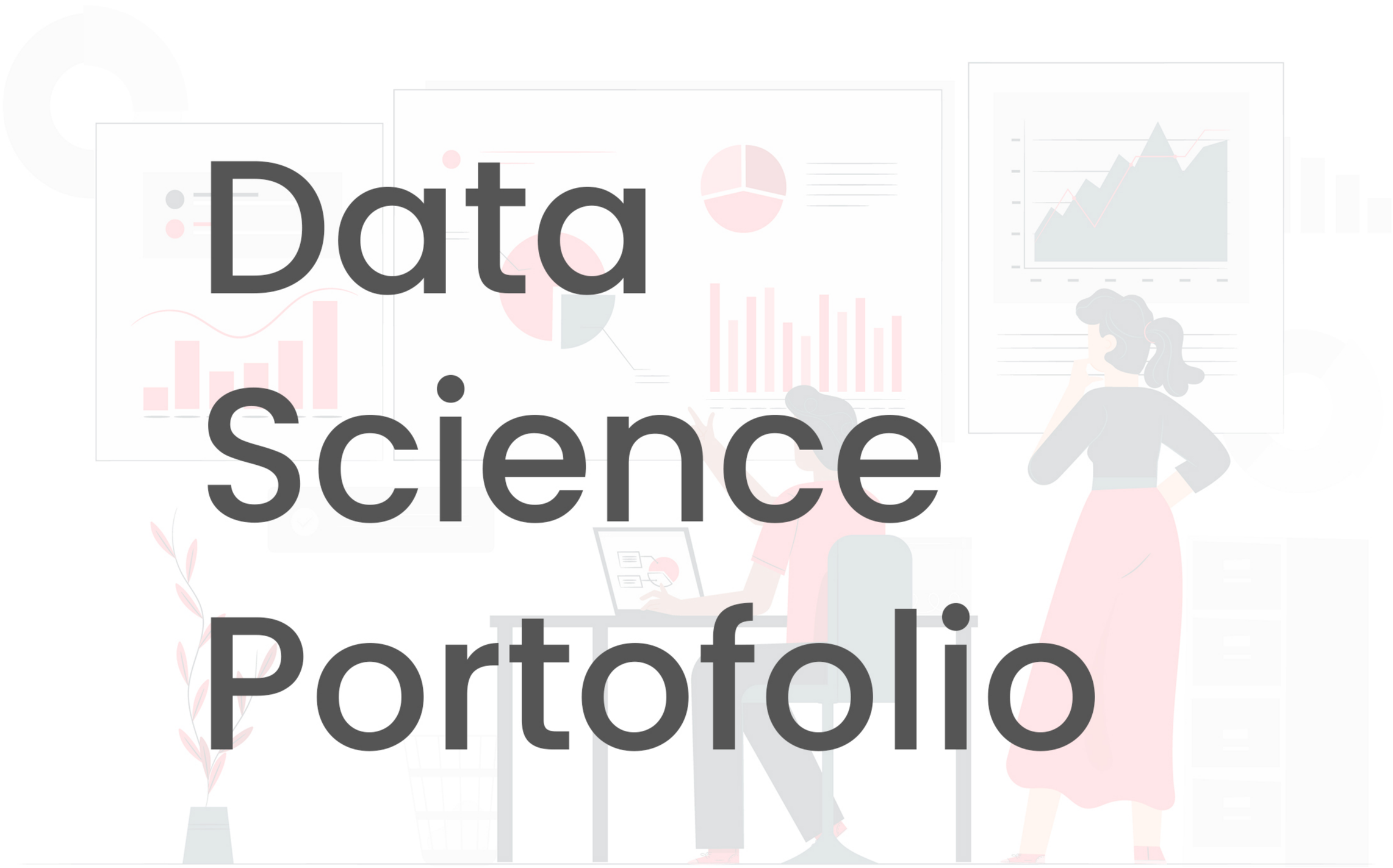


Data Science Portofolio

Mohamad Adhikasurya Haidar
Universitas Gadjah Mada
+62 82115529411
mohamadadhikasurya2019@mail.ugm.ac.id

Hello everyone, in this portofolio I would like to present some of my study from “Machine Learning Projects” books written by Muhammad Ardi. There are 2 machine learning project that i’d like to show which consist of :

· **Titanic Survival Analysis**

· **Customer Segmentation Using K-Means**

Titanic Survival Analysis

In this Analysis , i’m trying to make a model that can predict if any new passenger would survive or not. But first, let’s do an Exploratory Data Analysis over the data.

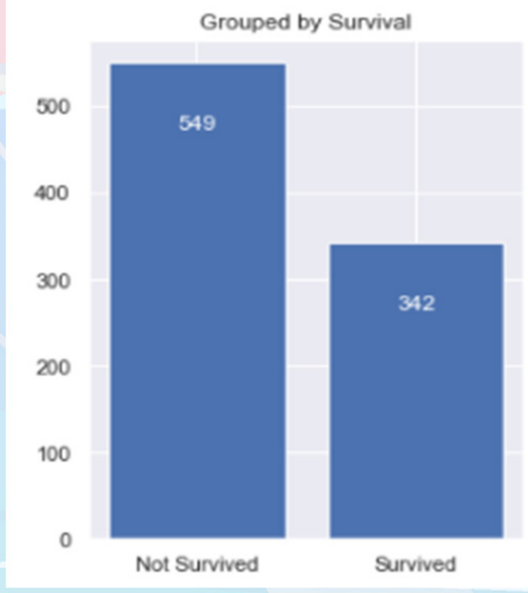
1. Let’s check if there’s any null values

```
In [8]: 1 df.isnull().sum()

Out[8]: PassengerId      0
        Survived         0
        Pclass           0
        Name             0
        Sex              0
        Age            177
        SibSp            0
        Parch            0
        Ticket           0
        Fare             0
        Cabin          687
        Embarked         2
        dtype: int64
```

By using *.isnull().sum()* we can check the total number of null values in each of feature we have in the dataframe. As we can see, there’s a lot of null values in Age and Cabin values. We may have to replace those values for modelling purposes. On the other hand, since there’re only 2 null values in Embarked feature, we may just drop the rows which consist those null values in Embarked feature. We’ll proccess these null values in Data Preparation for modelling.

