Problem :

Given Some data of the customers of a mall that contain the following:

1. CustomerID, which is the customer's ID number.
2. Male / Female
3. Age, which is the age of the customer.
4. Annual Income, which is the customer's annual salary.
5. Class. The mall classifies customers into three categories according to their spending in the mall, in order to be able to adequately market each category. Tier 1 is the least spender, Tier 2 is the average spender, and Tier 3 spends the most. ### We need to creat a model that predict each customer category. # What we will cover : ## 1. Exploratory Data analysis ## 2. Data cleaning and preprocessing ## 3. Model building, evaluation and predicting the target of the test Data.

In [32]:
```
# importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from scipy.stats import uniform, truncnorm, randint
from sklearn.model_selection import RandomizedSearchCV

#ignore warning messages
import warnings
warnings.filterwarnings('ignore')

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [33]:
```
# loading the data
df_train = pd.read_csv("data.csv")
df_test = pd.read_csv("test_data.csv")
```

EDA and Data cleaning

In [34]:
```
# show first 5 rows
df_train.head()
```

Out[34]:

	Unnamed: 0	CustomerID	Genre	Age	Annual Income (k$)	Class
0	0	10	Female	30	19	3
1	1	20	Female	35	23	3
2	2	200	Male	30	137	3
3	3	153	Female	44	78	1
4	4	4	Female	23	16	3

In [35]:
```
# Information about the data
df_train.info()
```

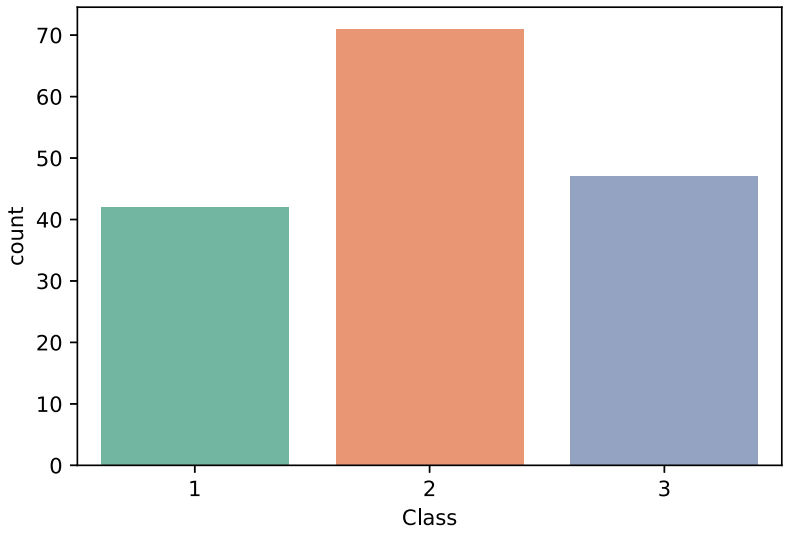
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160 entries, 0 to 159
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   Unnamed: 0           160 non-null    int64
 1   CustomerID           160 non-null    int64
 2   Genre                160 non-null    object
 3   Age                  160 non-null    int64
 4   Annual Income (k$)   160 non-null    int64
 5   Class                160 non-null    int64
dtypes: int64(5), object(1)
memory usage: 7.6+ KB
```

Great! we have no nulls. we need to drop (unnamed: 0) column and rename our columns (Genre, Annual Income (K$))

