33.33%)

P5 Prepare training and test data sets in order to implement a machine learning solution for an appropriate learning problem. (100%)

P6 Implement a machine learning solution with a suitable machine learning algorithm and demonstrate the outcome. (100%)

M4 Evaluate the effectiveness of the learning algorithm used in the application. (100%)

D2 Critically evaluate the implemented learning solution and its effectiveness in meeting end user requirements (33.33%)

Assessment & grading criteria

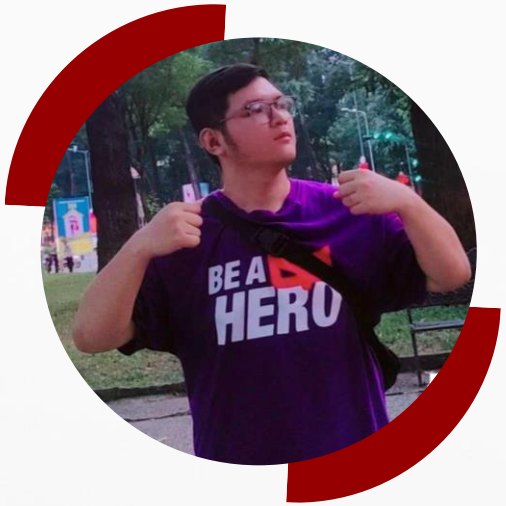
How the activity meets the requirements of the criteria			
Student signature:	**Hoang Anh Quy**	**Date:**	**August 12th, 2024**
Assessor signature:		**Date:**	
Assessor name:			

Group 1

Machine Learning

Meet Group 1 members

Nguyen Hoang Thanh Nam



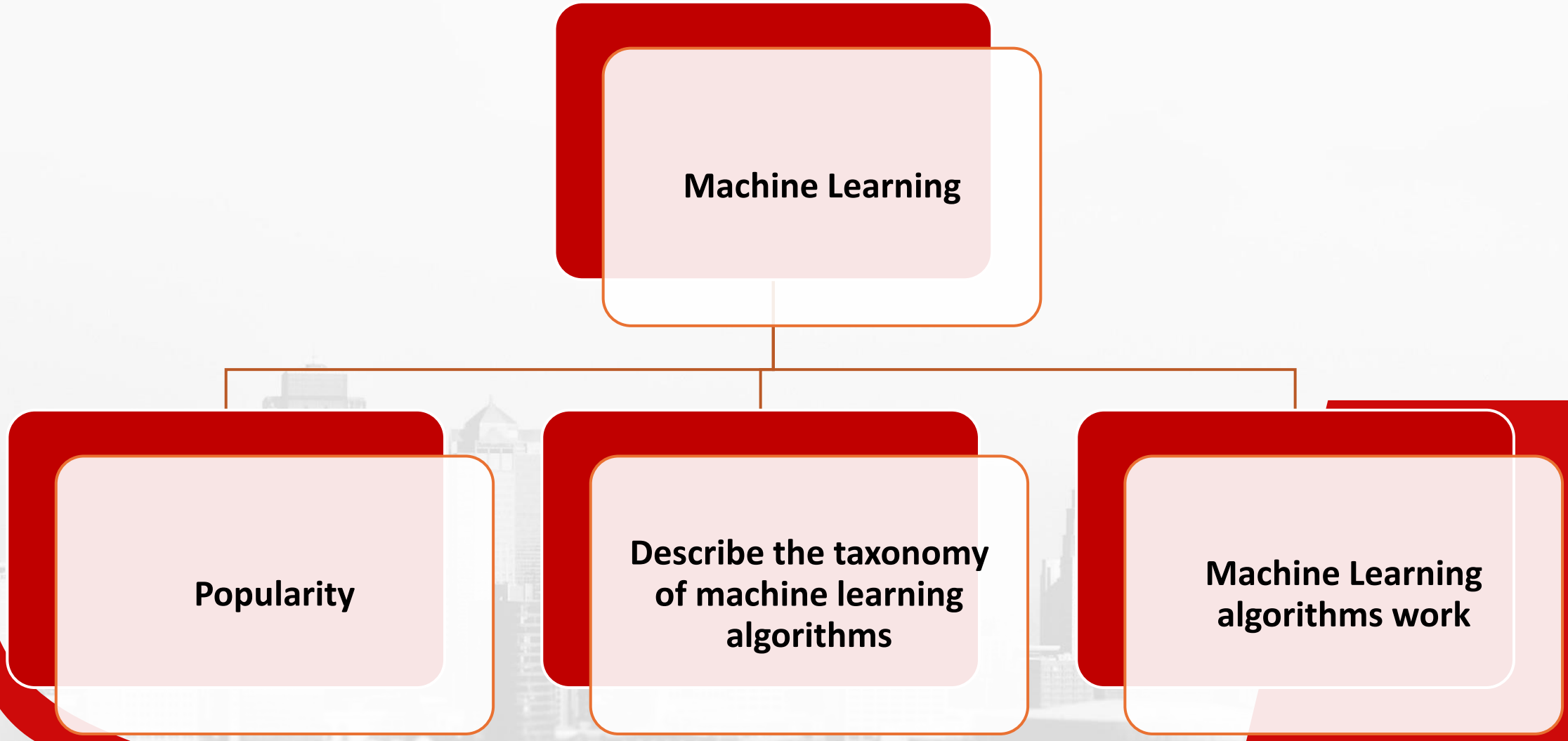
Pham Anh Tuan

Hoang Anh Quy

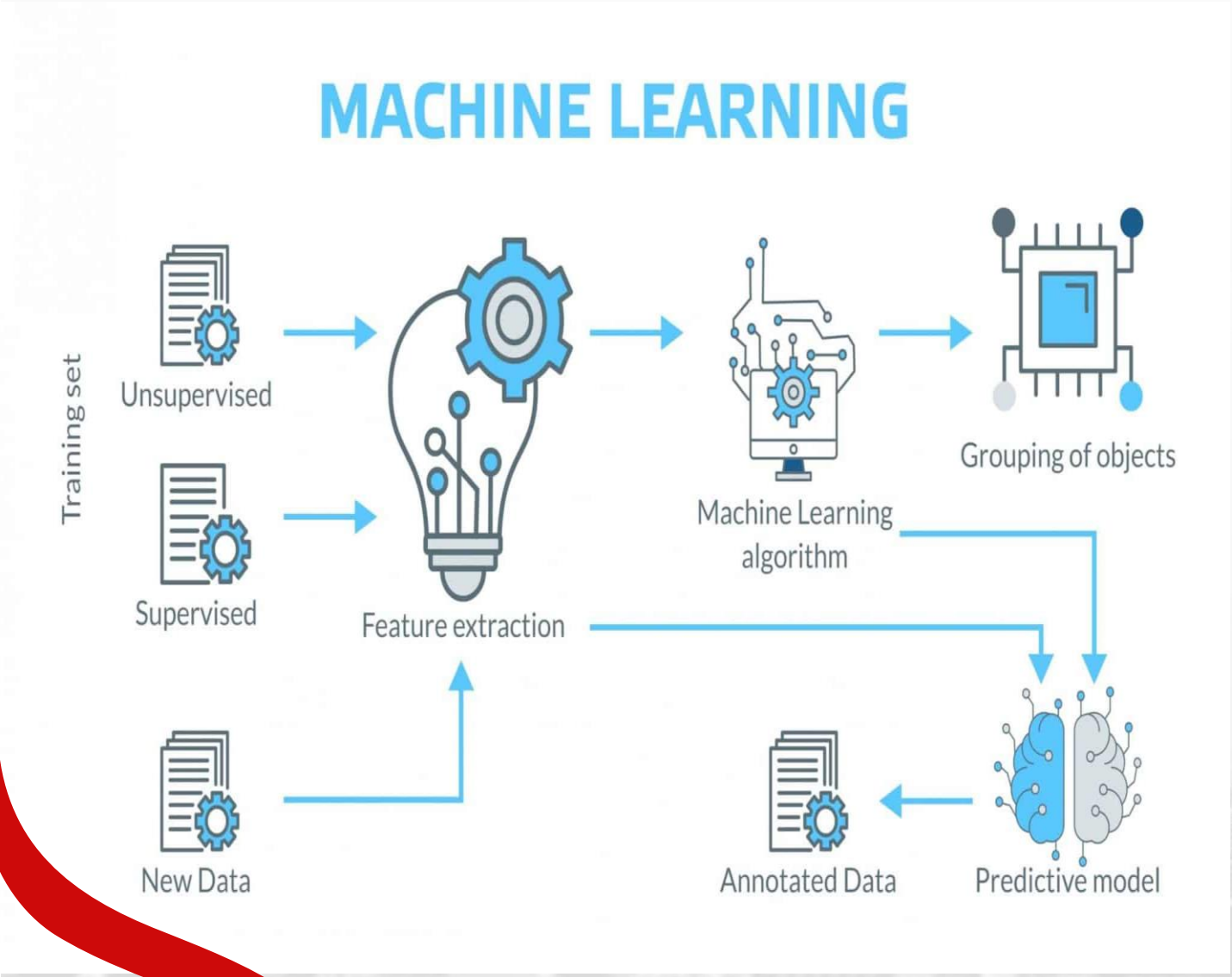
Table OF Content

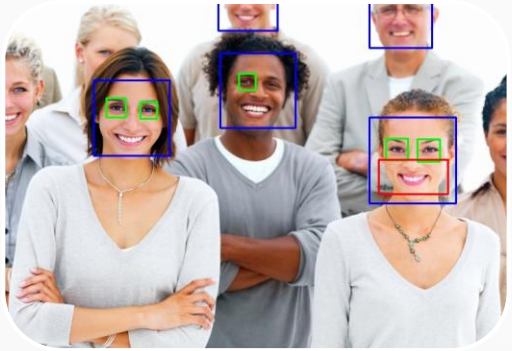
1. Analyse the types of learning problems and demonstrate the taxonomy of machine learning algorithms
2. Evaluate the category of machine learning algorithms on this business
3. Investigate a range of machine learning algorithms and how these algorithms solve learning problems
4. Demonstrate the efficiency of these algorithms by implementing them using an appropriate programming language or machine learning tool

Analyse the types of learning problems and demonstrate the taxonomy of machine learning algorithms



Analyse the types of learning problems and demonstrate the taxonomy of machine learning algorithms





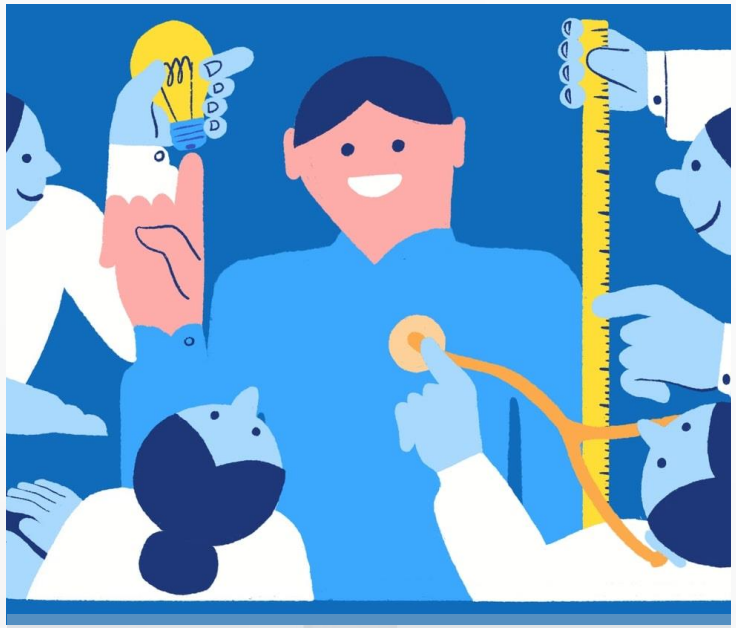
Face recognition

Film recommendations from Netflix

Spam filtering in email

Stock predictions

Analyse the types of learning problems and demonstrate the taxonomy of machine learning algorithms



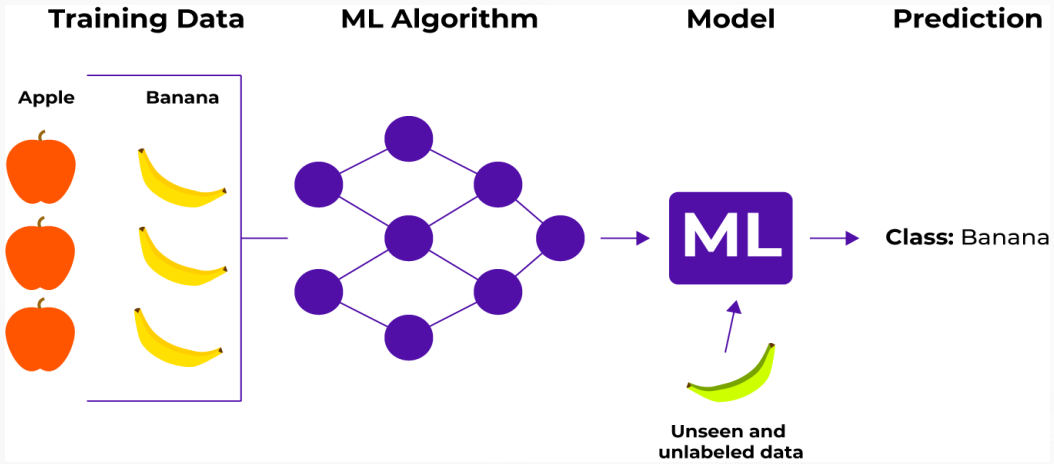
Customer Data Analysis

Optimize Business Processes

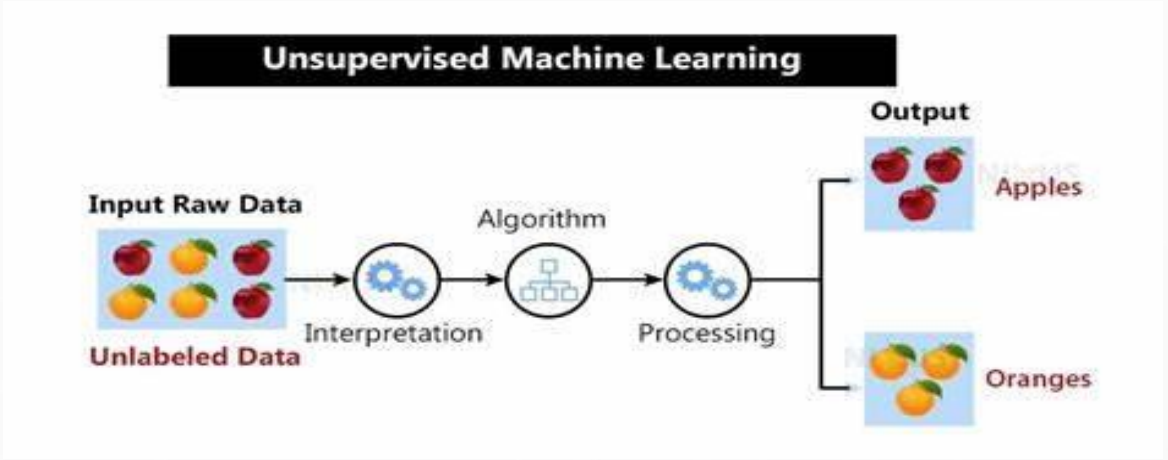
Personalize customer experience

- Increase Operational Efficiency
- Smart Decision Making
- Improve Customer Experience
- Fraud Detection and Security
- Predicting Market Trends

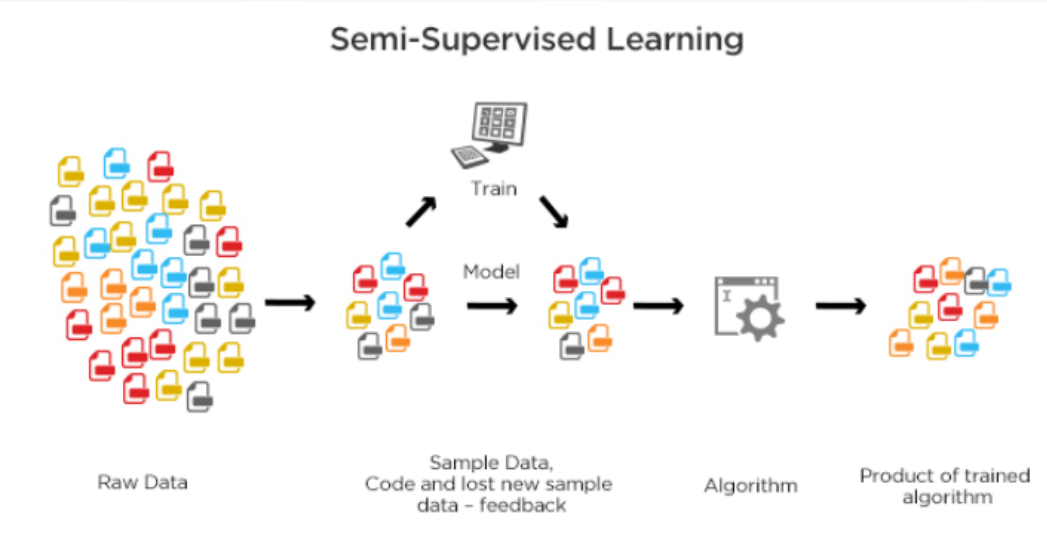
Analyse the types of learning problems and demonstrate the taxonomy of machine learning algorithms