2. Make a bar plot to see the total of passengers that survived and doesn’t.

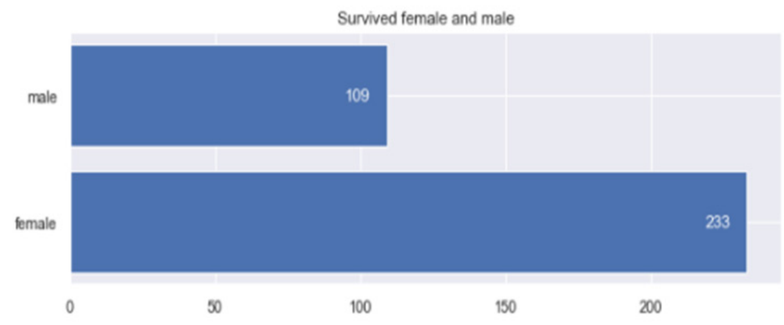
```
survived_count = df['Survived'].value_counts()
survived_count
plt.figure(figsize=(4,5))
plt.bar(survived_count.index,survived_count.values)
plt.title('Grouped by Survival')
plt.xticks([0,1],['Not Survived','Survived'])


for i, value in enumerate(survived_count.values):
    plt.text(i, value-70,s = str(value), fontsize=12, color='white',
             horizontalalignment='center', verticalalignment='center')
plt.show()
```



We can tell from the bar plot that there are less people who survived than those who doesn’t.

3. Passengers survival based on genders

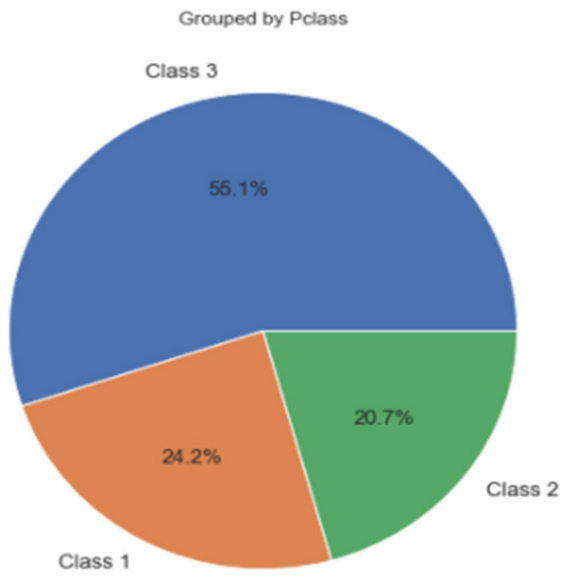
```
In [13]: 1 survived_sex = df[df['Survived']==1]['Sex'].value_counts()
         2
         3 plt.figure(figsize=(10,3))
         4 plt.barh(survived_sex.index,survived_sex.values)
         5 plt.title('Survived female and male')
         6
         7 for i, value in enumerate(survived_sex.values):
         8     plt.text(value-10, i,s = str(value), fontsize=12, color='white',
         9              horizontalalignment='center', verticalalignment='center')
        10 plt.show()
```



We can tell from graph that there are more female survivors than man. It’s kinda obvious because women and childern safety is top priority.

4. Ticket class distribution

```
pclass_count = df['Pclass'].value_counts()
pclass_count
plt.figure(figsize=(7,7))
plt.title('Grouped by Pclass')
plt.pie(pclass_count.values,
        labels = ['Class {}'.format(i) for i in pclass_count.index],
        autopct='%1.1f%%', textprops={'fontsize':13})
plt.show()
```



From the pie plot above, we can tell that most of the passengers in titanic ship bought class 3 tickets. It could be because people want to save their budget, or simply couldn't afford higher class ticket.

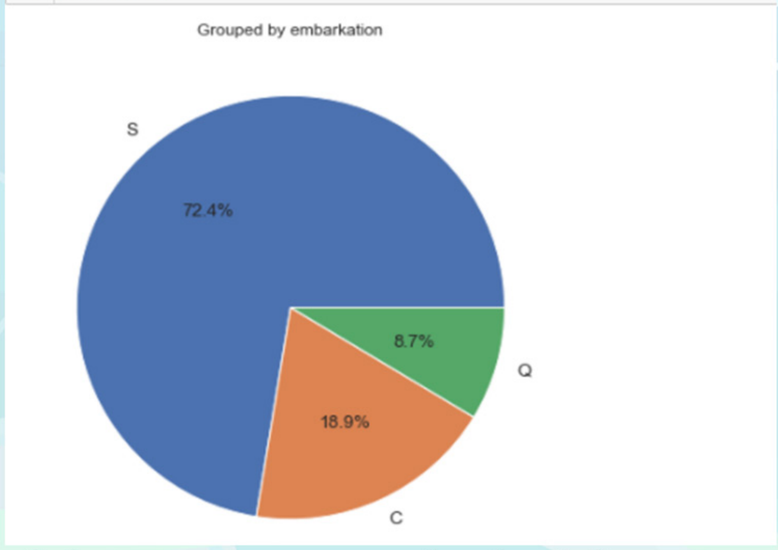
5. Passengers distribution by gender

```
In [16]:
sex_count = df['Sex'].value_counts()
plt.figure(figsize=(7,7))
plt.title('Grouped by Sex')
plt.pie(sex_count.values,
        labels = ['{}'.format(i) for i in sex_count.index],
        autopct='%1.1f%%', textprops={'fontsize':13})
plt.show()
```