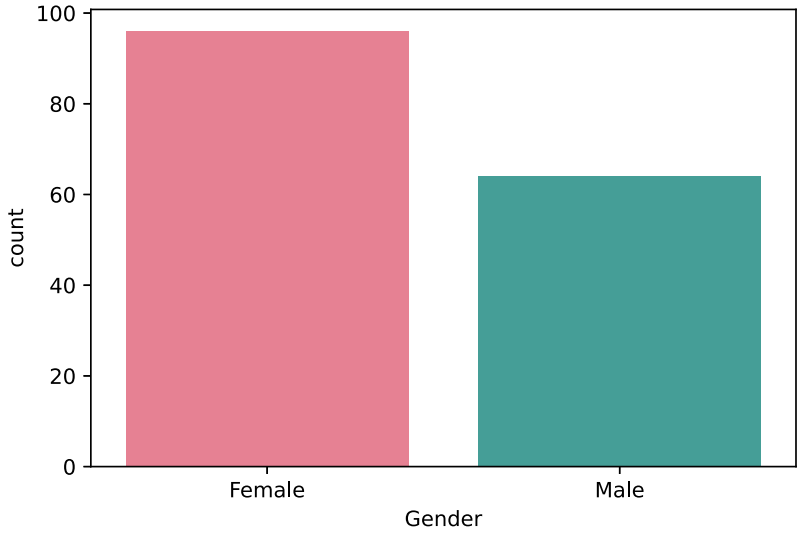
In [36]:
```
# dropping column
df_train.drop("Unnamed: 0", axis=1, inplace=True)
df_test.drop("Unnamed: 0", axis=1, inplace=True)

#rename columns
df_train.rename(columns={'Genre': 'Gender', 'Annual Income (k$)': 'Annual_Income_k$'}, inplace=True)
df_test.rename(columns={'Genre': 'Gender', 'Annual Income (k$)': 'Annual_Income_k$'}, inplace=True)
```

In [37]:
```
# target Class Distribution
sns.countplot(df_train.Class, palette="Set2");
```



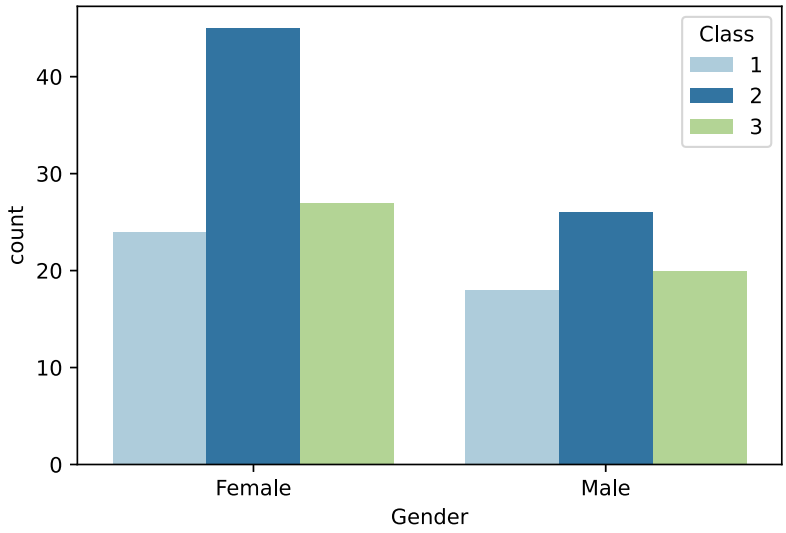
most customers in the data are from 2nd class, 1st and 3rd are slightly equall.

In [38]:
```
# Gender distribution
sns.countplot(df_train.Gender, palette="husl");
```

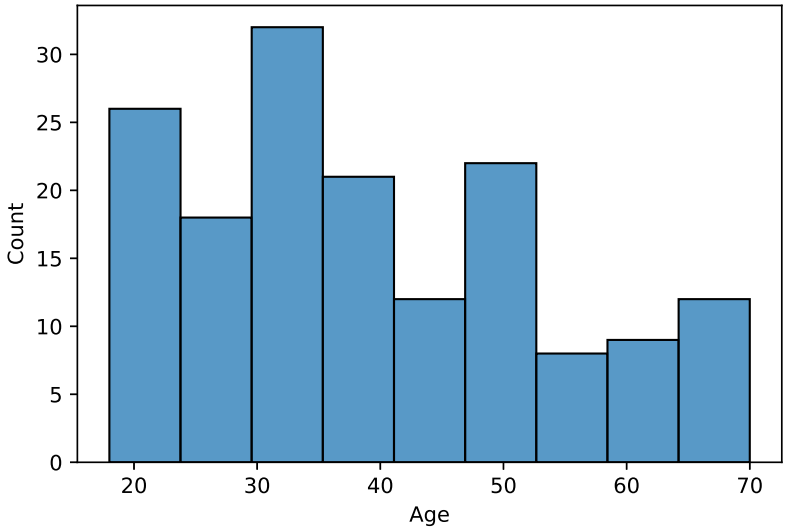


females are about 60% of the data while males are 40%

In [39]:
```
# Gender distribution to each class
sns.countplot("Gender", hue="Class", data=df_train, palette="Paired");
```