Supervised Learning



Unsupervised Learning



Semi-supervised Learning

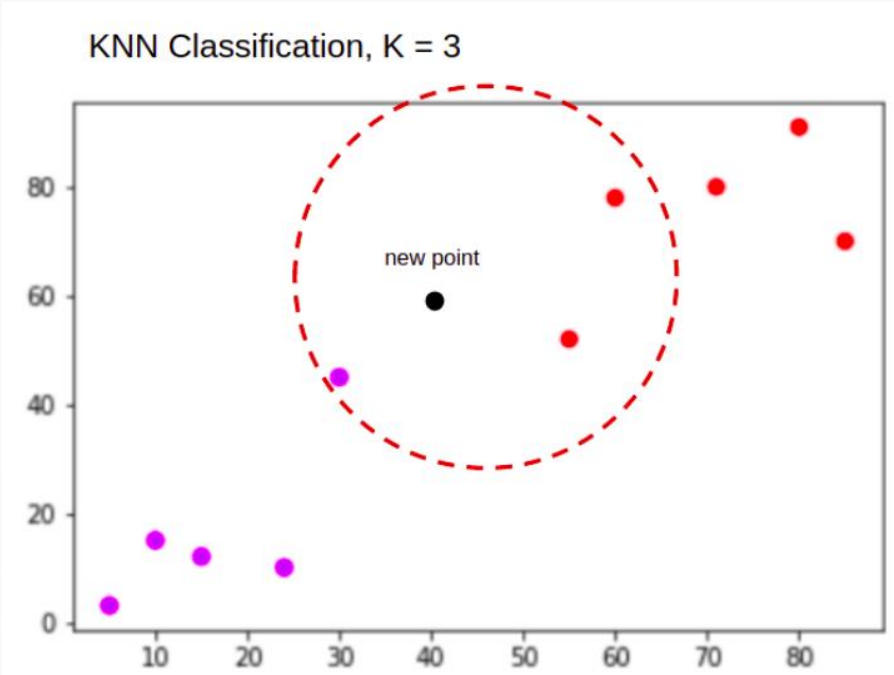
Reinforcement Learning

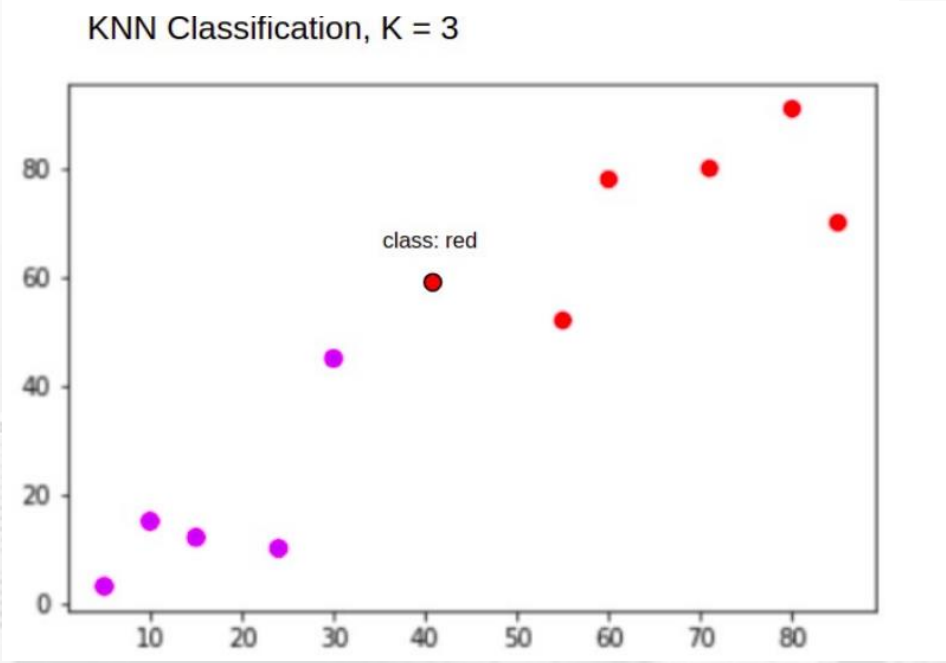
Algorithms & Problem-Solving

1. k-mean Clustering
2. KNN

KNN

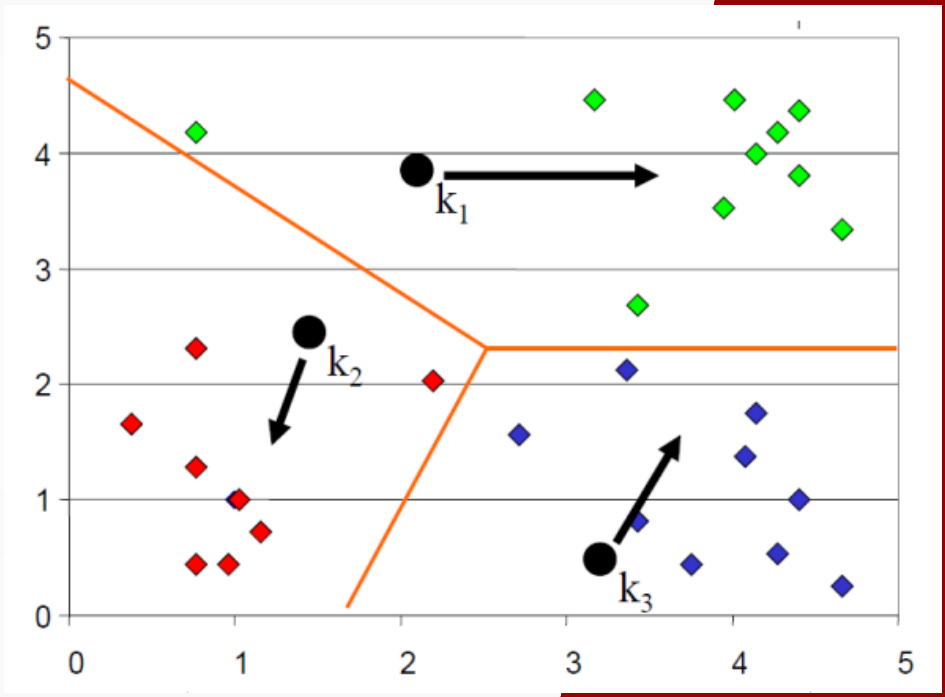
1. What is KNN
2. Applicable
3. How it works
4. Advantages & Disadvantages

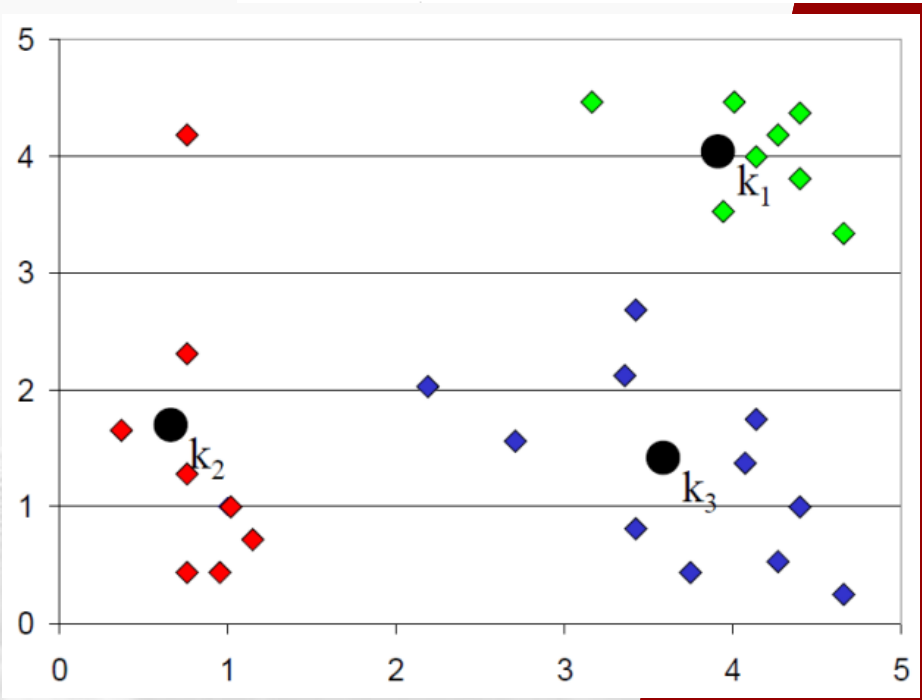




k-mean Clustering

1. What is k-mean Clustering
2. Applicable
3. How it works
4. Advantages & Disadvantages





Evaluate the category of machine learning algorithms on this business task

Machine learning algorithms are suitable for customer segmentation.

To segment customers

Choosing the K-means clustering algorithm

- Simple and effective and easy to understand.
- Predetermine the number of clusters (k)
- Suitable for quantitative features.

Create model clustering.

Data preparation and declare the necessary libraries

Select k centroids randomly

- Select the number of clusters.
- Create a DataFrame.
- Generate random centroids

Calculation error.

Assign clusters to each data point.

Update centroids and run K-Means algorithm

K-Means Algorithm Implementation and save labeled data

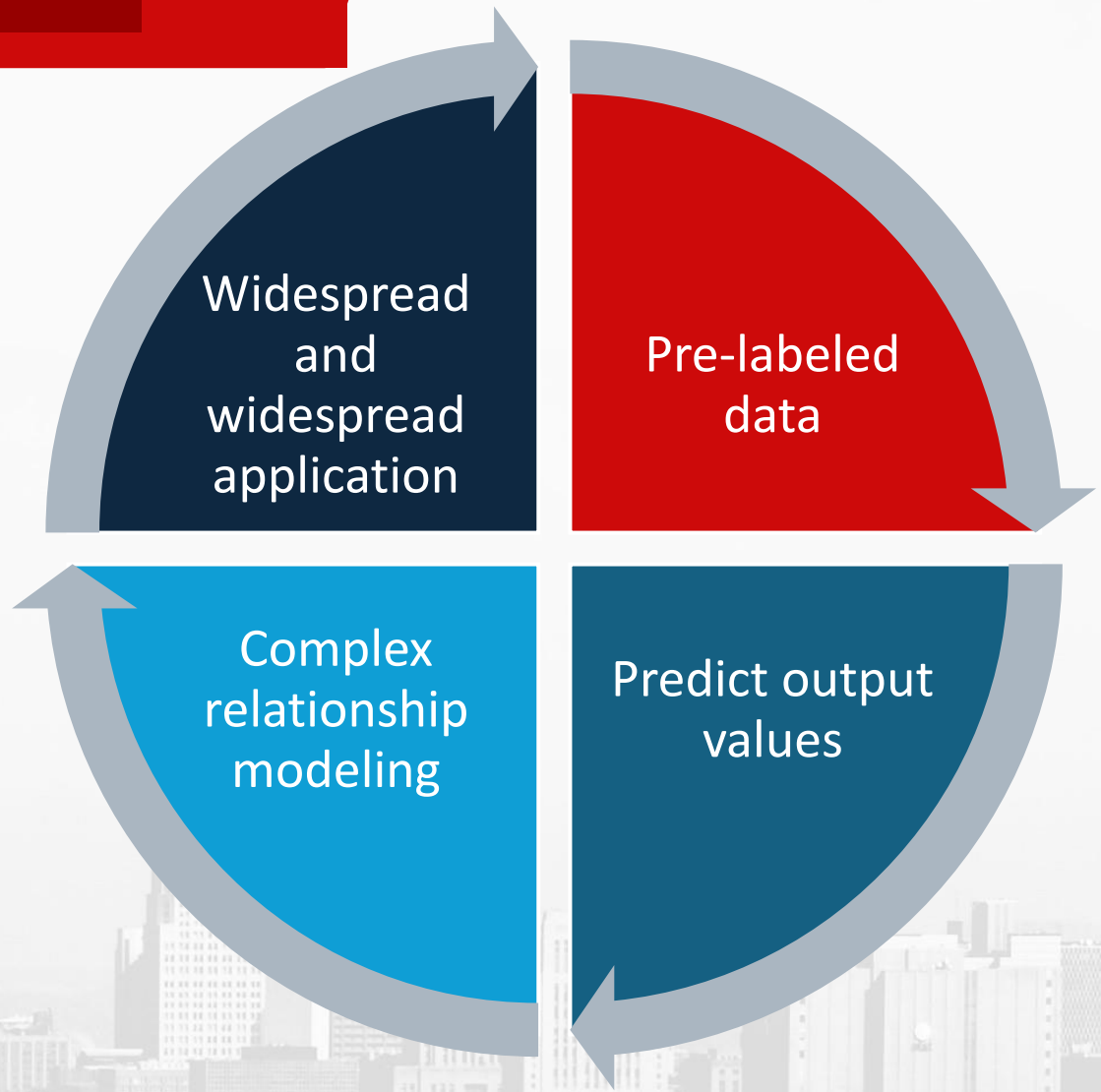
Results of K-Means Clustering

```
np.random.seed(42)
df['centroid'], df['error'], centroids =  kmeans(df[['Gender','Age','Annual Income (k$)','Spending Score (1-100)']], 5)
df.head()
```

	Gender	Age	Annual Income (k$)	Spending Score (1-100)	centroid	error
0	1	19	15	39	2	1.305645e+06
1	1	21	15	81	1	4.081887e+04
2	0	20	16	6	2	9.305465e+05
3	0	23	16	77	1	2.220005e+04
4	0	31	17	40	2	4.266312e+05

Reasons to apply supervised learning for customer prediction:

- K-nearest Neighbor
- Support Vector Machine
- Decision Tree
- Linear Regression



Results of KNN

```python
# Create and fit the KNN classifier
knn = KNN(num_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
  correct = 0
  for i in range(len(actual)):
    if actual[i] == predicted[i]:
      correct += 1
      print("Actual: " + str(actual[i]) + ' vs Predicted:' + str(predicted[i]))
  return correct / float(len(actual)) * 100.0

# Evaluate the model (example using accuracy)
accuracy = accuracy_metric(y_test, y_pred)
print('Accuracy: %.3f%%' % accuracy)
```

```
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Actual: 3 vs Predicted:3
Actual: 1 vs Predicted:1
Accuracy: 97.500%
```

Critically evaluate why machine learning is essential to the design of intelligent machines.

Learning from Data and Adaptation

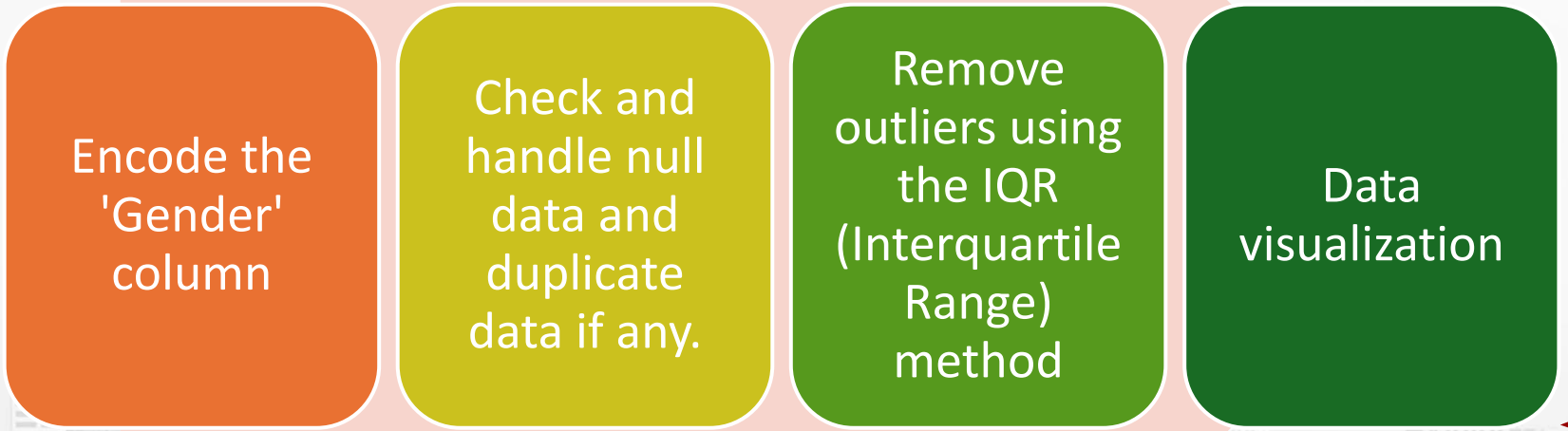
Automation and Increased Efficiency

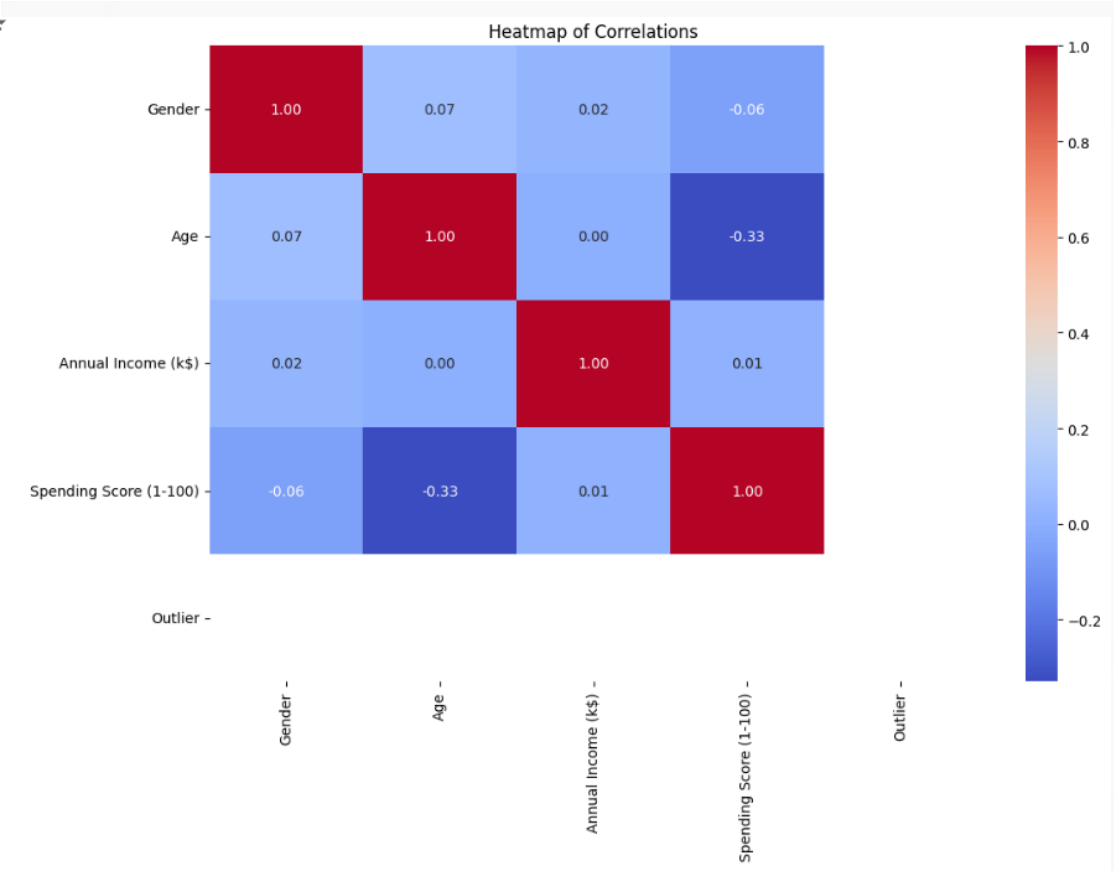
Personalization of User Experience

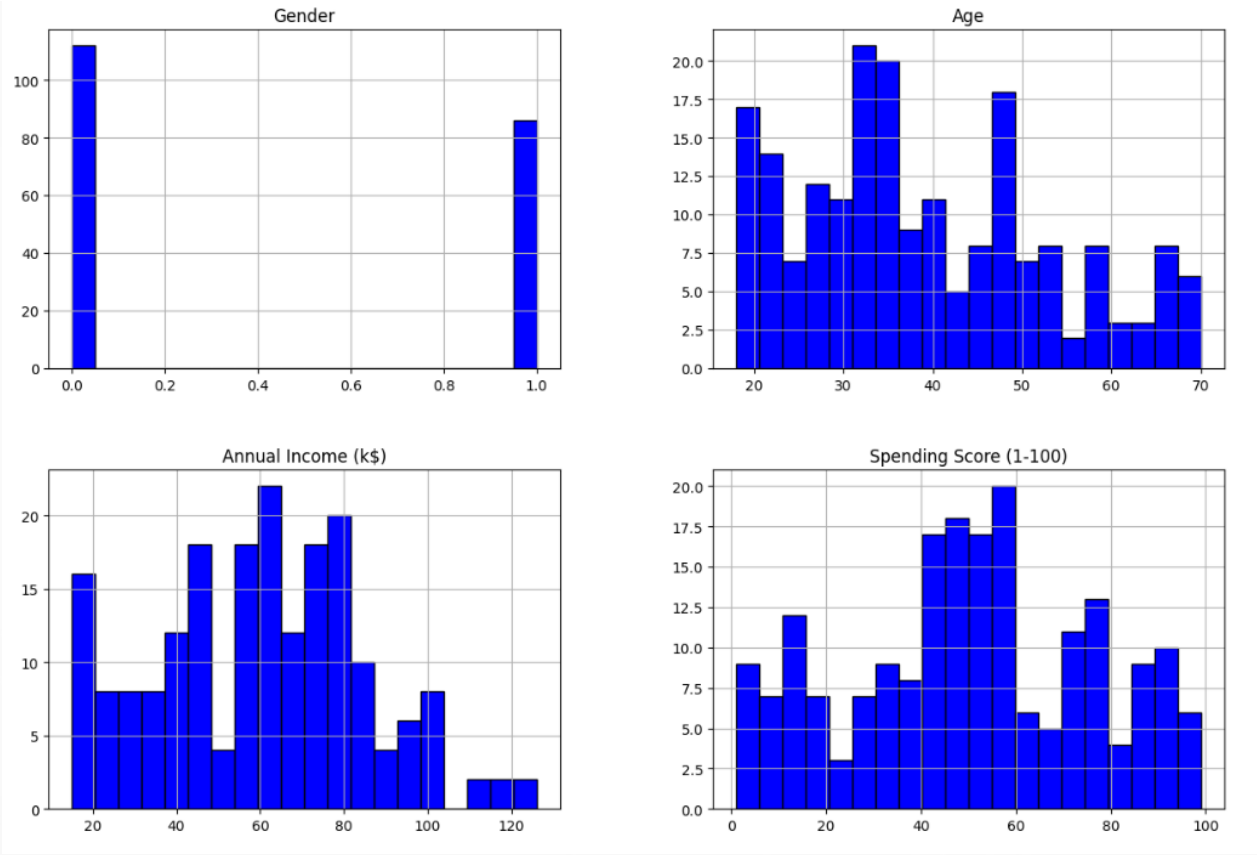
Challenges in Applying Machine Learning

Ethical and Data Security Issues

Prepare training and test data sets in order to implement a machine learning solution.