There are more male passangers than female passengers. Probably most of the female passengers are wifes or daughter

6. Passangers distribution by embarkation point

```
embarked_count = df['Embarked'].value_counts()

plt.figure(figsize=(7,7))
plt.title('Grouped by embarkation')
plt.pie(embarked_count.values,
        labels = ['{}'.format(i) for i in embarked_count.index],
        autopct='%1.1f%%', textprops={'fontsize':13})
plt.show()
```

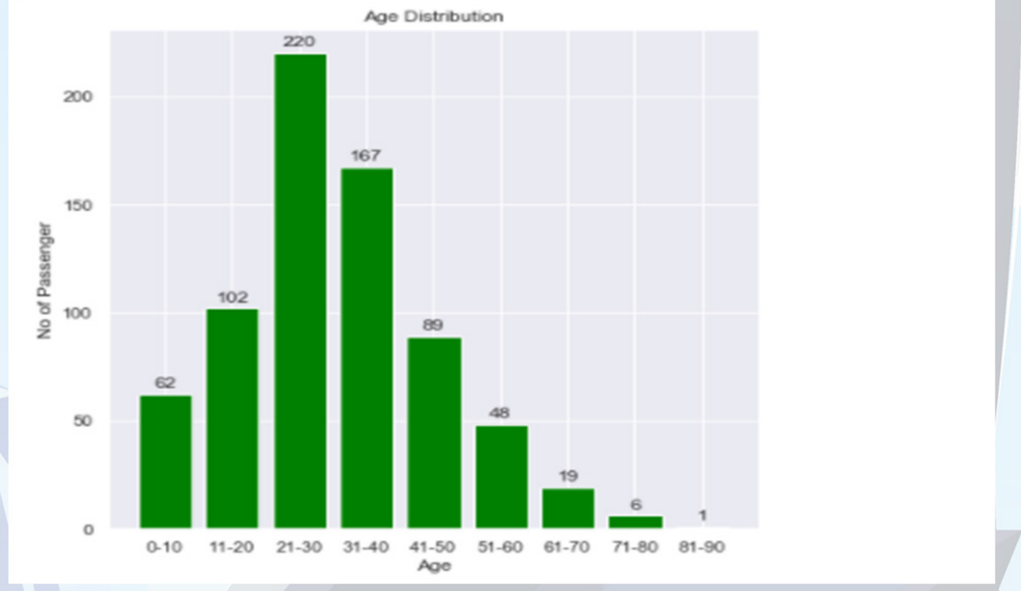


Most of the passengers embarked from southampton.

7. Passengers distribution by Age

Since there exist a null value, we will only plot the non-null value data. Histogram will be used to show the age distribution.

```
ages = df[df['Age'].notnull()]['Age'].values

ages_hist = np.histogram(ages, bins=[0,10,20,30,40,50,60,70,80,90])
ages_hist_label = ['0-10','11-20','21-30',
                   '31-40','41-50','51-60',
                   '61-70','71-80','81-90']

plt.figure(figsize=(7,7))
plt.title('Age Distribution')
plt.bar(ages_hist_label, ages_hist[0], color='Green')
plt.xlabel('Age')
plt.ylabel('No of Passenger')
for i, bin in zip(ages_hist[0], range(9)) :
    plt.text(bin, i+5, str(int(i)), fontsize=12,
             horizontalalignment='center', verticalalignment='center')
plt.show()
```



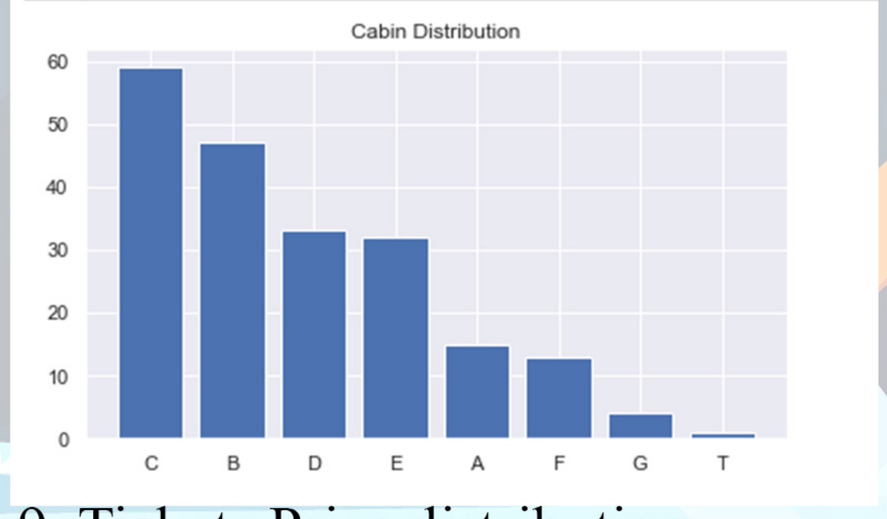
We can infer that most of the passengers are from ages 20s to 40s.