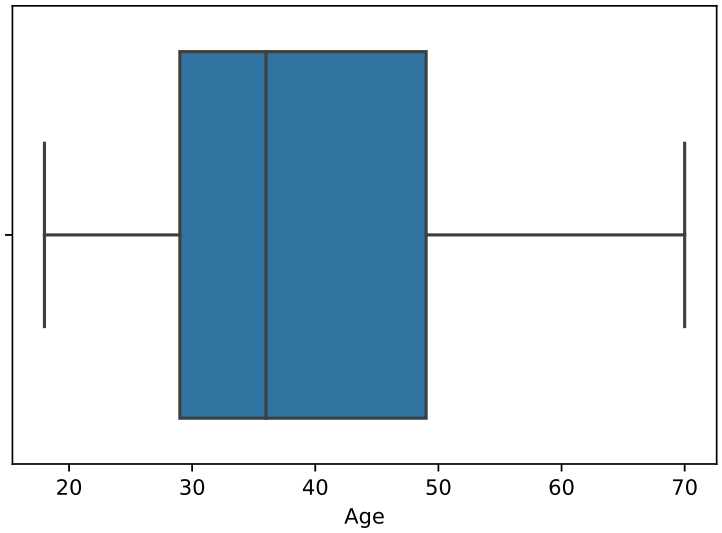
In [40]:
```
# Age distribution
sns.histplot(df_train.Age);
```



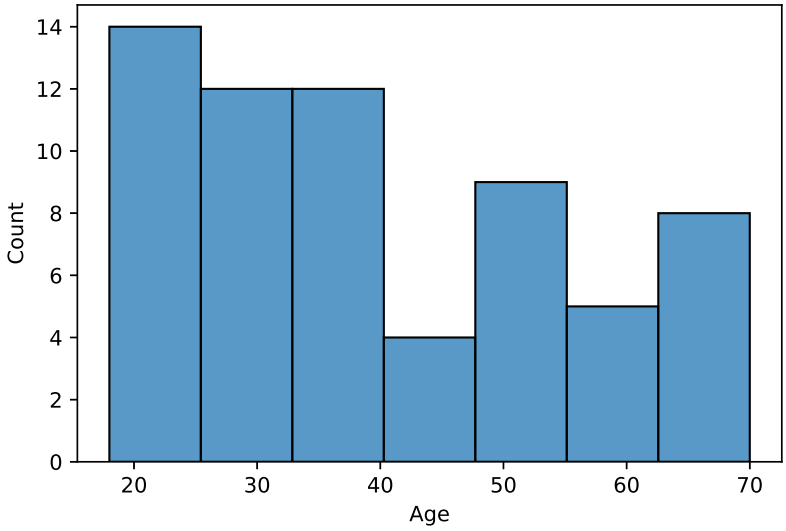
most customers ages from 20 to 50.