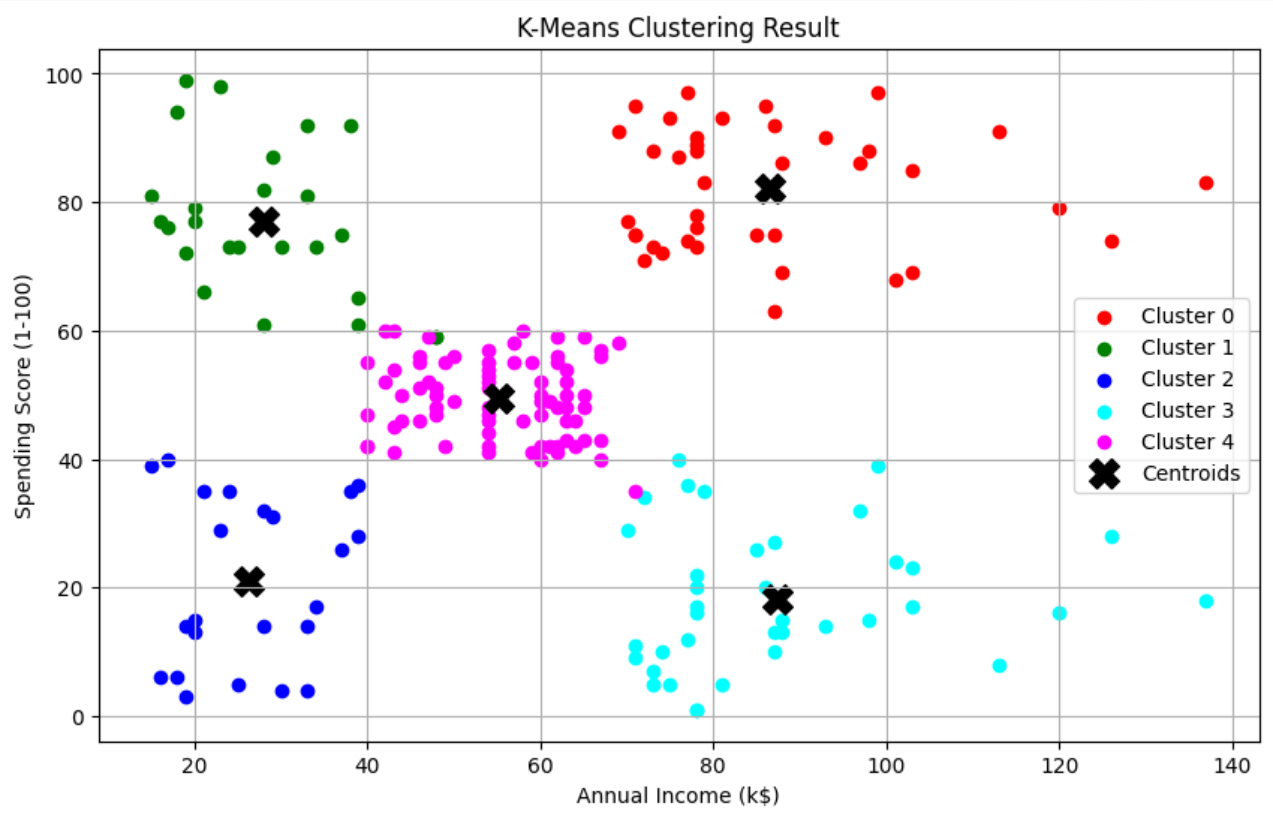


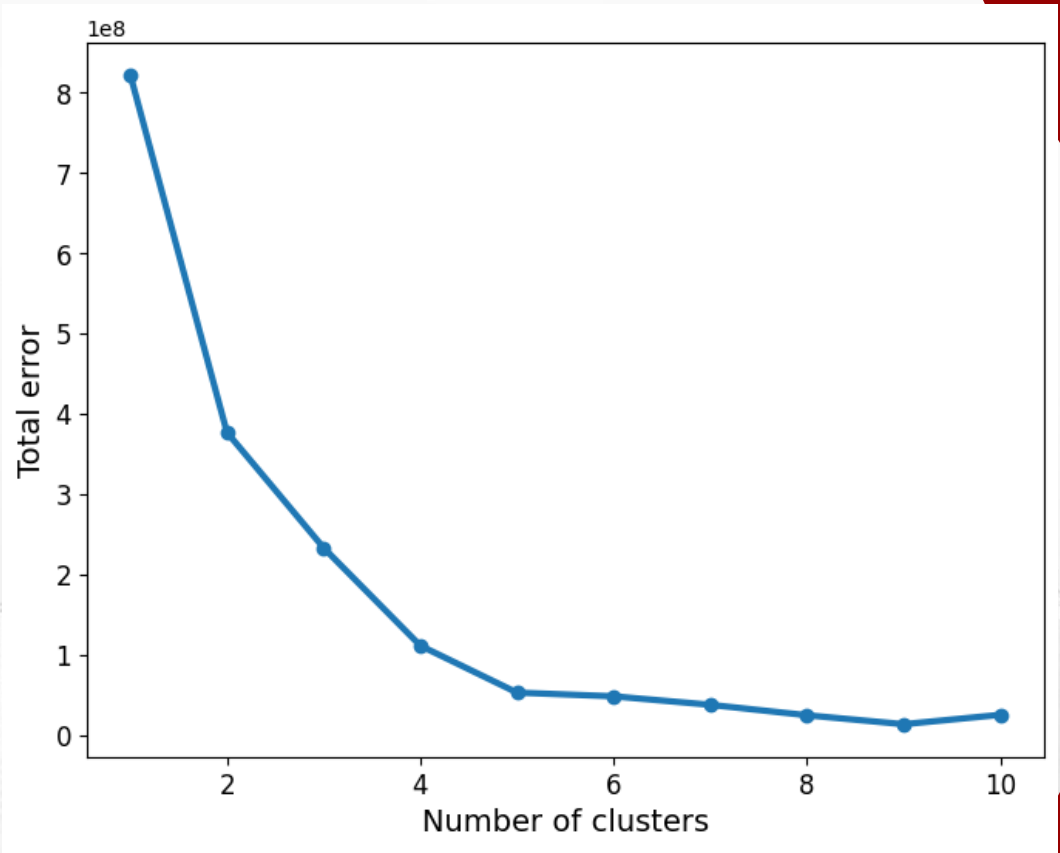


The elements in this data are generally not significantly correlated with each other, they are quite independent and do not influence each other in this data set.

Customer data is diverse, unbiased, well-distributed, and well-represents customer characteristics.

Elbow Method and Parameter K

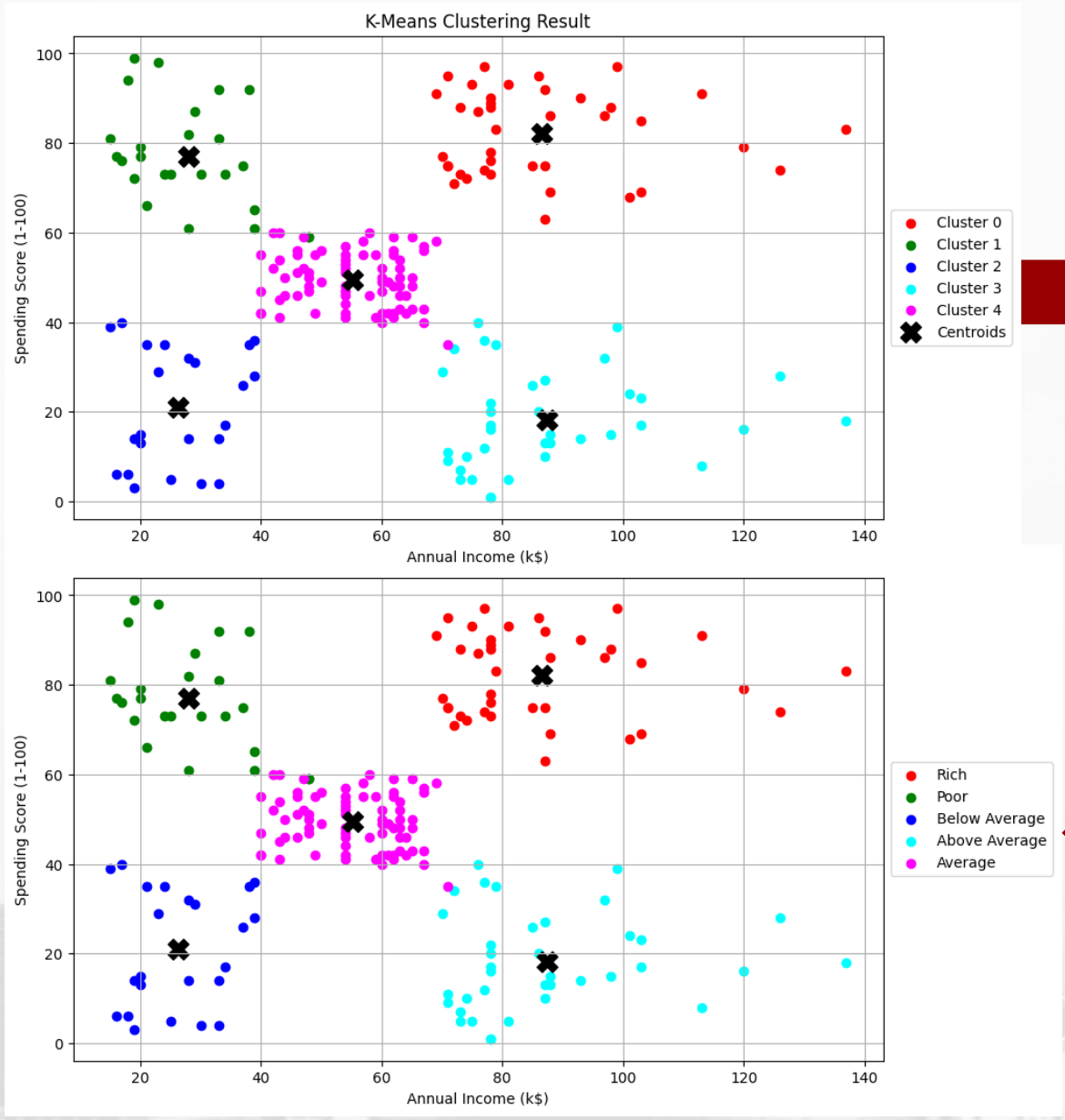




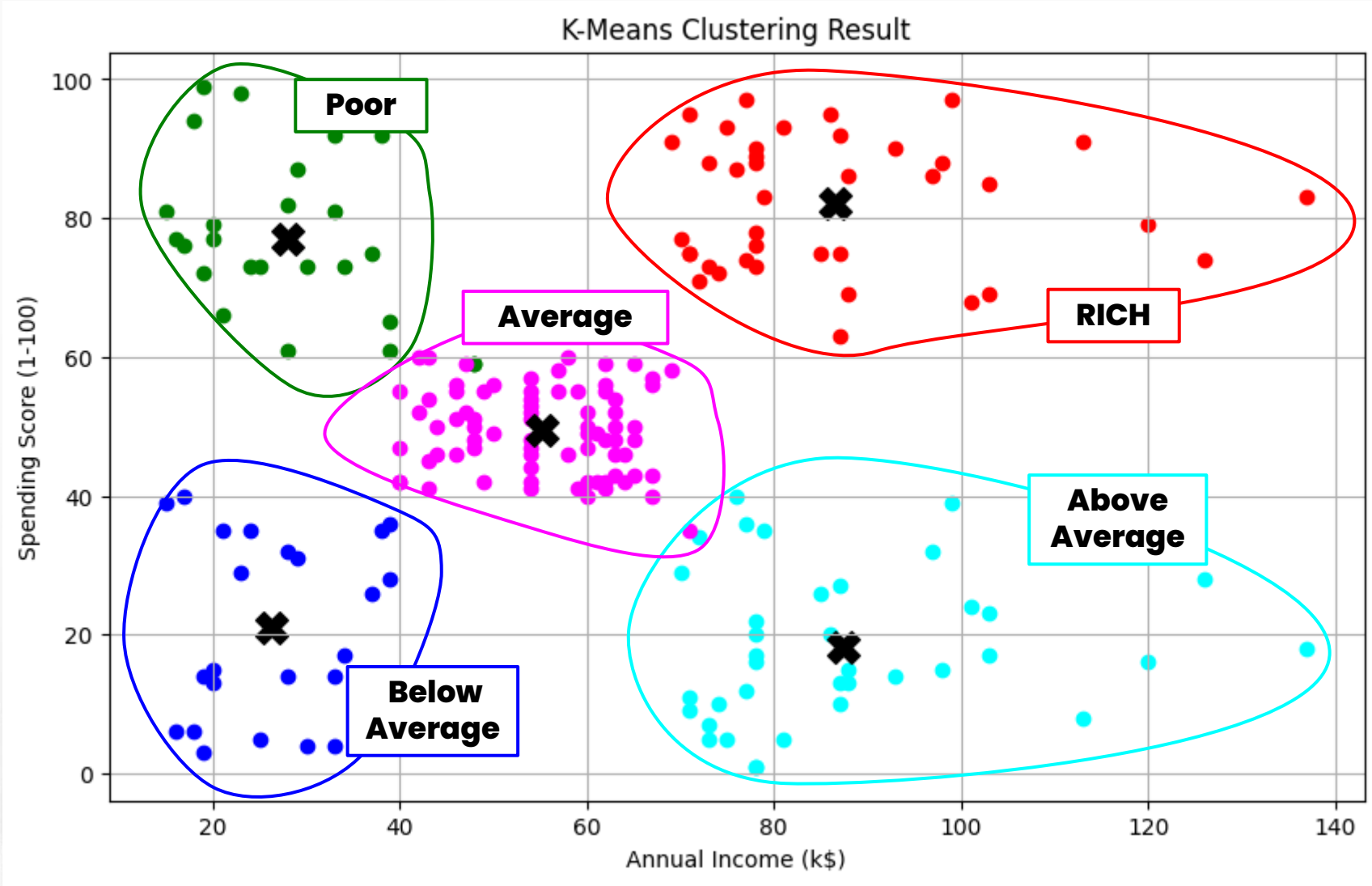
	Gender	Age	Annual Income (k$)	Sp
0	1	19	15	
1	1	21	15	
2	0	20	16	
3	0	23	16	
4	0	31	17	
..	...	...	...	
195	0	35	120	
196	0	45	126	
197	1	32	126	
198	1	32	137	
199	1	30	137	

	error	Label
0	1.305645e+06	Poor
1	4.081887e+04	Below Average
2	9.305465e+05	Poor
3	2.220005e+04	Below Average
4	4.266312e+05	Poor
..	...	...
195	1.288222e+06	Rich
196	2.578706e+06	Above Average
197	2.637538e+06	Rich
198	6.383256e+06	Above Average
199	6.526315e+06	Rich

[200 rows x 7 columns]



Label Data and Testing the Classifier



New data clustering and Evaluation the model

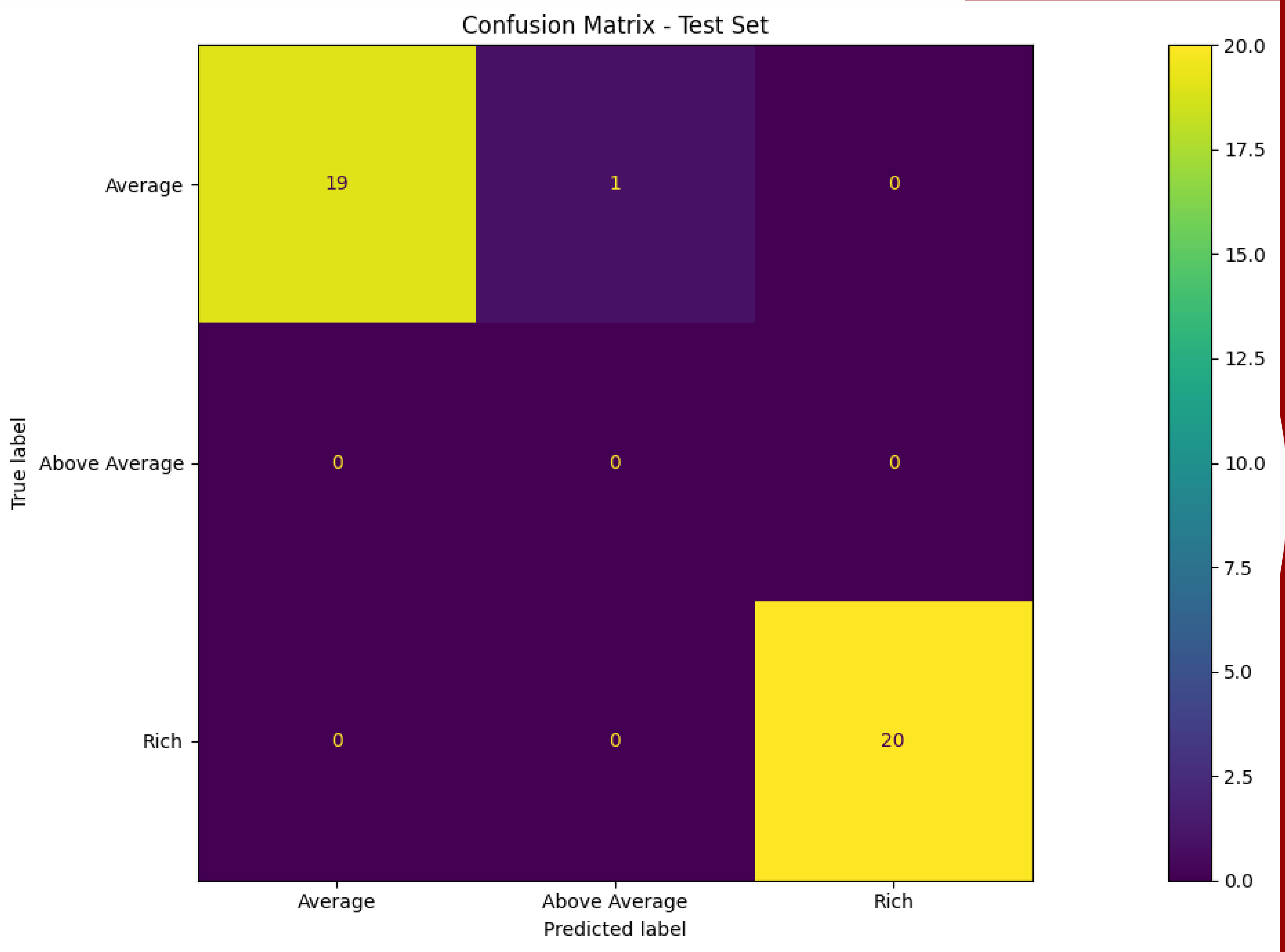
```
Accuracy: 0.975
Precision: 1.0
Recall: 0.975
F1-score: 0.9871794871794872
```

```python
# Create a new data point
new_data = [1, 35, 20, 20]  # Example values for the new data point

# Make a prediction for the new data point
prediction = knn.predict([new_data])

# Print the prediction
print('Prediction for new data:', prediction)
```

```
Prediction for new data: ['Poor']
```

Label Data and Testing the Classifier

```
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
```