8. Cabins distribution

Just like the age distribution, since there exist a null value in cabin's feature, we need to use only the non-null value. Also, we're going to distribute the cabin's feature only by its initial name.

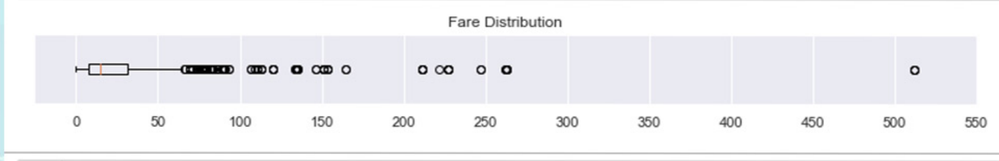
```
cabins = df['Cabin'].dropna()
def take_initial(x):
    return x[0]
cabins = cabins.apply(take_initial)
```

```
cabins_count = cabins.value_counts()

plt.figure(figsize=(7,4))
plt.title("Cabin Distribution")
plt.bar(cabins_count.index, cabins_count.values)
plt.show()
```



9. Tickets Price distribution

```
plt.figure(figsize=(13,1))
plt.title("Fare Distribution")
plt.boxplot(df["Fare"], vert=False)
plt.xticks(range(0,600,50))
plt.yticks([])
plt.show()
```




```
df['Fare'].describe()
```

```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```

Now that we've finished doing the ETL, we're going to do some feature engineering to make our data is ready to use for modelling using the Logistic Regression Algorithm.

Feature Engineering

1. SibSp and Parch Feature

We're going to combine the SibSp + Parch + 1(the passenger itself) as new column "family size" to measure the family size of each passengers. This feature would be usefull to be included in later modelling.

```
df["SibSp"].unique()
```

```
array([1, 0, 3, 4, 2, 5, 8], dtype=int64)
```

```
df["FamilySize"] = df['SibSp'] + df['Parch'] + 1
df[['Name', 'SibSp', 'Parch','FamilySize']]
```

	Name	SibSp	Parch	FamilySize
0	Braund, Mr. Owen Harris	1	0	2
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	0	2
2	Heikkinen, Miss. Laina	0	0	1
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0	2
4	Allen, Mr. William Henry	0	0	1
...	...	...	...	...
886	Montvila, Rev. Juozas	0	0	1
887	Graham, Miss. Margaret Edith	0	0	1
888	Johnston, Miss. Catherine Helen "Carrie"	1	2	4
889	Behr, Mr. Karl Howell	0	0	1
890	Dooley, Mr. Patrick	0	0	1

891 rows × 4 columns

2. Embarked feature

Since there only 2 missing data in this feature, we may drop the rows that include this missing data. The type of data of this features is categorical data, therefore we need to change it to a numerical value using one-hot encoding method.

```
print("Sebelum drop :", df.shape)
df = df.dropna(subset=['Embarked'])
print("Setelah drop :", df.shape)
```

```
Sebelum drop : (891, 13)
Setelah drop : (889, 13)
```

```
embarked_one_hot = pd.get_dummies(df['Embarked'], prefix='Embarked')
df = pd.concat([df, embarked_one_hot], axis=1)
```

3. Cabins feature

Since there're too many missing value in this feature, we can't just simply drop the rows because we're going to lose too many data. And it's also not good to fill the missing value with the most used cabins, so we will just fill the missing value as unknown 'U'. Because the data type is categorical data, we have to do a one-hot encoding to change the data type into numerical value.

```
df['Cabin'] = df['Cabin'].fillna('U')
```

```
def take_initial(x):
    return x[0]
df['Cabin'] = df['Cabin'].apply(take_initial)
```

```
cabin_one_hot = pd.get_dummies(df['Cabin'], prefix='Cabin')
df = pd.concat([df, cabin_one_hot], axis=1)

#Hapus kolom cabin yang aslinya
df = df.drop('Cabin', axis=1)

#Tampilkan nama tiap kolom
df.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked', 'Cabin_A', 'Cabin_B', 'Cabin_C',
       'Cabin_D', 'Cabin_E', 'Cabin_F', 'Cabin_G', 'Cabin_T', 'Cabin_U'],
      dtype='object')
```

4. Name feature

We can make a categorical feature out of Name feature. It's that we can categorize the passengers name by its title, and then we make it as a numerical data types by one-hot encoding.