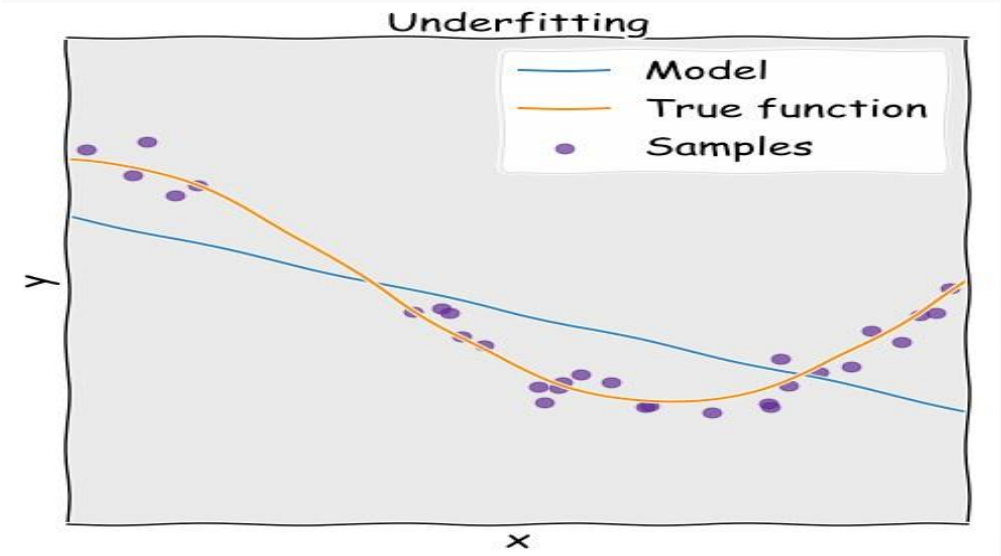


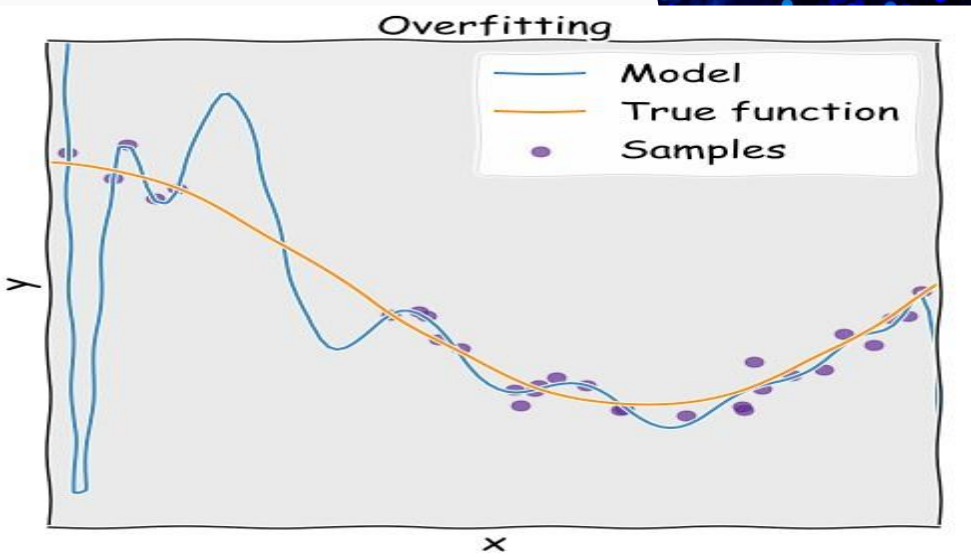
Group 1

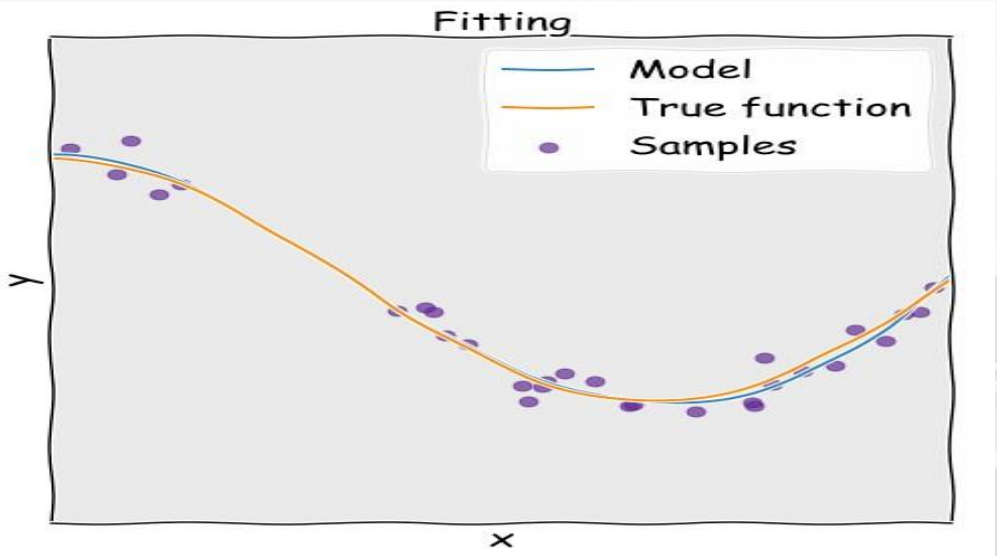
Task 4

Discuss whether the result is balanced, underfitting, or overfitting

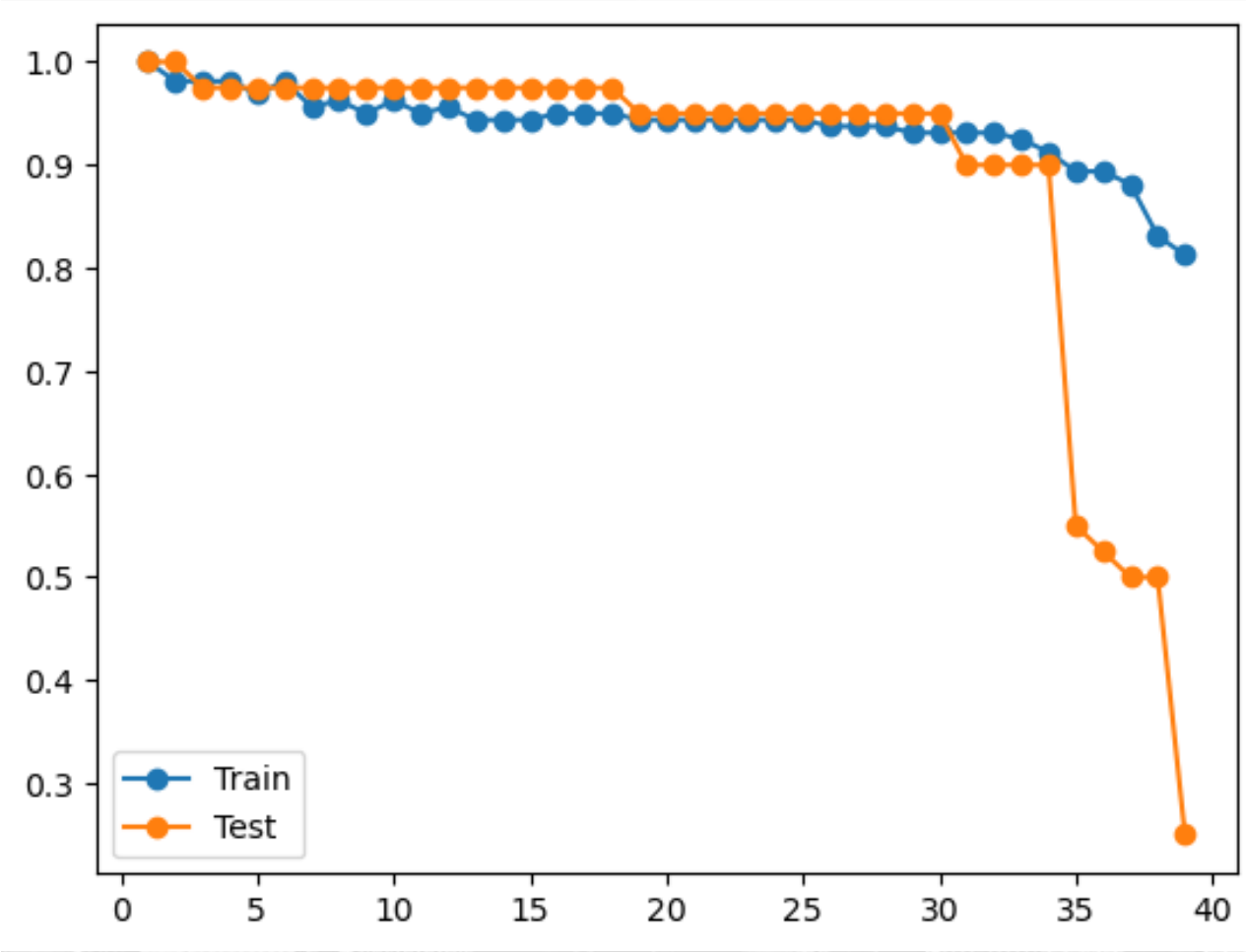
Discuss balanced, underfitting, or overfitting







Compare the results and discuss why your model underfitting and overfitting.

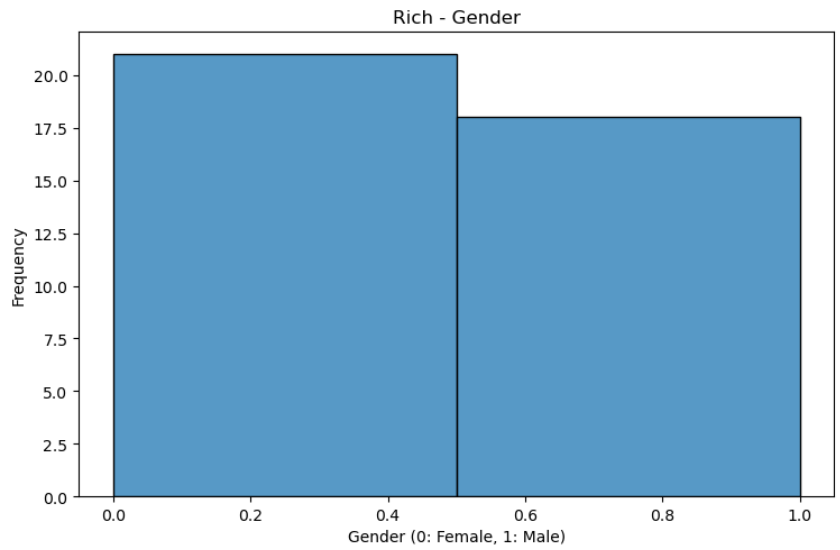


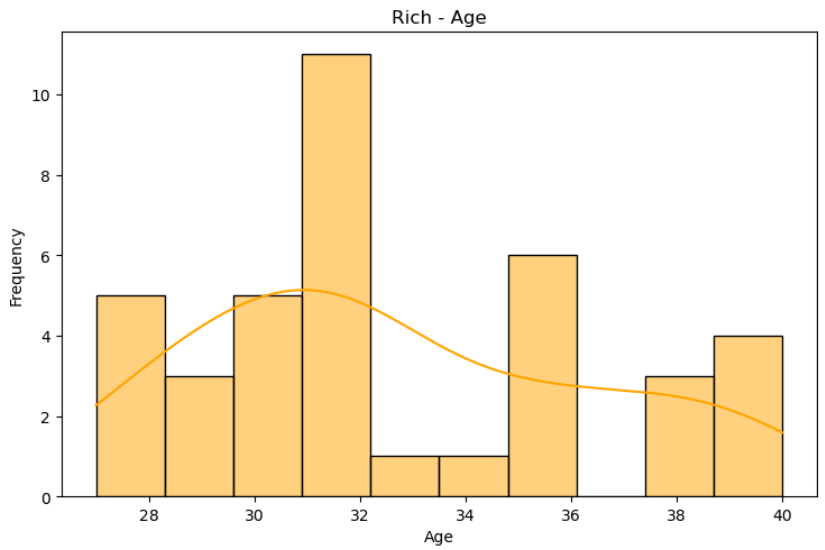

```
>1, train: 1.000, test: 1.000
>2, train: 0.981, test: 1.000
>3, train: 0.981, test: 0.975
>4, train: 0.981, test: 0.975
>5, train: 0.969, test: 0.975
>6, train: 0.981, test: 0.975
>7, train: 0.956, test: 0.975
>8, train: 0.963, test: 0.975
>9, train: 0.950, test: 0.975
>10, train: 0.963, test: 0.975
>30, train: 0.931, test: 0.950
>31, train: 0.931, test: 0.900
>32, train: 0.931, test: 0.900
>33, train: 0.925, test: 0.900
>34, train: 0.912, test: 0.900
>35, train: 0.894, test: 0.550
>36, train: 0.894, test: 0.525
>37, train: 0.881, test: 0.500
>38, train: 0.831, test: 0.500
>39, train: 0.812, test: 0.250
```

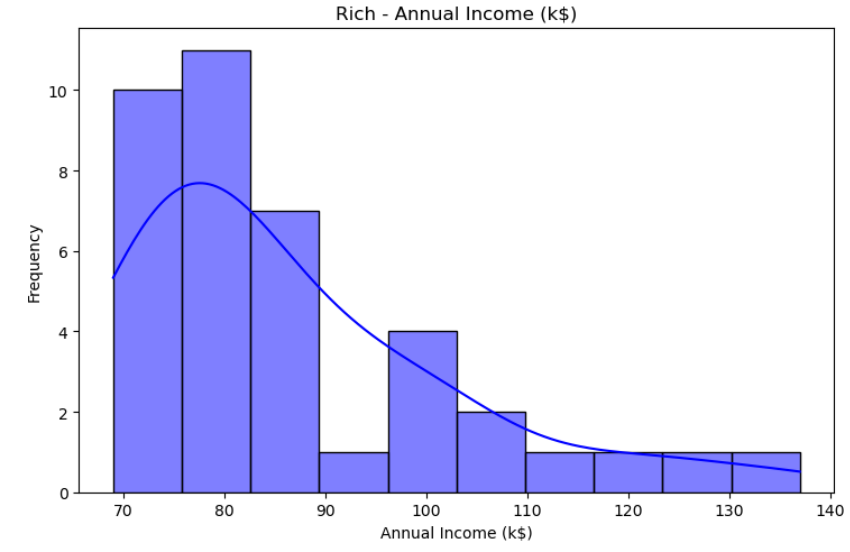
Produce solutions if the model is underfitting or overfitting

Measure	Overfitting (Small K)	Underfitting (Large K)	Description
Choosing an appropriate K value	Choosing a very small K can lead to overfitting	Choosing a very large K can lead to underfitting	Use cross-validation to find the optimal K value.
Data Normalization	Reduces the impact of unbalanced features	Reduces the impact of unbalanced features	Normalize features to the same scale, such as using Min-Max Scaling.
Eliminating irrelevant features	Reduces unnecessary complexity	Improves focus on important features	Remove irrelevant features or use PCA to reduce dimensionality.
Enhancing the training data (Data Augmentation)	Improves generalization ability	Reduces dependency on limited data	Collect more data or generate additional data from existing data.

Produce some business strategies on the result of clustering





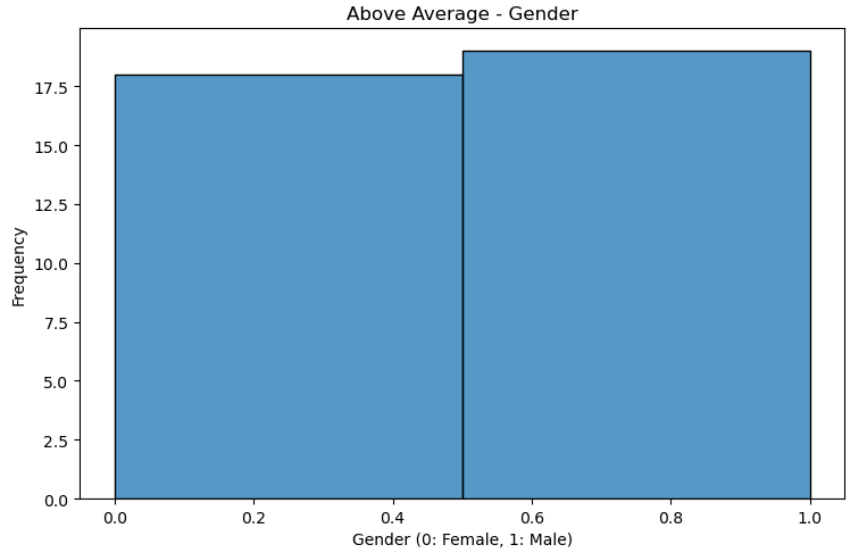


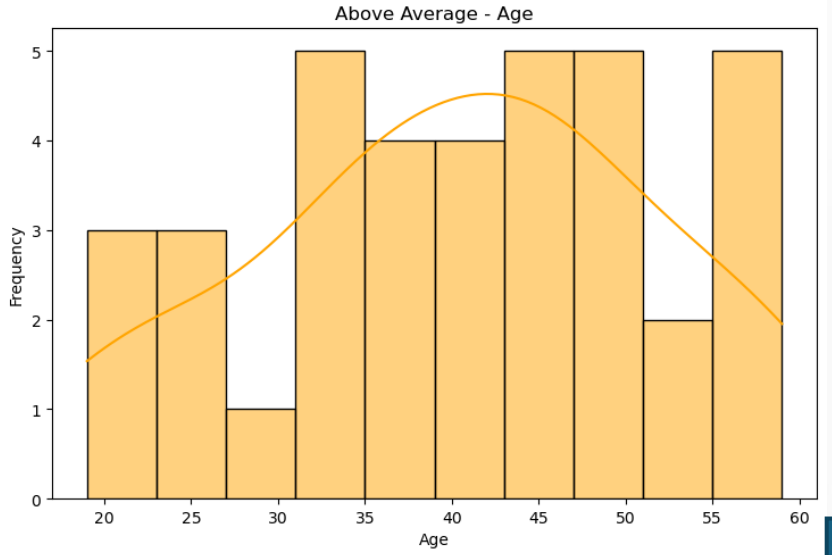
Premium Products and Services

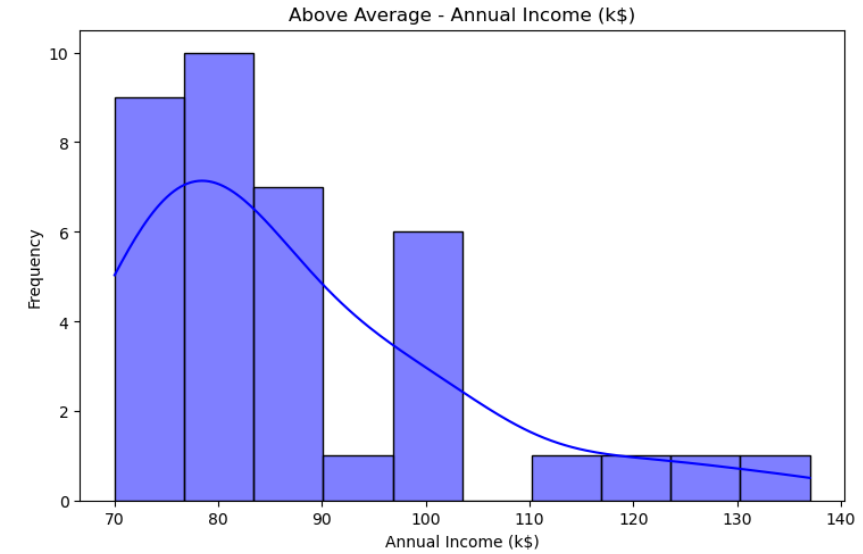
Exclusive Events

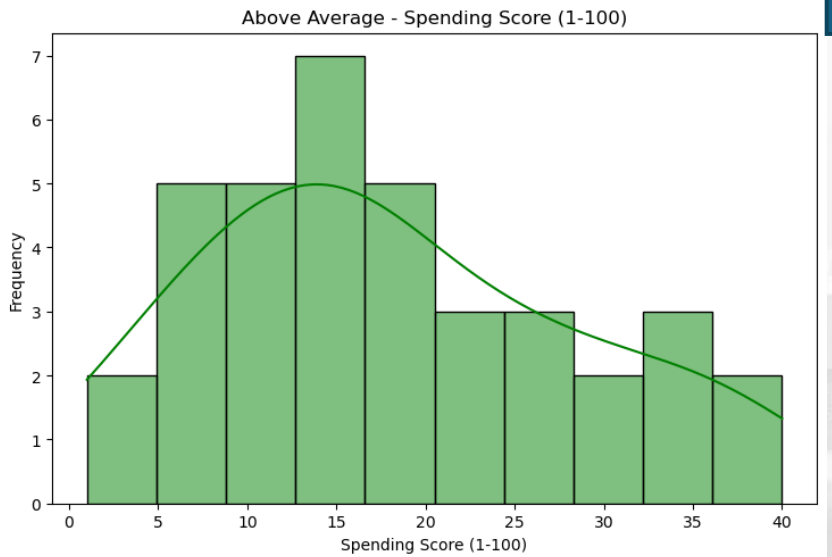
Loyalty Programs

Produce some business strategies on the result of clustering







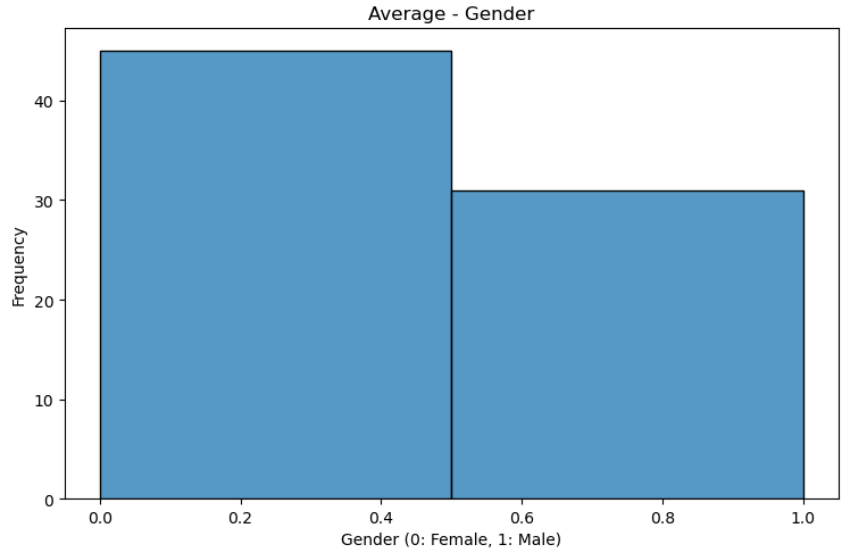


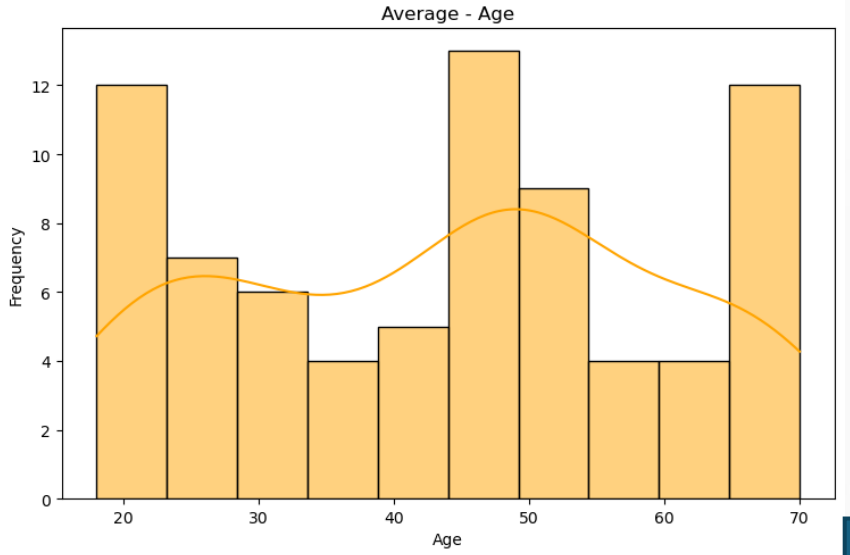
Discounts and Promotions

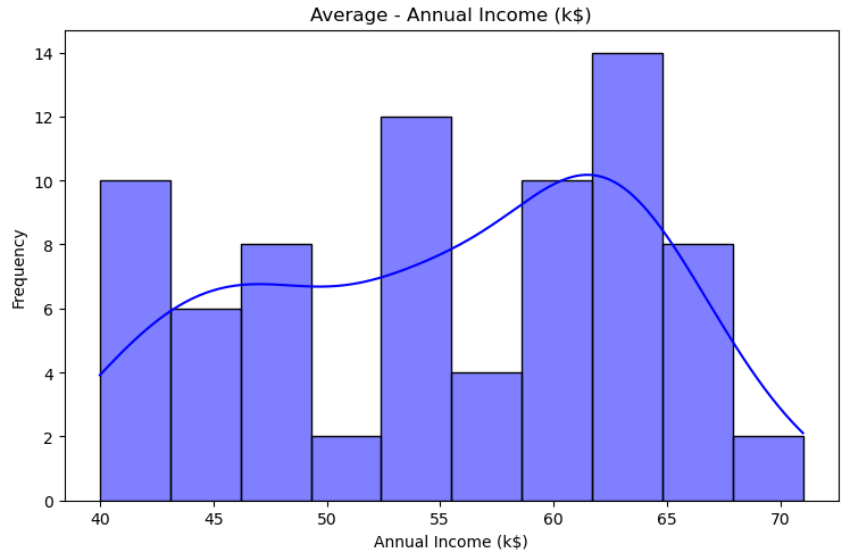
Loyalty Programs

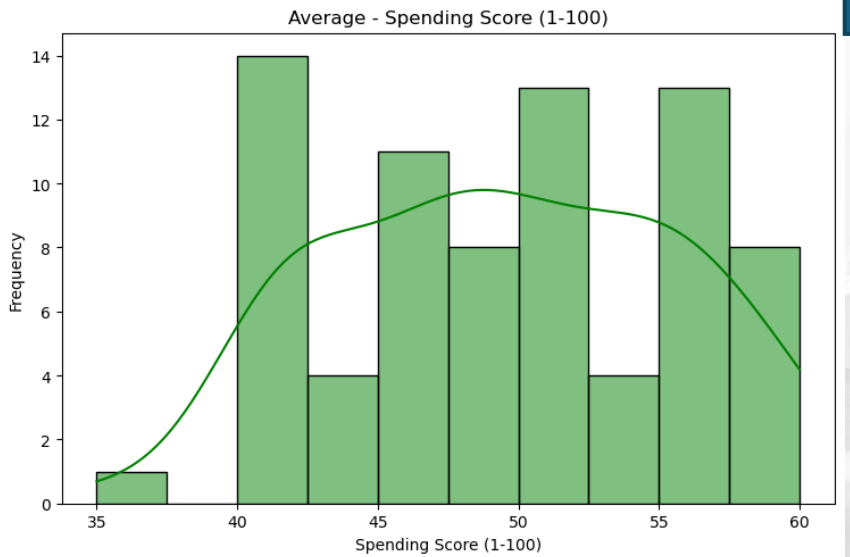
Customer Service

Produce some business strategies on the result of clustering







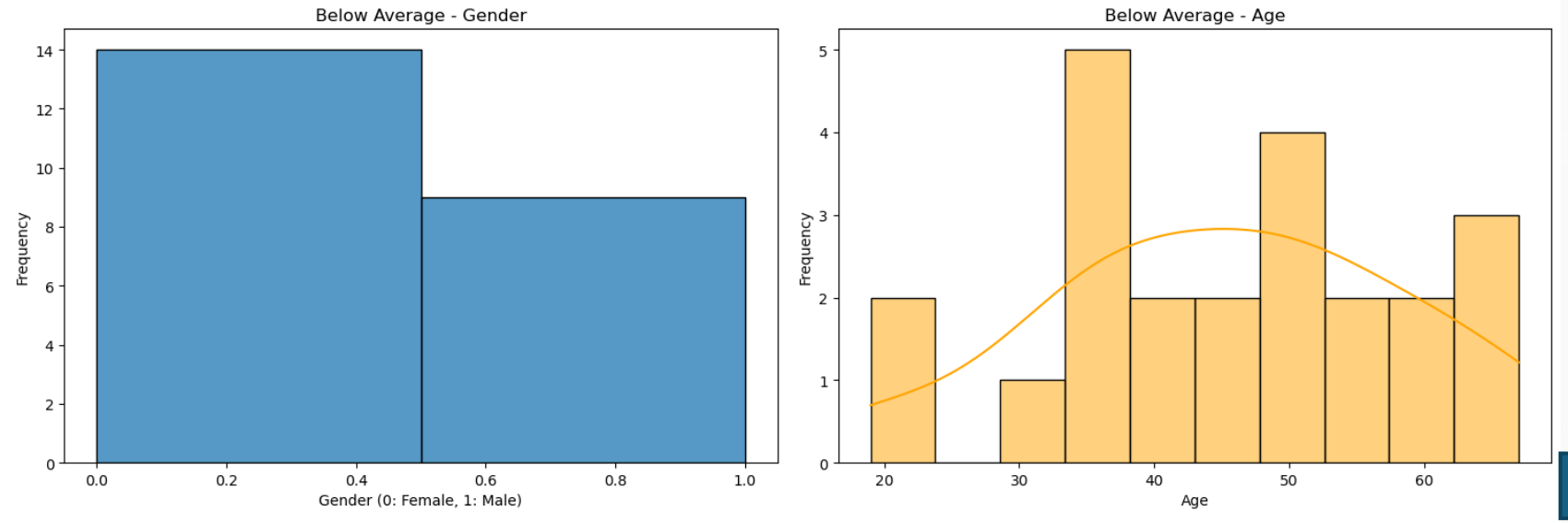


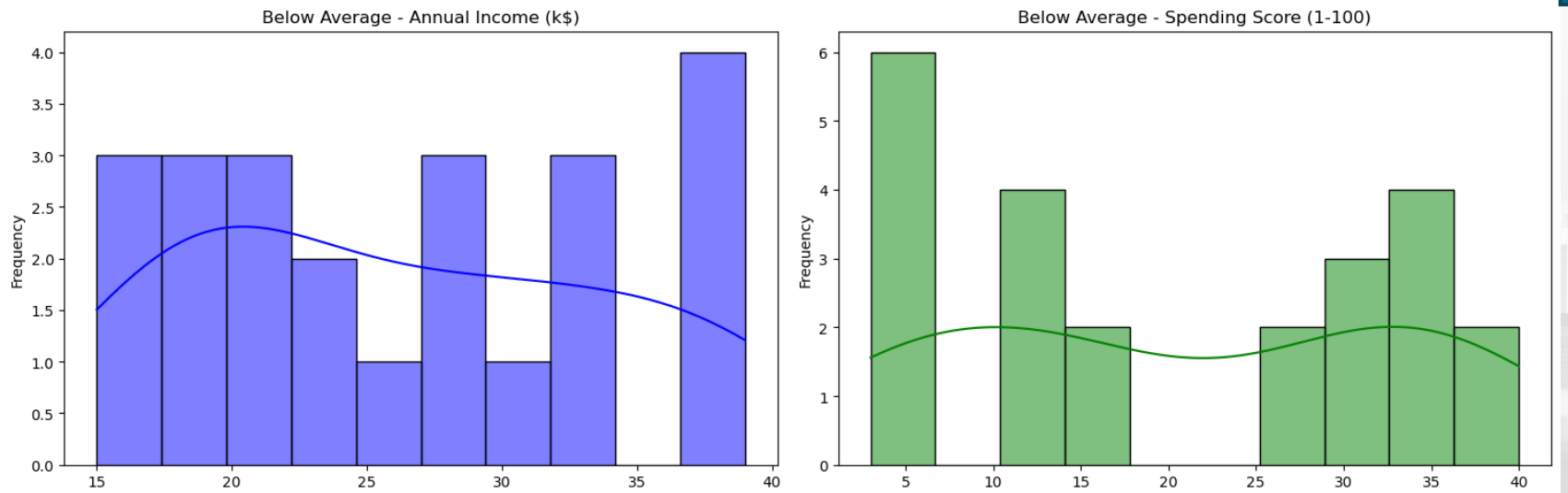
Seasonal Promotions

Loyalty Programs

Family Events

Produce some business strategies on the result of clustering



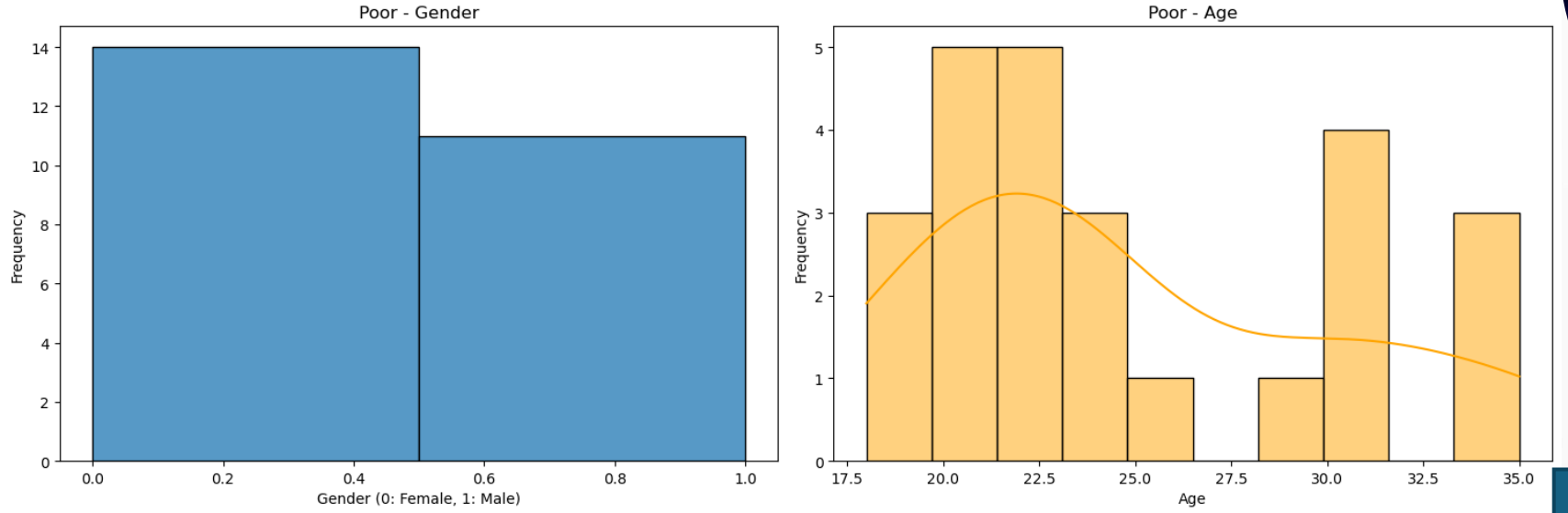


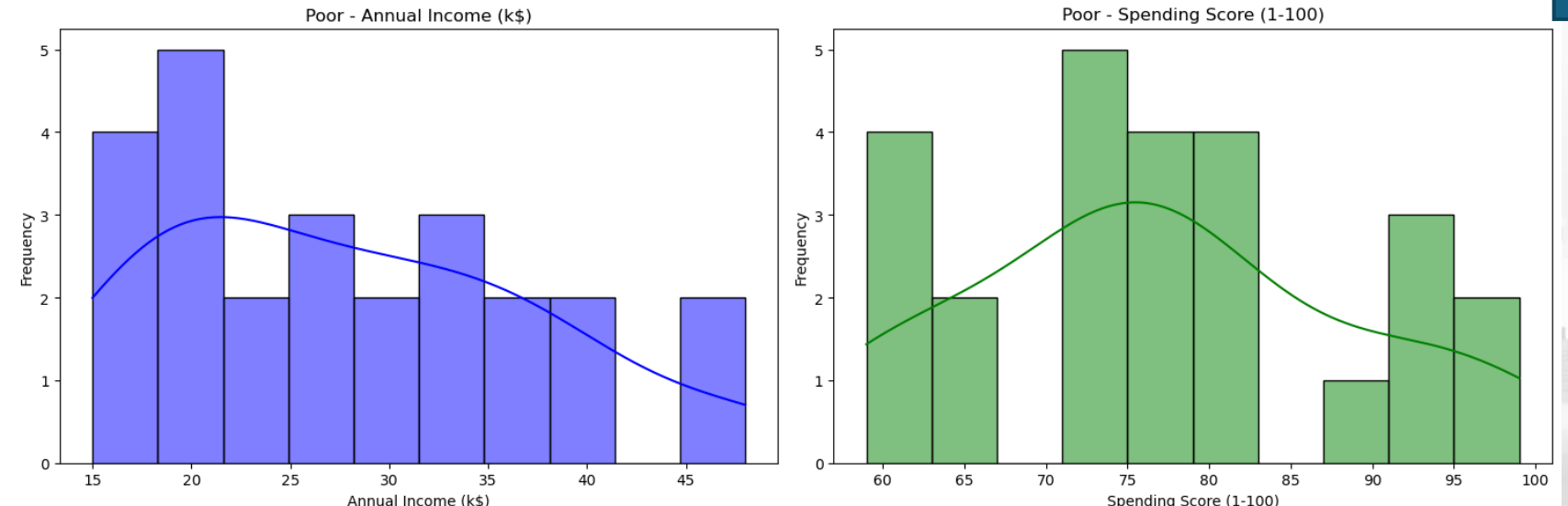
Low-Cost Promotions

Financial Support Programs

Community Events

Produce some business strategies on the result of clustering





- Deep Discounts
- Financial Support Programs
- Community Events

D2 Critically evaluate the implemented learning solution and its effectiveness in meeting end user requirements

KNN

- **Correctness:** Implements KNN logic accurately but lacks optimization.
- **Scalability:** Inefficient for large datasets; no KD-tree or Ball-tree usage.
- **Efficiency:** Nested loops slow down distance calculations.
- **Evaluation Metrics:** Focuses mainly on accuracy.
- **End-User:** Simple and understandable but may fall short on performance.

K-means

- **Correctness:** Follows the K-means process correctly; includes centroid initialization, assignment, and update.
- **Efficiency:** Could be optimized; uses nested loops for distance calculations.
- **Scalability:** Struggles with large datasets; lacks parallel processing.
- **Initialization:** Uses random initialization, which may lead to poor clustering.
- **Visualization:** Provides clear visualizations.
- **End-User :** Suitable for basic tasks but needs enhancements for complex applications.

References

An, C. H. N. K., 2023. Machine Learning Basics. [Online]
Available at: https://viblo.asia/p/machine-learning-co-ban-lesson-01-so-luoc-ve-machine-learning-W13VMgMdJY7
[Accessed 12 7 2024].

Le, K., 2023. What is machine learning?. [Online]
Available at: https://topdev.vn/blog/machine-learning-la-gi/#supervised-learning
[Accessed 15 7 2024].

Pratt, M. K., 2024. Top 12 machine learning use cases and business applications. [Online]
Available at: https://www.techtarget.com/searchenterpriseai/feature/Top-12-machine-learning-use-cases-and-business-applications
[Accessed 15 7 2024].

Alpaydin, E., 2020. Introduction to Machine Learning. 4th edition ed. s.l.:The MIT Press.

Andreas Müller, Sarah Guido , 2016. Introduction to Machine Learning with Python: A Guide for Data Scientists. 1st edition ed. s.l.:O'Reilly Media.

Brownlee, J., 2020. Develop k-Nearest Neighbors in Python From Scratch. [Online]
Available at: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
[Accessed 10 7 2024].

ROGEL-SALAZAR, D. J., 2023. Getting started with k-means clustering in Python. [Online]
Available at: https://domino.ai/blog/getting-started-with-k-means-clustering-in-python
[Accessed 10 7 2024].

SHRUTI_IYYER, 2019. Step by Step KMeans Explained in Detail. [Online]
Available at: https://www.kaggle.com/code/shrutimechlearn/step-by-step-kmeans-explained-in-detail
[Accessed 9 7 2024].