```
def get_title(x):
    return x.split(',')[1].split('.')[0].strip()

df['Title'] = df['Name'].apply(get_title)
```

```
df['Title'].unique()
```

```
array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
       'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'the Countess',
       'Jonkheer'], dtype=object)
```

```
title_one_hot = pd.get_dummies(df['Title'], prefix='Title')
df = pd.concat([df, title_one_hot], axis=1)

# Tampilkan semua kolom di dataframe
df.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'FamilySize', 'Embarked_C', 'Embarked_Q',
       'Embarked_S', 'Cabin_A', 'Cabin_B', 'Cabin_C', 'Cabin_D', 'Cabin_E',
       'Cabin_F', 'Cabin_G', 'Cabin_T', 'Cabin_U', 'Title', 'Title_Capt',
       'Title_Col', 'Title_Don', 'Title_Dr', 'Title_Jonkheer', 'Title_Lady',
       'Title_Major', 'Title_Master', 'Title_Miss', 'Title_Mlle', 'Title_Mme',
       'Title_Mr', 'Title_Mrs', 'Title_Ms', 'Title_Rev', 'Title_Sir',
       'Title_the Countess'],
      dtype='object')
```

5. Sex feature

In this feature, we obviously just need to simply change the data types from categorical to numerical value through one-hot-encoding

```
sex_one_hot = pd.get_dummies(df['Sex'], prefix='Sex')
df = pd.concat([df, sex_one_hot], axis=1)

# Hapus kolom sex yang aslinya
df = df.drop('Sex', axis=1)

df.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'FamilySize', 'Embarked_C', 'Embarked_Q',
       'Embarked_S', 'Cabin_A', 'Cabin_B', 'Cabin_C', 'Cabin_D', 'Cabin_E',
       'Cabin_F', 'Cabin_G', 'Cabin_T', 'Cabin_U', 'Title', 'Title_Capt',
       'Title_Col', 'Title_Don', 'Title_Dr', 'Title_Jonkheer', 'Title_Lady',
       'Title_Major', 'Title_Master', 'Title_Miss', 'Title_Mlle', 'Title_Mme',
       'Title_Mr', 'Title_Mrs', 'Title_Ms', 'Title_Rev', 'Title_Sir',
       'Title_the Countess', 'Sex_female', 'Sex_male'],
      dtype='object')
```

6. Age feature

We've noticed that there are lots of missing value in this feature. We're going to fill the missing value by the mean of ages of each titles just like in the following code :

```
age_median = df.groupby('Title')['Age'].median()
age_median
```

```
Title
Capt            70.0
Col             58.0
Don             40.0
Dr              46.5
Jonkheer        38.0
Lady            48.0
Major           48.5
Master           3.5
Miss            21.0
Mlle            24.0
Mme             24.0
Mr              30.0
Mrs             35.0
Ms              28.0
Rev             46.5
Sir             49.0
the Countess    33.0
Name: Age, dtype: float64
```

```
def fill_age(x) :
    for index, age in zip(age_median.index, age_median.values):
        if x['Title'] == index:
            return age
df['Age'] = df.apply(lambda x : fill_age(x) if np.isnan(x['Age']) else x['Age'], axis=1)
```

7. We're going to drop feature that we're not going to use in the modeling.

```
df = df.drop(['PassengerId', 'Ticket','Title','Name','Cabin','embarked' ], axis=1)
```

Now that the data is ready, we're going to start to make the model using Logistic Regression to classify/predict whether any new passenger will survive. or not.

1. We split the data into train data and test data. We use the train data for training the model, and the test data to evaluate the accuracy of the model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=21, test_size=0.2)

print('X_train.shape\t:', X_train.shape)
print('y_train.shape\t:', y_train.shape)
print('X_test.shape\t:', X_test.shape)
print('y_test.shape\t:', y_test.shape)
```

```
X_train.shape   : (711, 37)
y_train.shape   : (711,)
X_test.shape    : (178, 37)
y_test.shape    : (178,)
```

We use 80% of the data to train the model, and the rest for testing the model.

Train the Model using Logistic Regression Algorithm

1. We split the data into train data and test data. We use the train data for training the model, and the test data to evaluate the accuracy of the model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=21, test_size=0.2)

print('X_train.shape\t:', X_train.shape)
print('y_train.shape\t:', y_train.shape)
print('X_test.shape\t:', X_test.shape)
print('y_test.shape\t:', y_test.shape)
```

```
X_train.shape   : (711, 37)
y_train.shape   : (711,)
X_test.shape    : (178, 37)
y_test.shape    : (178,)
```

We use 80% of the data to train the model, and the rest for testing the model.

2. Train the model using *scikit learn* Library

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

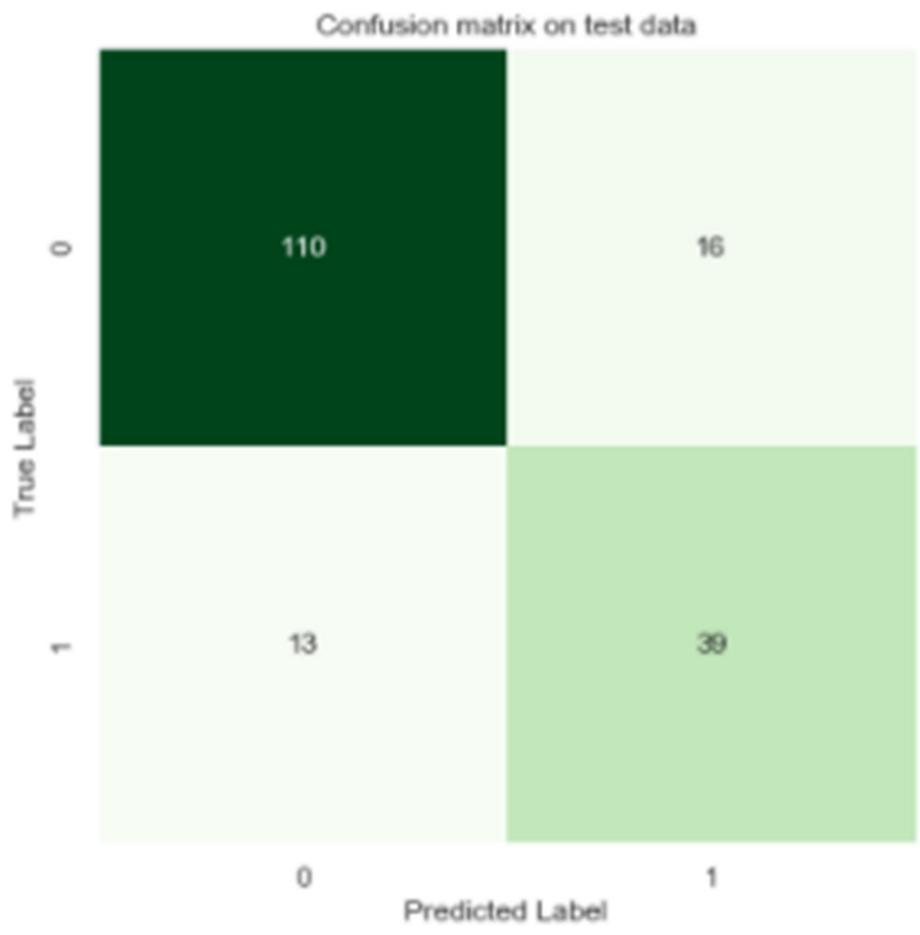
3. Evaluate the model using confusion matrix