Group 1

Thank You

For Your Attention

k-means-clustering-scratch

August 12, 2024

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import warnings
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

blobs = pd.read_csv('Mall_Customers.csv')
colnames = list(blobs.columns[1:-1])
blobs.head()
```

[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40

[ ]:
```
# Encode the 'Gender' column
label_encoder = LabelEncoder()
blobs['Gender'] = label_encoder.fit_transform(blobs['Gender'])

print(blobs.head())
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40

[ ]:
```
# Display information of DataFrame
blobs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    int64
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(5)
memory usage: 7.9 KB
```

[ ]:
```python
# Counts the number of null values (or NaN) in each column of a DataFrame.
print(blobs.isnull().sum())
```

```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

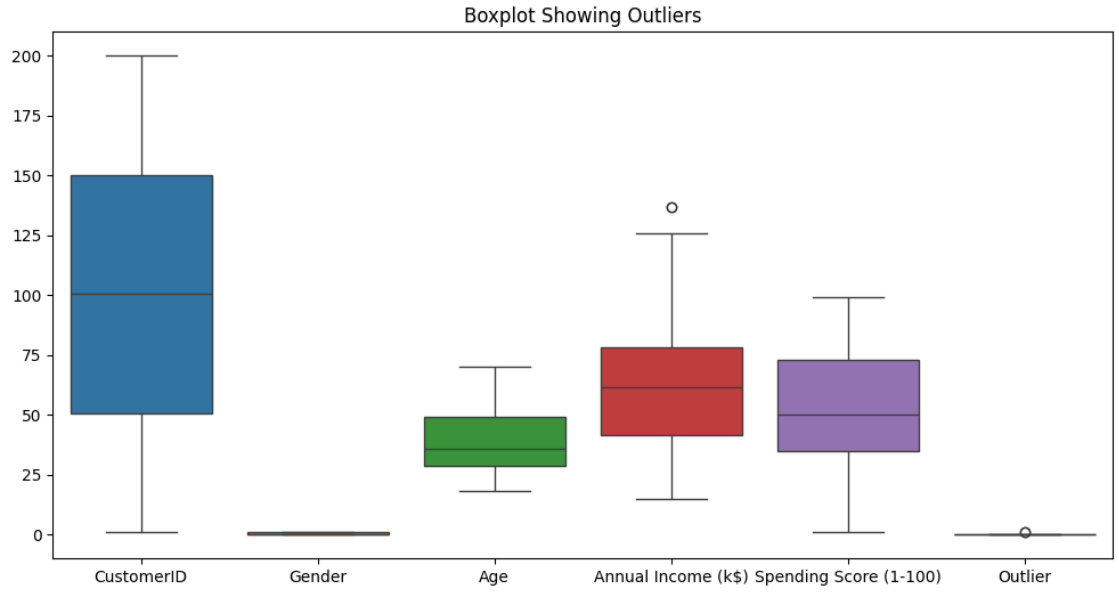
[ ]:
```python
# Identify and count the total number of duplicate rows.
print("Number of duplicate rows: ", blobs.duplicated().sum())
```

```
Number of duplicate rows:  0
```

[ ]:
```python
# Remove outliers using the IQR (Interquartile Range) method
Q1 = blobs.quantile(0.25)
Q3 = blobs.quantile(0.75)
IQR = Q3 - Q1

# Identify outliers
outliers = ((blobs < (Q1 - 1.5 * IQR)) | (blobs > (Q3 + 1.5 * IQR))).any(axis=1)

# Shows rows containing outliers
print("Rows containing outliers:")
print(blobs[outliers])

# Highlight rows that contain outliers
blobs['Outlier'] = outliers
```

```
Rows containing outliers:
     CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
198         199       1   32                 137                      18
199         200       1   30                 137                      83
```

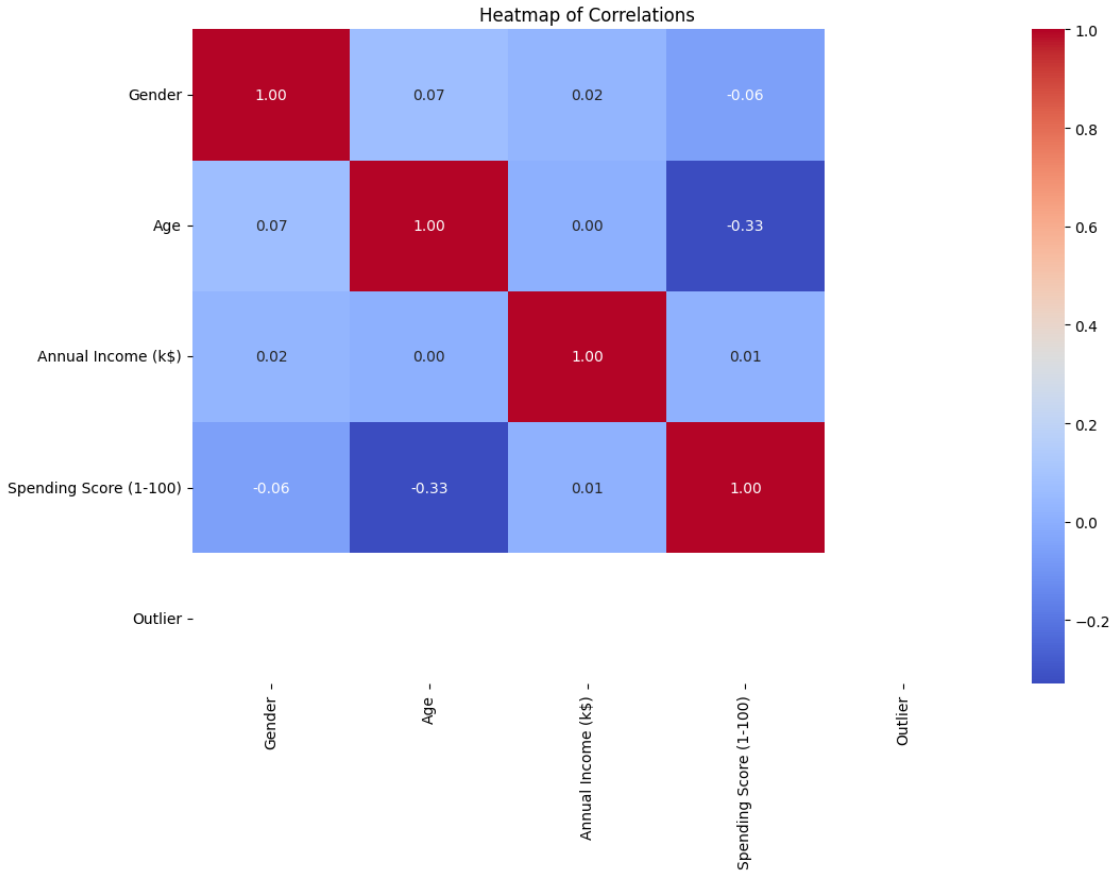
[ ]:
```
# Visualize data with box plots to see outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=blobs)
plt.title("Boxplot Showing Outliers")
plt.show()

# Remove outliers from DataFrame
blobs = blobs[~outliers]

blobs.info()
```



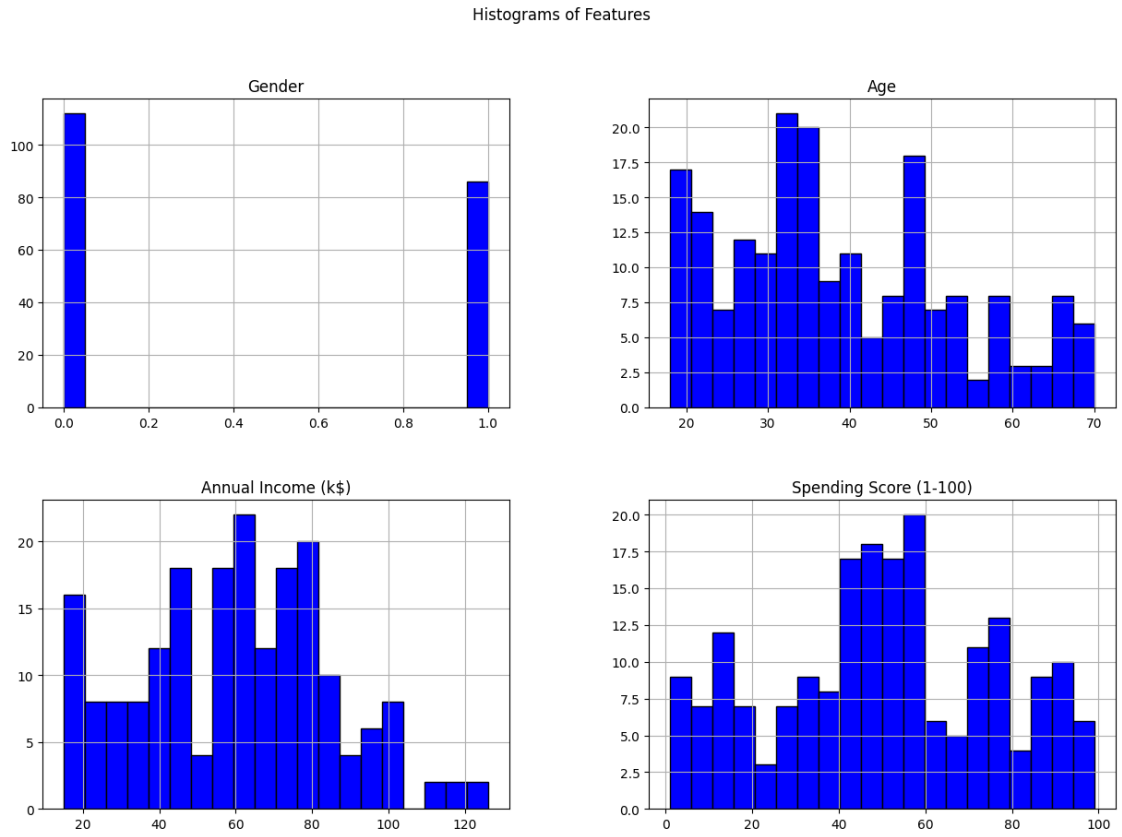

```
<class 'pandas.core.frame.DataFrame'>
Index: 198 entries, 0 to 197
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              198 non-null    int64
 1   Gender                  198 non-null    int64
 2   Age                     198 non-null    int64
 3   Annual Income (k$)      198 non-null    int64
 4   Spending Score (1-100)  198 non-null    int64
 5   Outlier                 198 non-null    bool
dtypes: bool(1), int64(5)
memory usage: 9.5 KB
```

[ ]:
```
# Drop 'CustomerID' column for visualization
blobs_viz = blobs.drop(columns=['CustomerID'])
```

```
# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(blobs_viz.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title('Heatmap of Correlations')
plt.show()
```



[ ]:
```
# Plot histograms for each column
blobs_viz.hist(figsize=(15, 10), bins=20, color='blue', edgecolor='black')
plt.suptitle('Histograms of Features')
plt.show()
```



From the histograms above, we can draw some characteristics of the customer group in the data

0.0.1 Characteristics of the customer group:

1. **Gender:**

- The number of female customers is larger than that of male customers.
- The data on gender is not too skewed, because the number of men and women is still quite balanced, although there is a difference.

2. **Age:**

- Customers belong to many different ages, but are concentrated in the age group of around 30 and 40-50.
- No age group is absolutely dominant, showing the diversity of customer ages.

3. **Annual Income (k$):**

- Customers' annual income ranges from 20 to 120 thousand dollars.
- There are some common income levels such as around 60 thousand dollars, but no income level is absolutely dominant.
- Income data is quite diverse and not skewed much.

4. **Spending Score (1-100):**

- Spending scores are distributed quite evenly from 1 to 100.
- Some spending scores such as around 40-60 have a higher frequency, but are not too different from other scores.
- Spending score distribution is quite even, not skewed.

0.0.2 Assessing data skewness:

- **Gender:** Not too skewed, because both genders have a significant number of customers.
- **Age:** Distribution is quite diverse, not skewed towards a specific age.
- **Annual Income (k$):** Not skewed much, although there are some more popular income levels but they are not absolutely dominant.
- **Spending Score (1-100):** Spending score distribution is quite even, not skewed.

In summary, from the histograms, we can see that the customer base in this data is quite diverse in terms of gender, age, income, and spending points. The data is not skewed much, showing balance and good representation of customer characteristics.

[ ]:
```
colnames
```

[ ]:
```
['Gender', 'Age', 'Annual Income (k$)']
```

[ ]:
```
def initiate_centroids(k, dset):
    '''
    Select k data points as centroids
    k: number of centroids
    dset: pandas dataframe
    '''
    centroids = dset.sample(k)
    return centroids

np.random.seed(42)
k=3
df = blobs[['Gender','Age','Annual Income (k$)','Spending Score (1-100)']]
centroids = initiate_centroids(k, df)
centroids
```

[ ]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
65	1	18	48	59
114	0	18	65	48
16	0	35	21	35

[ ]:
```
def rsserr(a,b):
    '''
    Calculate the root of sum of squared errors.
```

```
    a and b are numpy arrays
    '''
    return np.square(np.sum((a-b)**2))
```

[ ]:
```
for i, centroid in enumerate(range(centroids.shape[0])):
    err = rsserr(centroids.iloc[centroid,:], df.iloc[36,:])
    print('Error for centroid {0}: {1:.2f}'.format(i, err))
```

```
Error for centroid 0: 6436369.00
Error for centroid 1: 6240004.00
Error for centroid 2: 293764.00
```

[ ]:
```
def centroid_assignation(dset, centroids):
    '''
    Given a dataframe `dset` and a set of `centroids`, we assign each
    data point in `dset` to a centroid.
    - dset - pandas dataframe with observations
    - centroids - pa das dataframe with centroids
    '''
    k = centroids.shape[0]
    n = dset.shape[0]
    assignation = []
    assign_errors = []

    for obs in range(n):
        # Estimate error
        all_errors = np.array([])
        for centroid in range(k):
            err = rsserr(centroids.iloc[centroid, :], dset.iloc[obs,:])
            all_errors = np.append(all_errors, err)

        # Get the nearest centroid and the error
        nearest_centroid =  np.where(all_errors==np.amin(all_errors))[0].
 ↪tolist()[0]
        nearest_centroid_error = np.amin(all_errors)

        # Add values to corresponding lists
        assignation.append(nearest_centroid)
        assign_errors.append(nearest_centroid_error)

    return assignation, assign_errors
```

[ ]:
```
df['centroid'], df['error'] = centroid_assignation(df, centroids)
df.head()
```

```
<ipython-input-16-e891e7f5dc1a>:1: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:
```
   Gender  Age  Annual Income (k$)  Spending Score (1-100)  centroid  \
0       1   19                  15                      39         2
1       1   21                  15                      81         0
2       0   20                  16                       6         2
3       0   23                  16                      77         0
4       0   31                  17                      40         2

        error
0     95481.0
1   2502724.0
2   1190281.0
3   1887876.0
4      3249.0
```

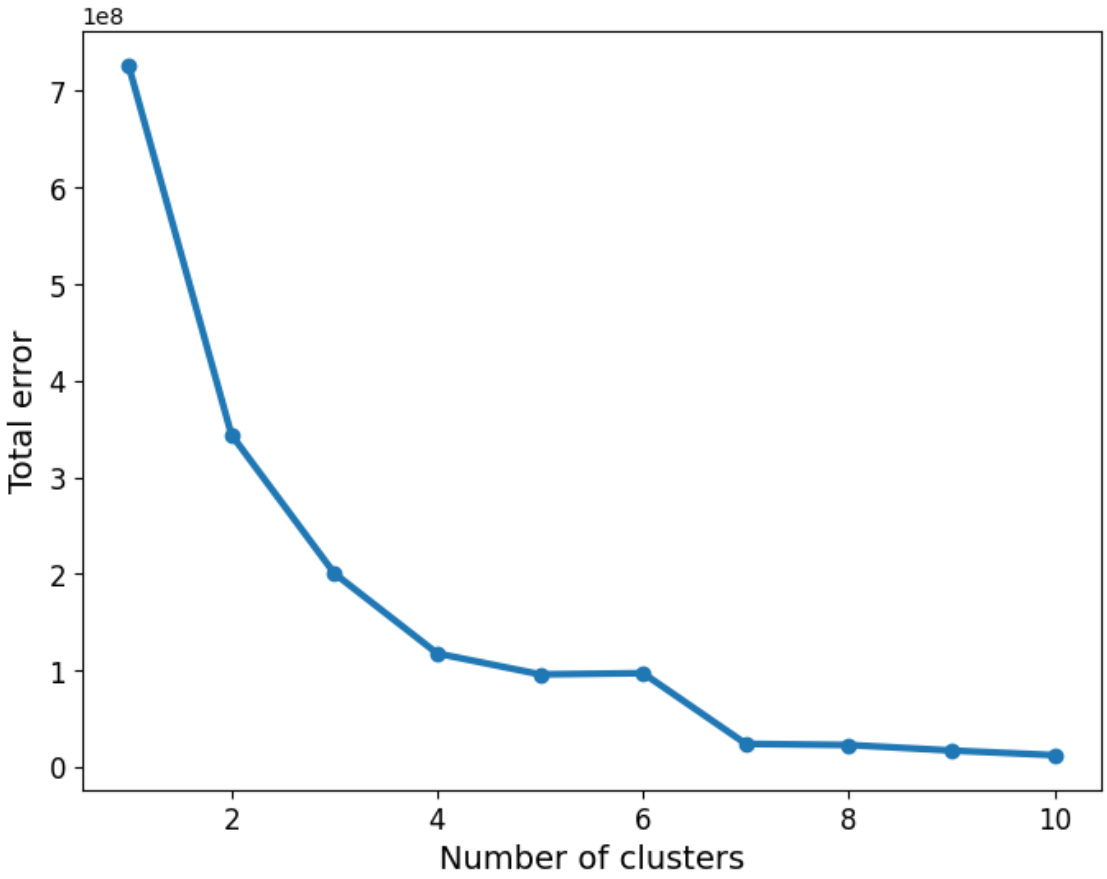
[ ]:
```
centroids = df.groupby('centroid').agg('mean').loc[:, colnames].
 ↪reset_index(drop = True)
centroids
```

[ ]:
```
      Gender        Age  Annual Income (k$)
0   0.407407  30.240741           45.055556
1   0.456311  39.174757           77.514563
2   0.414634  49.756098           34.658537
```

[ ]:
```
def kmeans(dset, k=2, tol=1e-4):
    '''
    K-means implementationd for a
    `dset`:  DataFrame with observations
    `k`: number of clusters, default k=2
    `tol`: tolerance=1E-4
    '''
    # Let us work in a copy, so we don't mess the original
    working_dset = dset.copy()
    # We define some variables to hold the error, the
    # stopping signal and a counter for the iterations
    err = []
    goahead = True
    j = 0

    # Step 2: Initiate clusters by defining centroids
```

```
    centroids = initiate_centroids(k, dset)

    while(goahead):
        # Step 3 and 4 - Assign centroids and calculate error
        working_dset['centroid'], j_err = centroid_assignation(working_dset,␣
↪centroids)
        err.append(sum(j_err))

        # Step 5 - Update centroid position
        centroids = working_dset.groupby('centroid').agg('mean').
↪reset_index(drop = True)

        # Step 6 - Restart the iteration
        if j>0:
            # Is the error less than a tolerance (1E-4)
            if err[j-1]-err[j]<=tol:
                goahead = False
        j+=1

    working_dset['centroid'], j_err = centroid_assignation(working_dset,␣
↪centroids)
    centroids = working_dset.groupby('centroid').agg('mean').reset_index(drop =␣
↪True)
    return working_dset['centroid'], j_err, centroids
```

[ ]:
```
err_total = []
n = 10

df_elbow = blobs[['Gender','Age','Annual Income (k$)','Spending Score (1-100)']]

for i in range(n):
    _, my_errs, _ = kmeans(df_elbow, i+1)
    err_total.append(sum(my_errs))
fig, ax = plt.subplots(figsize=(8, 6))
plt.plot(range(1,n+1), err_total, linewidth=3, marker='o')
ax.set_xlabel(r'Number of clusters', fontsize=14)
ax.set_ylabel(r'Total error', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



[ ]:
```
np.random.seed(42)
df['centroid'], df['error'], centroids =  kmeans(df[['Gender','Age','Annual␣
 ↪Income (k$)','Spending Score (1-100)']], 5)
df.head()
```

[ ]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)	centroid	\
0	1	19	15	39	2	
1	1	21	15	81	0	
2	0	20	16	6	2	
3	0	23	16	77	0	
4	0	31	17	40	2	

	error
0	1.305645e+06
1	4.081887e+04
2	9.305465e+05
3	2.220005e+04
4	4.266312e+05

[ ]:
```
df.head(500)
```

[ ]:
```
     Gender  Age  Annual Income (k$)  Spending Score (1-100)  centroid  \
0         1   19                  15                      39         2
1         1   21                  15                      81         0
2         0   20                  16                       6         2
3         0   23                  16                      77         0
4         0   31                  17                      40         2
..      ...  ...                 ...                     ...       ...
193       0   38                 113                      91         3
194       0   47                 120                      16         4
195       0   35                 120                      79         3
196       0   45                 126                      28         4
197       1   32                 126                      74         3

              error
0    1.305645e+06
1    4.081887e+04
2    9.305465e+05
3    2.220005e+04
4    4.266312e+05
..            ...
193  7.726337e+05
194  1.374928e+06
195  1.501002e+06
196  2.884568e+06
197  2.994160e+06

[198 rows x 6 columns]
```

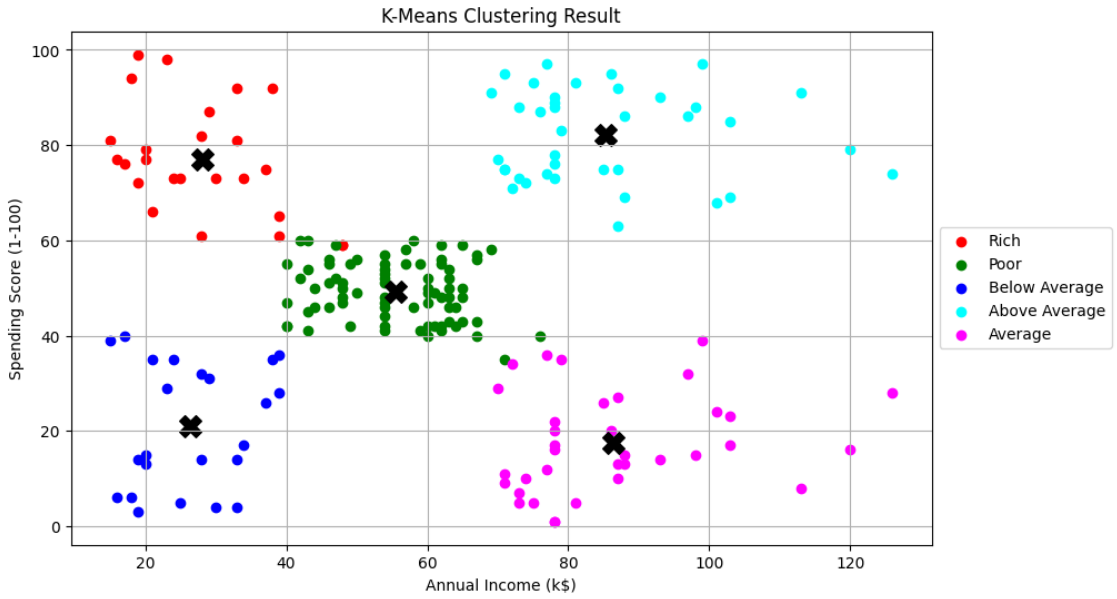
[ ]:
```
# Define the mapping from centroid numbers to labels
centroid_labels = {0: 'Poor', 1: 'Average', 2: 'Below Average', 3: 'Above␣
 ↪Average', 4: 'Rich'}

# Map the centroid numbers to labels
df['centroid_label'] = df['centroid'].map(centroid_labels)
print(df)
```

```
     Gender  Age  Annual Income (k$)  Spending Score (1-100)  centroid  \
0         1   19                  15                      39         2
1         1   21                  15                      81         0
2         0   20                  16                       6         2
3         0   23                  16                      77         0
4         0   31                  17                      40         2
..      ...  ...                 ...                     ...       ...
193       0   38                 113                      91         3
194       0   47                 120                      16         4
```

```
195       0   35                  120                     79          3
196       0   45                  126                     28          4
197       1   32                  126                     74          3

             error centroid_label
0    1.305645e+06  Below Average
1    4.081887e+04           Poor
2    9.305465e+05  Below Average
3    2.220005e+04           Poor
4    4.266312e+05  Below Average
..            …             …
193  7.726337e+05  Above Average
194  1.374928e+06           Rich
195  1.501002e+06  Above Average
196  2.884568e+06           Rich
197  2.994160e+06  Above Average

[198 rows x 7 columns]
```

[ ]:
```
# prompt: draw chart for K Means Clustering result



# Assuming your dataframe is named 'df' and has columns 'Annual Income (k$)'
 ↪and 'Spending Score (1-100)'
# and 'centroid' for cluster assignment

# Create a color map for different clusters
colors = ['red', 'green', 'blue', 'cyan', 'magenta', 'yellow', 'black',
 ↪'purple', 'pink', 'orange']

# Plot the data points with different colors for each cluster
plt.figure(figsize=(10, 6))
for i in range(len(df['centroid'].unique())):
    cluster_data = df[df['centroid'] == i]
    plt.scatter(cluster_data['Annual Income (k$)'], cluster_data['Spending
 ↪Score (1-100)'],
                color=colors[i % len(colors)], label='Cluster {}'.format(i))

# Plot the centroids
plt.scatter(centroids['Annual Income (k$)'], centroids['Spending Score
 ↪(1-100)'],
            color='black', marker='X', s=200, label='Centroids')

# Set labels and title
legend1 = ["Rich","Poor","Below Average","Above Average","Average"]
# Add labels and title
```

```
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('K-Means Clustering Result')
plt.legend(legend1, loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.show()
```



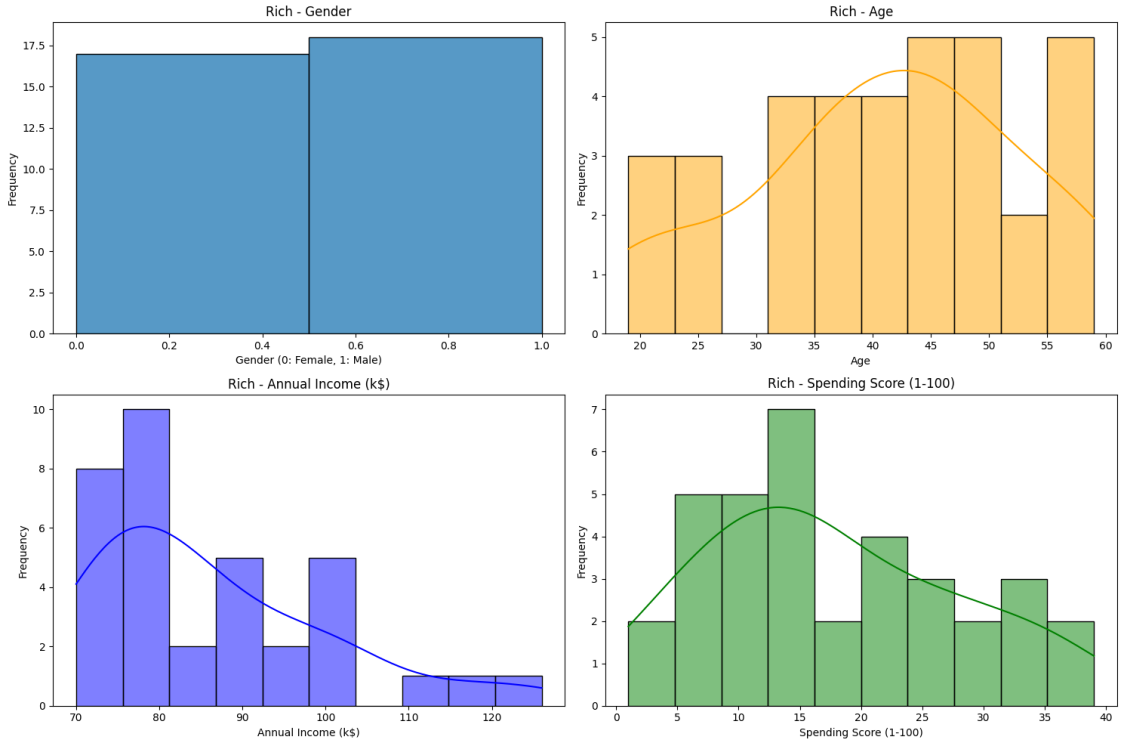
[ ]:
```
# Lọc các nhãn cần thiết
filtered_df = df[df['centroid_label'].isin(['Rich', 'Poor', 'Below Average',␣
 ↪'Above Average', 'Average'])]

# In ra DataFrame đã lọc
print(filtered_df)

# Đếm số lượng dòng cho mỗi nhãn
label_counts = filtered_df['centroid_label'].value_counts()
print(label_counts)
```

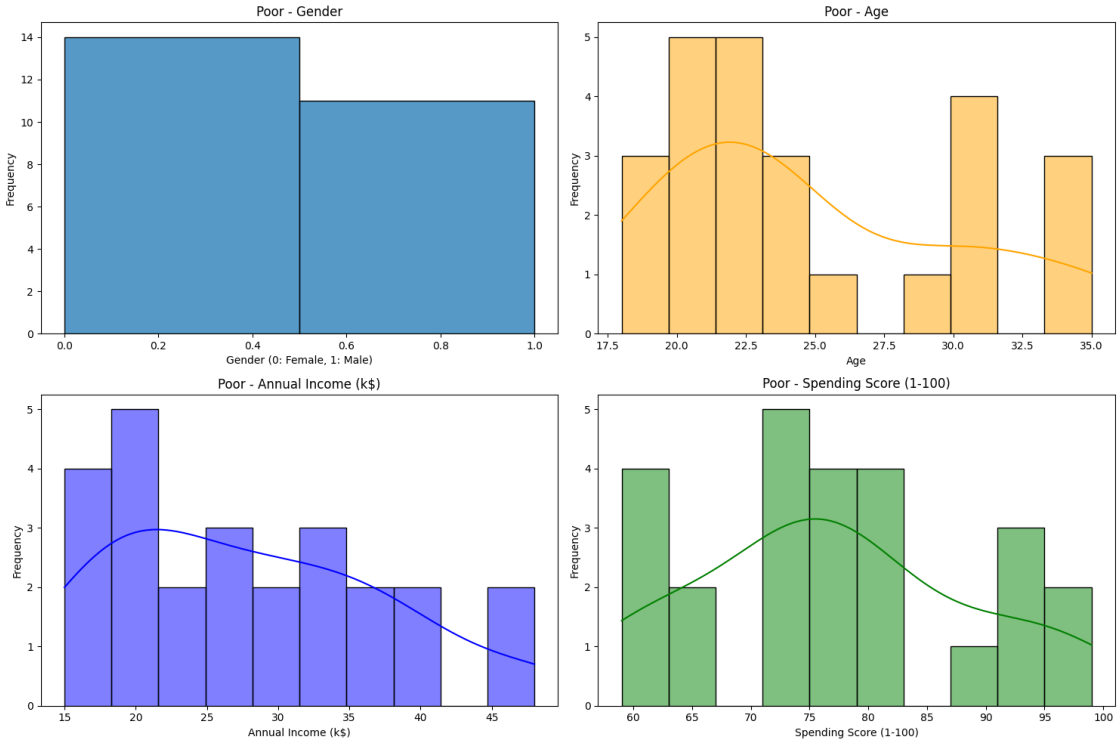
	Gender	Age	Annual Income (k$)	Spending Score (1-100)	centroid	\
0	1	19	15	39	2	
1	1	21	15	81	0	
2	0	20	16	6	2	
3	0	23	16	77	0	
4	0	31	17	40	2	
..	…	…	…	…	…	
193	0	38	113	91	3	
194	0	47	120	16	4	

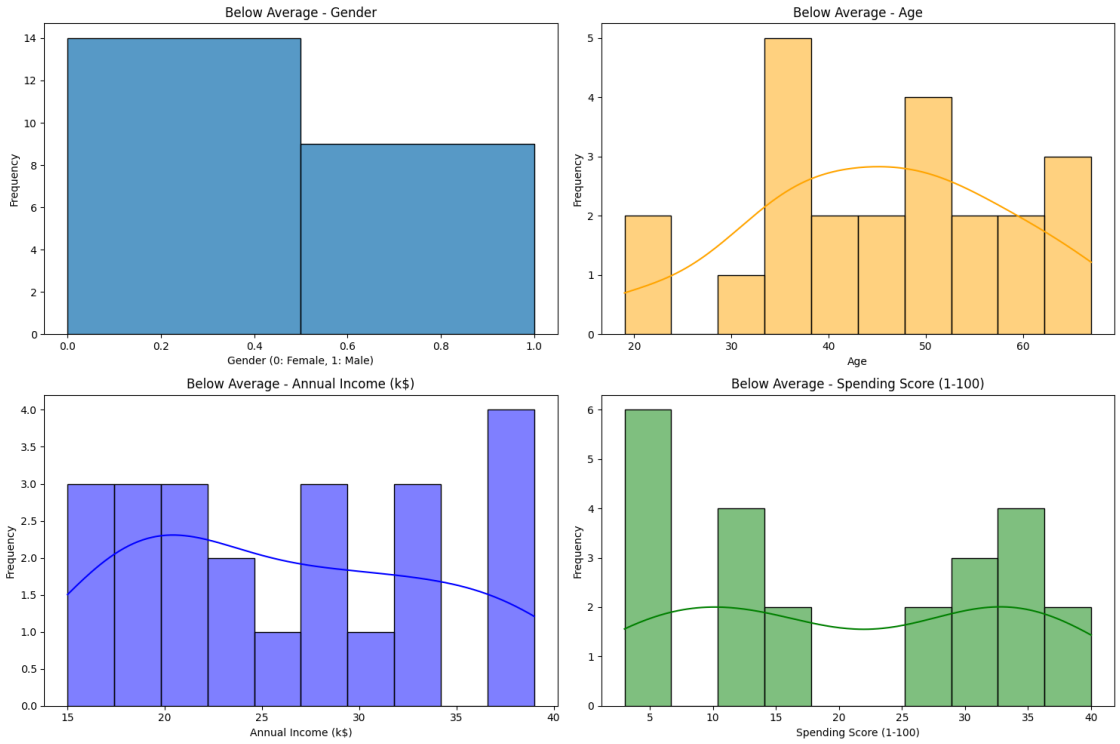
```
195       0   35                 120                      79          3
196       0   45                 126                      28          4
197       1   32                 126                      74          3

             error centroid_label
0    1.305645e+06  Below Average
1    4.081887e+04           Poor
2    9.305465e+05  Below Average
3    2.220005e+04           Poor
4    4.266312e+05  Below Average
..            …             …
193  7.726337e+05  Above Average
194  1.374928e+06           Rich
195  1.501002e+06  Above Average
196  2.884568e+06           Rich
197  2.994160e+06  Above Average

[198 rows x 7 columns]
centroid_label
Average          77
Above Average    38
Rich             35
Poor             25
Below Average    23
Name: count, dtype: int64
```

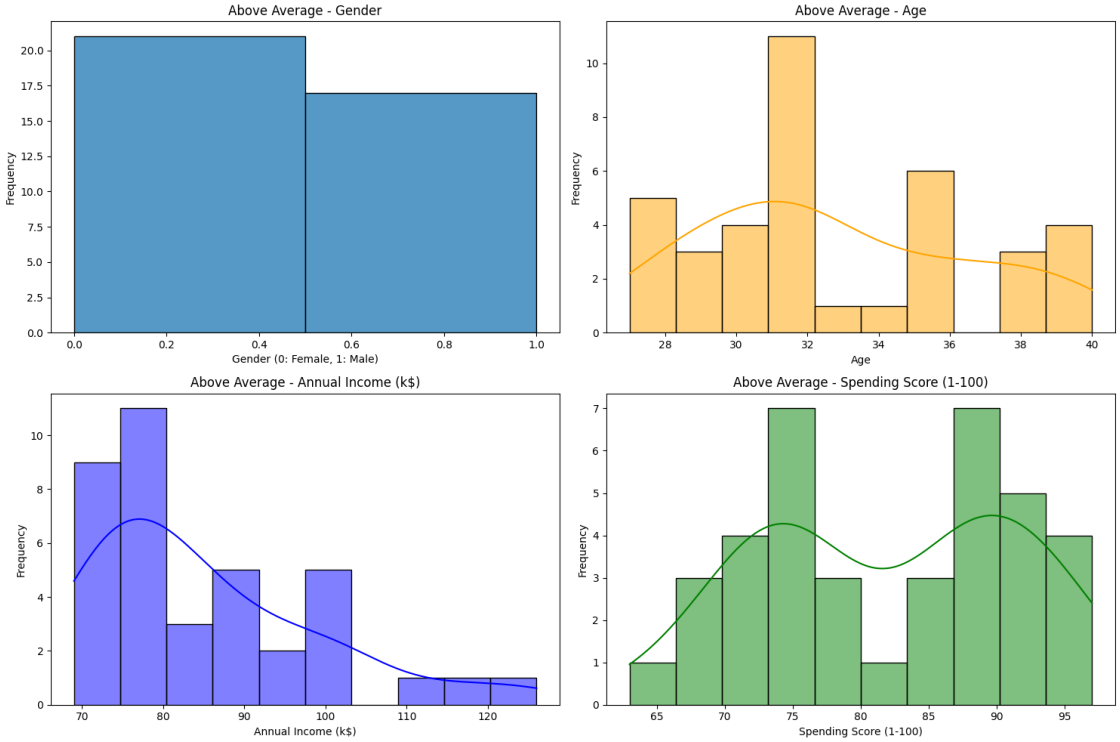
[ ]:
```
warnings.filterwarnings('ignore')
init_notebook_mode(connected=True)
# Vẽ biểu đồ phân phối cho mỗi nhãn
labels = ['Rich', 'Poor', 'Below Average', 'Above Average', 'Average']

for label in labels:
    subset = filtered_df[filtered_df['centroid_label'] == label]

    plt.figure(figsize=(15, 10))

    plt.subplot(2, 2, 1)
    sns.histplot(subset['Gender'].astype('category').cat.codes, kde=False,␣
 ↪bins=2)
    plt.title(f'{label} - Gender')
    plt.xlabel('Gender (0: Female, 1: Male)')
    plt.ylabel('Frequency')