```
test_preds = clf.predict(X_test)
```

```
cm = confusion_matrix(y_test, test_preds)
cm
```

```
array([[110,  16],
       [ 13,  39]], dtype=int64)
```

```
plt.figure(figsize=(6,6))
plt.title('Confusion matrix on test data')
sns.heatmap(cm, annot=True, fmt='d', cmap=plt.cm.Greens, cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




```
print("Train accuracy\t:", clf.score(X_train, y_train))
print("Test accuracy\t:", clf.score(X_test, y_test))
```

```
Train accuracy  : 0.8368495077355836
Test accuracy   : 0.8370786516853933
```

As we mentioned before, we use the test set to evaluate our model. First, we use the model to predict the passengers in the test data whether each of them will survive or not. And then we're comparing the predicted result (test_preds) with the true result (y_test) using the confusion matrix.

As we can see, from the confusion matrix we have a 110 true positive, 13 false positive (the predicted says it doesn't survives, but the true result is survives), 16 False positive, and 39 false negative.

Since it has the accuracy of the test data is 83 %, i'd say this model is quite good to use.

Customer Segmentation Using K-Means Algorithm

In this project, we will try to clusterize customers at Mall. The purpose of this clusterization is to help the marketing team for targeting the mall's future customers.

First let's take a look at our data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

df = pd.read_csv("Mall_Customers.csv")
df.head(10)
```

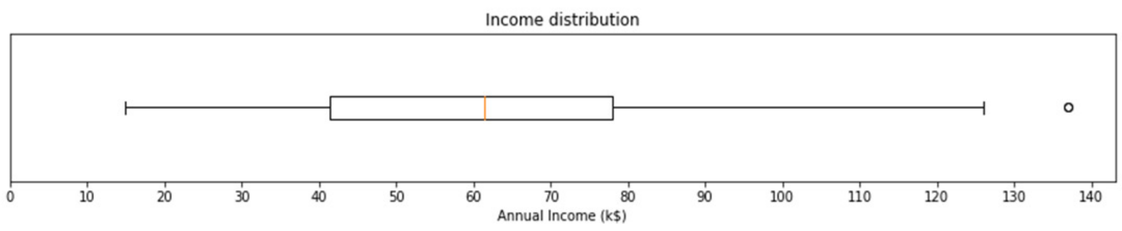
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

Here in the tabel/data we have information about CustomerID, gender, age, Annual income, and spending score of each customer who's spend some of their money at the mall.

To understand more of the data, we're going to do some Exploratory Data Analysis.

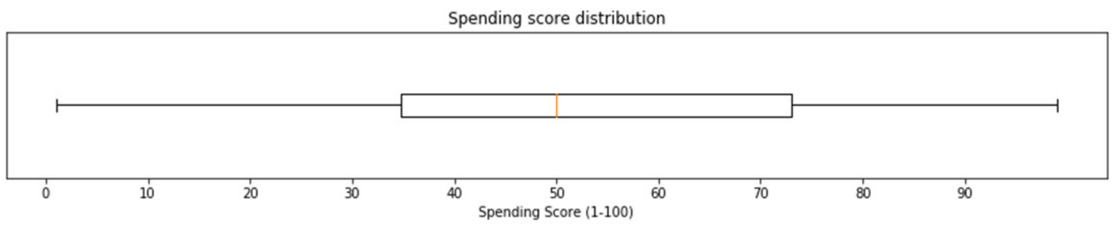
Let's create a boxplot to see the distribution of the Annual Income (k$) and Spending Score (1-100) feature using matplotlib library.

```
plt.figure(figsize=(15,2))
plt.boxplot(df['Annual Income (k$)'], vert = False)
plt.title('Income distribution')
plt.yticks([])
plt.xticks(range(0,150,10))
plt.xlabel('Annual Income (k$)')
plt.show()
```



From Annual income distribution boxplot we can tell that most of the costumers has an annual salary from approximately 40K to 80 K $. In the boxplot we can see there's an outlier data above the max value, which means there's a costumer that has much more annual salary than the other customer.