    plt.subplot(2, 2, 2)
    sns.histplot(subset['Age'], kde=True, bins=10, color='orange')
    plt.title(f'{label} - Age')
    plt.xlabel('Age')
```

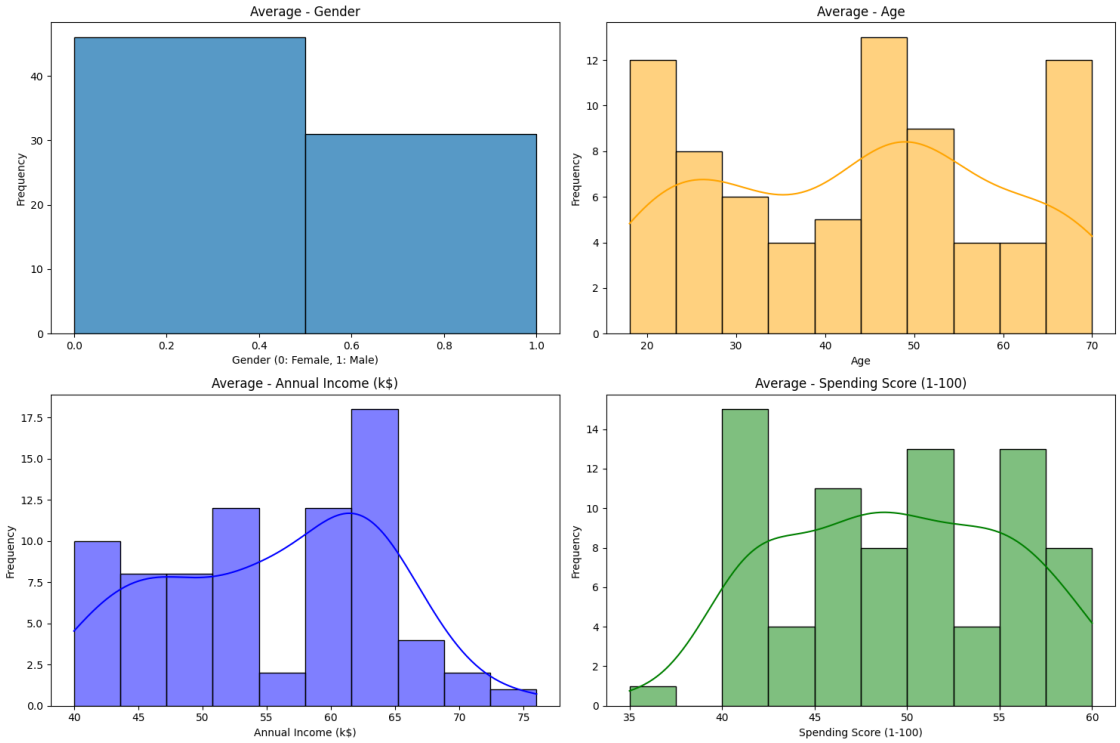
```
    plt.ylabel('Frequency')

    plt.subplot(2, 2, 3)
    sns.histplot(subset['Annual Income (k$)'], kde=True, bins=10, color='blue')
    plt.title(f'{label} - Annual Income (k$)')
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Frequency')

    plt.subplot(2, 2, 4)
    sns.histplot(subset['Spending Score (1-100)'], kde=True, bins=10,␣
↪color='green')
    plt.title(f'{label} - Spending Score (1-100)')
    plt.xlabel('Spending Score (1-100)')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()
```




Poor - Gender
Frequency
Gender (0: Female, 1: Male)
Poor - Age
Frequency
Age
Poor - Annual Income (k$)
Frequency
Annual Income (k$)
Poor - Spending Score (1-100)
Frequency
Spending Score (1-100)


Below Average - Gender
Frequency
Gender (0: Female, 1: Male)
Below Average - Age
Frequency
Age
Below Average - Annual Income (k$)
Frequency
Annual Income (k$)
Below Average - Spending Score (1-100)
Frequency
Spending Score (1-100)


Above Average - Gender
Frequency
Gender (0: Female, 1: Male)
Above Average - Age
Frequency
Age
Above Average - Annual Income (k$)
Frequency
Annual Income (k$)
Above Average - Spending Score (1-100)
Frequency
Spending Score (1-100)


Average - Gender
Frequency
Gender (0: Female, 1: Male)
Average - Age
Frequency
Age
Average - Annual Income (k$)
Frequency
Annual Income (k$)
Average - Spending Score (1-100)
Frequency
Spending Score (1-100)

[ ]:
```
tmp_df = df[['Gender','Age','Annual Income (k$)','Spending Score␣
↪(1-100)','centroid_label']]
tmp_df.to_csv('Mall_Customers_Labeled.csv', index=False)
```

knn-from-scratch

August 12, 2024

[ ]:
```
from math import sqrt
from random import seed
from random import randrange
from csv import reader
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

[ ]:
```
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
        neighbors = get_neighbors(train, test_row, num_neighbors)
        output_values = [row[-1] for row in neighbors]
        prediction = max(set(output_values), key=output_values.count)
        return prediction
```

[ ]:
```
# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
        distance = 0.0
        for i in range(len(row1)-1):
                distance += (row1[i] - row2[i])**2
        return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
        distances = list()
        for train_row in train:
                dist = euclidean_distance(test_row, train_row)
                distances.append((train_row, dist))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
                neighbors.append(distances[i][0])
        return neighbors
```

```
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
        neighbors = get_neighbors(train, test_row, num_neighbors)
        output_values = [row[-1] for row in neighbors]
        prediction = max(set(output_values), key=output_values.count)
        return prediction

# Test distance function
dataset = [[2.7810836,2.550537003,0],
        [1.465489372,2.362125076,0],
        [3.396561688,4.400293529,0],
        [1.38807019,1.850220317,0],
        [3.06407232,3.005305973,0],
        [7.627531214,2.759262235,1],
        [5.332441248,2.088626775,1],
        [6.922596716,1.77106367,1],
        [8.675418651,-0.242068655,1],
        [7.673756466,3.508563011,1]]
prediction = predict_classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

```
Expected 0, Got 0.
```

[ ]:
```
# Load a CSV file
def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
        class_values = [row[column] for row in dataset]
        unique = set(class_values)
        lookup = dict()
        for i, value in enumerate(unique):
                lookup[value] = i
```

```
        for row in dataset:
                row[column] = lookup[row[column]]
        return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
        minmax = list()
        for i in range(len(dataset[0])):
                col_values = [row[i] for row in dataset]
                value_min = min(col_values)
                value_max = max(col_values)
                minmax.append([value_min, value_max])
        return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
        for row in dataset:
                for i in range(len(row)):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -␣
 ↪minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for _ in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
        folds = cross_validation_split(dataset, n_folds)
        scores = list()
        for fold in folds:
```

```
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
                predicted = algorithm(train_set, test_set, *args)
                # print(predicted)
                actual = [row[-1] for row in fold]
                accuracy = accuracy_metric(actual, predicted)
                scores.append(accuracy)
        return scores


# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
        predictions = list()
        for row in test:
                output = predict_classification(train, row, num_neighbors)
                predictions.append(output)
        return(predictions)
```

[ ]:
```
# Test the kNN on the Iris Flowers dataset
seed(1)
filename = '/content/Mall_Customers_Labeled.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 10
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds,␣
 ↪num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [94.73684210526315, 94.73684210526315, 100.0, 100.0, 100.0,
94.73684210526315, 94.73684210526315, 84.21052631578947, 94.73684210526315,
89.47368421052632]
Mean Accuracy: 94.737%
```

[ ]:
```
# prompt: help me to build a classifier base on the K-nearest-neighbors␣
 ↪algorithm, and write code to test the classifier.
```

```
# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
	for row in dataset:
		row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
	class_values = [row[column] for row in dataset]
	unique = set(class_values)
	lookup = dict()
	for i, value in enumerate(unique):
		lookup[value] = i
	for row in dataset:
		row[column] = lookup[row[column]]
	return lookup

# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
	distance = 0.0
	for i in range(len(row1)-1):
		distance += (row1[i] - row2[i])**2
	return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
	distances = list()
	for train_row in train:
		dist = euclidean_distance(test_row, train_row)
		distances.append((train_row, dist))
	distances.sort(key=lambda tup: tup[1])
	neighbors = list()
	for i in range(num_neighbors):
		neighbors.append(distances[i][0])
	return neighbors
```

```
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row, num_neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction

class KNN:
  def __init__(self, num_neighbors):
    self.num_neighbors = num_neighbors

  def fit(self, X, y):
    self.X_train = X
    self.y_train = y

  def predict(self, X_test):
    predictions = []
    for row in X_test:
      output = predict_classification(self.X_train, row, self.num_neighbors)
      predictions.append(output)
    return(predictions)
```

[ ]:
```
# Test the kNN on the Mall_Customers dataset
seed(1)
filename = '/content/Mall_Customers_Labeled.csv' # Assuming you have the␣
 ↪Mall_Customers dataset in a file named 'Mall_Customers_Labeled.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
# convert class column to integers
#str_column_to_int(dataset, len(dataset[0])-2)

# Split the dataset into train and test sets (example using 80/20 split)
train_size = int(0.8 * len(dataset))
train_set = dataset[:train_size]
test_set = dataset[train_size:]

# Prepare data for the KNN classifier
X_train = [row[:] for row in train_set]
y_train = [row[-1] for row in train_set]
X_test = [row[:] for row in test_set]
y_test = [row[-1] for row in test_set]

# Create and fit the KNN classifier
knn = KNN(num_neighbors=4)
knn.fit(X_train, y_train)
```

```python
# Make predictions on the test set
y_pred = knn.predict(X_test)

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
                        print("Actual: " + str(actual[i]) + ' vs Predicted:' +␣
 ↪str(predicted[i]))
        return correct / float(len(actual)) * 100.0

# Evaluate the model (example using accuracy)
accuracy = accuracy_metric(y_test, y_pred)
print('Accuracy: %.3f%%' % accuracy)
```

```
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
```

```
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Actual: Rich vs Predicted:Rich
Actual: Above Average vs Predicted:Above Average
Accuracy: 97.500%
```

[ ]:
```python
# Create a new data point
new_data = [1, 35, 20, 20]  # Example values for the new data point

# Make a prediction for the new data point
prediction = knn.predict([new_data])

# Print the prediction
print('Prediction for new data:', prediction)
```

```
Prediction for new data: ['Below Average']
```

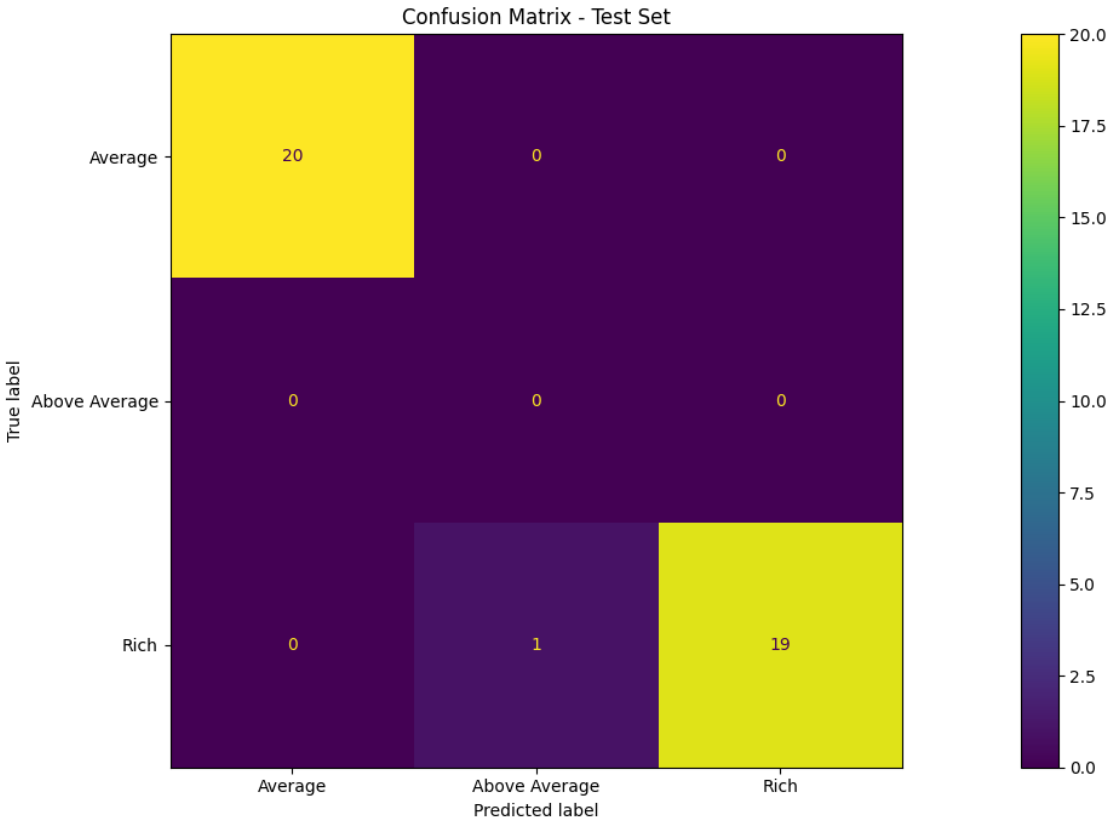
[ ]:
```python
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=1)

# Print the evaluation metrics
print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
```

```
Accuracy: 0.975
Precision: 1.0
Recall: 0.975
F1-score: 0.9871794871794872
```

[ ]:
```python
train_predictions = knn.predict(X_train)
test_predictions = knn.predict(X_test)

train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

print(f"Accuracy on training data: {train_accuracy}")
print(f"Accuracy on test data: {test_accuracy}")
```

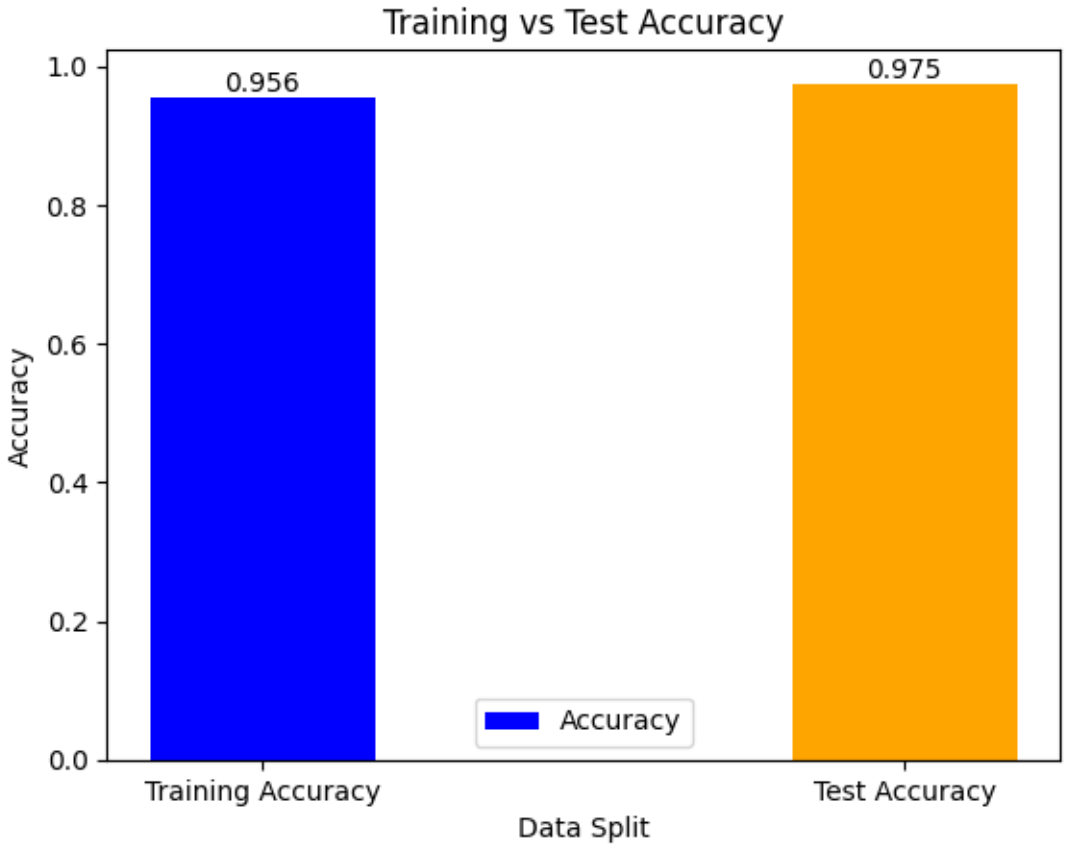
```
Accuracy on training data: 0.9556962025316456
Accuracy on test data: 0.975
```

[ ]:
```python
# Define the label mapping
labels = {0: 'Poor', 1: 'Average', 2: 'Below Average', 3: 'Above Average', 4:␣
 ↪'Rich'}

# Find unique labels in the test predictions
unique_labels_test = set(y_test).union(set(test_predictions))

# Convert unique labels to lowercase for consistency
unique_labels_test = {label.lower() for label in unique_labels_test}

# Get the label names for the unique classes, handling potential missing labels
label_names_test = [labels.get(i, 'Unknown') for i in range(5) if labels.get(i,␣
 ↪'').lower() in unique_labels_test]

# Calculate the confusion matrix for the test set
test_conf_matrix = confusion_matrix(y_test, test_predictions)

# Plot the confusion matrix for the test set
fig, ax = plt.subplots(figsize=(20, 7))
ConfusionMatrixDisplay(confusion_matrix=test_conf_matrix,␣
 ↪display_labels=label_names_test).plot(ax=ax)
ax.set_title("Confusion Matrix - Test Set")

plt.tight_layout()
plt.show()
```



[ ]:
```
def plot_accuracy(train_accuracy, test_accuracy):
    labels = ['Training Accuracy', 'Test Accuracy']
    accuracies = [train_accuracy, test_accuracy]

    fig, ax = plt.subplots()
    bar_width = 0.35
    index = range(len(labels))

    bars = ax.bar(index, accuracies, bar_width, color=['blue', 'orange'],␣
 ↪label='Accuracy')

    ax.set_xlabel('Data Split')
    ax.set_ylabel('Accuracy')
    ax.set_title('Training vs Test Accuracy')
    ax.set_xticks(index)
    ax.set_xticklabels(labels)
    ax.legend()

    for bar in bars:
        height = bar.get_height()
```

```
        ax.text(bar.get_x() + bar.get_width() / 2.0, height, f'{height:.3f}',␣
 ↪ha='center', va='bottom')

    plt.show()
plot_accuracy(train_accuracy, test_accuracy)
```



[ ]:
```
# evaluate knn performance on train and test sets with different numbers of␣
 ↪neighbors

# create dataset
#X, y = make_classification(n_samples=10000, n_features=20, n_informative=5,␣
 ↪n_redundant=15, random_state=1)
# split into train test sets
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# define lists to collect scores
train_scores, test_scores = list(), list()


# define the tree depths to evaluate
```

```
values = [i for i in range(1, 40)]
# evaluate a decision tree for each depth
for i in values:
 # configure the model
 model = KNN(num_neighbors=i)
 # fit model on the training dataset
 model.fit(X_train, y_train)
 # evaluate on the train dataset
 train_yhat = model.predict(X_train)
 train_acc = accuracy_score(y_train, train_yhat)
 train_scores.append(train_acc)
 # evaluate on the test dataset
 test_yhat = model.predict(X_test)
 test_acc = accuracy_score(y_test, test_yhat)
 test_scores.append(test_acc)
 # summarize progress
 print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
# plot of train and test scores vs number of neighbors
plt.plot(values, train_scores, '-o', label='Train')
plt.plot(values, test_scores, '-o', label='Test')
plt.legend()
plt.show()
```

```
>1, train: 1.000, test: 1.000
>2, train: 0.968, test: 1.000
>3, train: 0.981, test: 0.975
>4, train: 0.956, test: 0.975
>5, train: 0.962, test: 0.975
>6, train: 0.956, test: 0.975
>7, train: 0.956, test: 0.975
>8, train: 0.949, test: 0.975
>9, train: 0.949, test: 0.975
>10, train: 0.949, test: 0.975
>11, train: 0.956, test: 0.975
>12, train: 0.949, test: 0.975
>13, train: 0.949, test: 0.975
>14, train: 0.949, test: 0.975
>15, train: 0.949, test: 0.975
>16, train: 0.949, test: 0.975
>17, train: 0.956, test: 0.950
>18, train: 0.949, test: 0.950
>19, train: 0.949, test: 0.950
>20, train: 0.949, test: 0.950
>21, train: 0.949, test: 0.950
>22, train: 0.949, test: 0.950
>23, train: 0.949, test: 0.950
>24, train: 0.937, test: 0.950
```

```
>25, train: 0.943, test: 0.950
>26, train: 0.943, test: 0.950
>27, train: 0.943, test: 0.925
>28, train: 0.937, test: 0.925
>29, train: 0.937, test: 0.875
>30, train: 0.937, test: 0.900
>31, train: 0.905, test: 0.525
>32, train: 0.892, test: 0.525
>33, train: 0.880, test: 0.500
>34, train: 0.880, test: 0.500
>35, train: 0.880, test: 0.500
>36, train: 0.861, test: 0.475
>37, train: 0.804, test: 0.275
>38, train: 0.791, test: 0.150
>39, train: 0.785, test: 0.025
```