```
plt.figure(figsize=(15,2))
plt.boxplot(df['Spending Score (1-100)'], vert = False)
plt.title('Spending score distribution')
plt.yticks([])
plt.xticks(range(0,100,10))
plt.xlabel('Spending Score (1-100)')
plt.show()
```



From the spending score distribution boxplot we can tell that most of spending score lies around 35K to 73 K $. And there's no outlier data at the spending score distribution. Which means the distribution is (strictly) normally distributed.

We're going to use only these two feature to clusterize our customers, since the other features wouldn't really help at all with our analysis.

K-Means Clustering algorithm

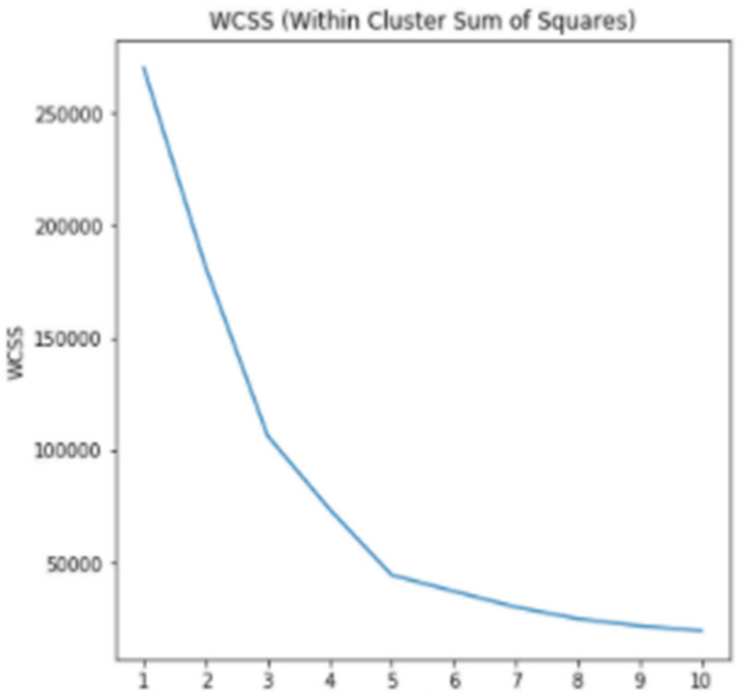
```
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values
```

```
plt.figure(figsize=(6,6))
plt.scatter(X[:,0], X[:,1], s=7)
plt.title('Annual income vs Spending score distribution')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()
```

First thing we do to use this k-means algorithm is to get our features values just as we can see at our code.
And then we might want to check how our data scattered (using the matplotlib library) to roughly predict how many cluster do we might have from our data.
We can tell that (intuitively) there are about 5 cluster.
Just to make sure, we're going to use the WCSS (within cluster sum of squares) method or the elbow method to determine the total of our clusters.

```
# Ngitung WCSS
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, random_state=41)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
# Menampilkan graph WCSS untuk melakukan elbow method
plt.figure(figsize=(6,6))
plt.title('WCSS (Within Cluster Sum of Squares)')
plt.plot(range(1,11), wcss)
plt.xticks(range(1,11))
plt.xlabel('K')
plt.ylabel('WCSS')
plt.show()
```



Using the elbow method, we can see that the plot starting to fashionly linear (from the graph) when the value of k is 5. which means our previous prediction was true that the total of the cluster for our analysis is 5 cluster.
Now that we have the total of our cluster, we can start to train our data to clusterize our customers.

```
# Inisialisasi object KMeans
kmeans = KMeans(n_clusters=5, random_state = 41)

# Memulai proses clustering
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=41, tol=0.0001, verbose=0)
```

```
y_kmeans = kmeans.predict(X)
print(y_kmeans)
```

```
[2 4 9 4 2 4 9 4 9 4 9 4 9 4 9 4 2 4 2 4 2 4 9 4 9 4 2 4 2 4 9 4 9 4 9 4 9
 4 2 4 2 4 2 0 2 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 7 0 0 7 7 0 0 0 0 0 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 3 7 5 7 3 1 5 1 5 7 5 1 3 1 5 1 5 1 3 7 3 1 3 7 5 1
 3 1 3 1 5 1 3 1 5 1 5 7 3 1 3 1 5 1 3 1 5 1 5 1 3 1 3 1 5 1 3 8 3 8 3 8 6
 8 6 8 6 8 6 8 6 8 6 8 6 8 6]
```

```
# koordinat center (titik pusat) dari tiap cluster
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[86.53846154 82.12820513]
 [55.2962963  49.51851852]
 [25.72727273 79.36363636]
 [88.2        17.11428571]
 [26.30434783 20.91304348]]
```

```python
# Inisialisasi object KMeans
kmeans = KMeans(n_clusters=5, random_state = 41)

# Memulai proses clustering
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=41, tol=0.0001, verbose=0)
```

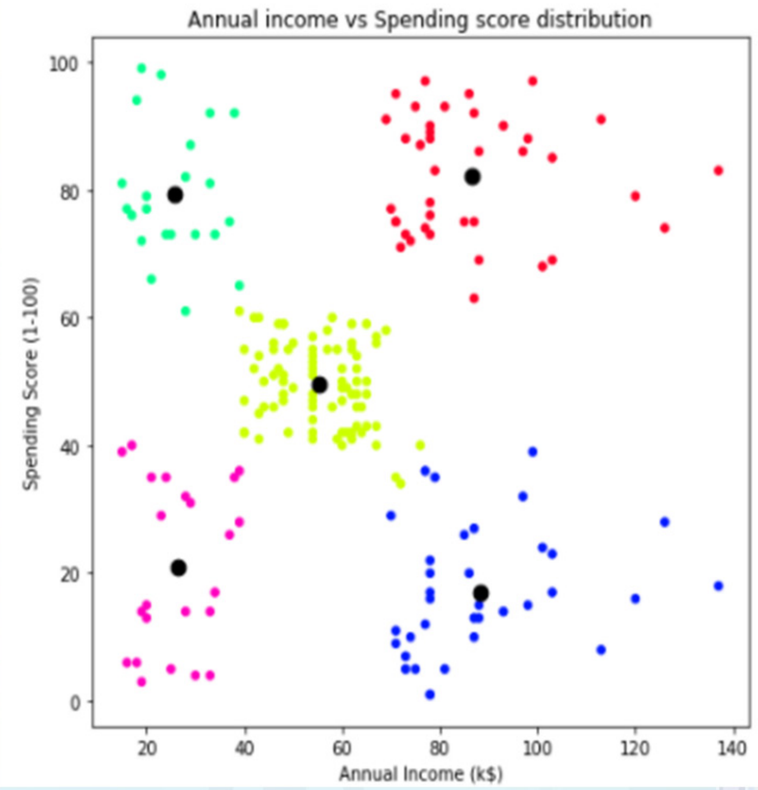
```python
y_kmeans = kmeans.predict(X)
print(y_kmeans)
```

```
[2 4 9 4 2 4 9 4 9 4 9 4 9 4 9 4 2 4 2 4 2 4 9 4 9 4 2 4 2 4 9 4 9 4 9 4 9
 4 2 4 2 4 2 0 2 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 7 0 0 7 7 0 0 0 0 0 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 3 7 5 7 3 1 5 1 5 7 5 1 3 1 5 1 5 1 3 7 3 1 3 7 5
 1 3 1 3 1 5 1 3 1 5 1 5 7 3 1 3 1 5 1 3 1 5 1 5 1 3 1 3 1 5 1 3 8 3 8 3 8
 3 8 6 8 6 8 6 8 6 8 6 8 6 8 6]
```

```python
# koordinat center (titik pusat) dari tiap cluster
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[86.53846154 82.12820513]
 [55.2962963  49.51851852]
 [25.72727273 79.36363636]
 [88.2        17.11428571]
 [26.30434783 20.91304348]]
```

```python
plt.figure(figsize=(7,7))
plt.scatter(X[:,0], X[:,1], s=20, c=y_kmeans,cmap='gist_rainbow')
plt.scatter(centroids[:,0],centroids[:,1], s=75, c='black')
plt.title('Annual income vs Spending score distribution')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()
```



To make our model, first we have to input number of cluster into the *KMeans* parameter. For the random state I specify 42, it actually doesn't really matter what value we want to specify to the *random_state* parameter. And then, we train our model with our data to starting clusterize our data and getting the *centroids of each cluster.* Now that we have our centroids and clusterized each of the customer, we can visualize it in a graph to see how our data is clusterize. But what actually can we tell from this clusterization ?

1. The *red* cluster represents the customers who both have high annual income and spending score. The mall need keep advertize to these customers so that tey will keep shopping at the mall. We can advertize through email or maybe through text massages to reach these customers.

2. The *blue* cluster represents the customers who has high annual income but doesn't spend so much. The mall need to advertize more to these customer, or maybe have a questioner to ask their opinion to improve the mall so that these customers would shop more ofter.

3. The *green* clusters represents customers who have medium annual income and also medium spending score.
I believe there's no need any special treatment for these customer.

4. The *pink* clusters represents customers who have low annual income and also low spending score. I honestly don't think we need to put more ad to these customer since they don't spend so much and also probably these customers try to save their money as much as they can.

5. The last cluster (i'm not sure what color they are) is kind of weird. It's represents customers who have low income but they spend much money at the mall. It could be they already have a big saving (from previous jobs or from parents) or could be they always in debt to someone

Our table with added cluster columns

```
1 df.head(20)
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	low income, low spend
1	2	Male	21	15	81	Low Income, spend much
2	3	Female	20	16	6	low income, low spend
3	4	Female	23	16	77	Low Income, spend much
4	5	Female	31	17	40	low income, low spend
5	6	Female	22	17	76	Low Income, spend much
6	7	Female	35	18	6	low income, low spend
7	8	Female	23	18	94	Low Income, spend much
8	9	Male	64	19	3	low income, low spend
9	10	Female	30	19	72	Low Income, spend much
10	11	Male	67	19	14	low income, low spend
11	12	Female	35	19	99	Low Income, spend much
12	13	Female	58	20	15	low income, low spend
13	14	Female	24	20	77	Low Income, spend much
14	15	Male	37	20	13	low income, low spend
15	16	Male	22	20	79	Low Income, spend much
16	17	Female	35	21	35	low income, low spend
17	18	Male	20	21	66	Low Income, spend much
18	19	Male	52	23	29	low income, low spend
19	20	Female	35	23	98	Low Income, spend much

On the table above, i've added new column which is cluster to represents each of the customer's cluster. I've write the *Red* cluster as 'High income, spend much', the *blue* cluster as 'High Income, low spend', the *green* cluster as 'medium income, medium spend', the *pink* cluster as 'low income, low spend', and the last cluster as 'Low Income, spend much'.
This way the marketing team can read the data easier to target their marketing area.

*Background pictures are taken from *freepik.com*

Thank you for your time!