In [41]:
```
# age boxplot
sns.boxplot(df_train["Age"])
```

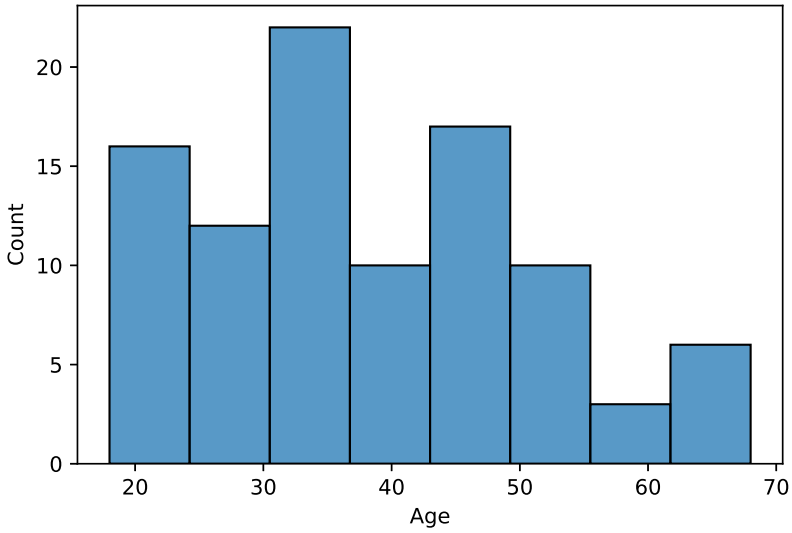
Out[41]:
```
<AxesSubplot:xlabel='Age'>
```



In [42]:
```
# Male age distribution
sns.histplot(df_train[df_train.Gender == 'Male'].Age);
```

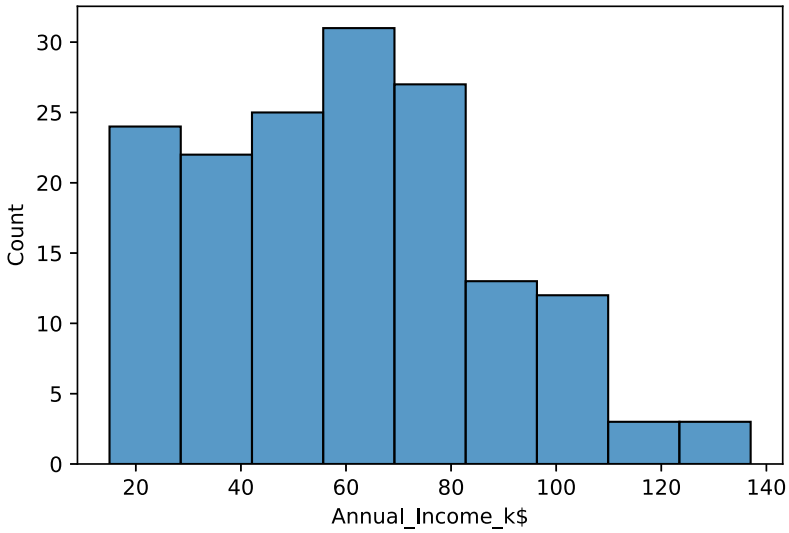


In [43]:
```
# Female age distribution
sns.histplot(df_train[df_train.Gender == 'Female'].Age);
```



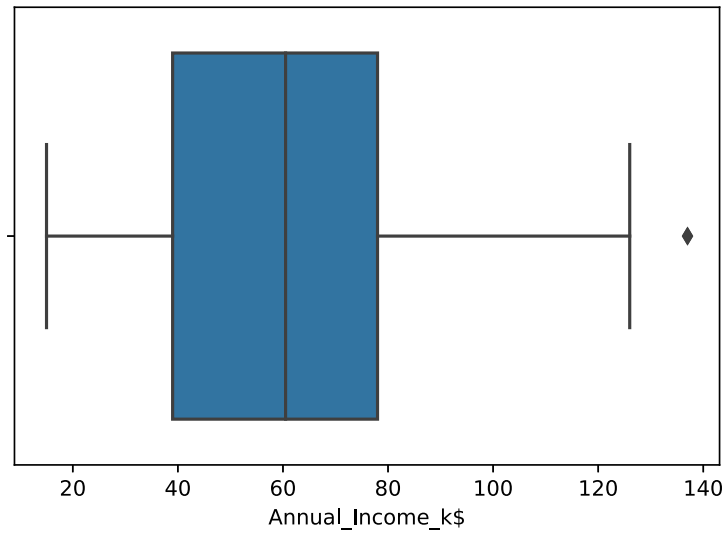
more males ages about 20->25 while more females ages about 30->35

In [44]:
```
# Annual income distribution
sns.histplot(df_train["Annual_Income_k$"]);
```



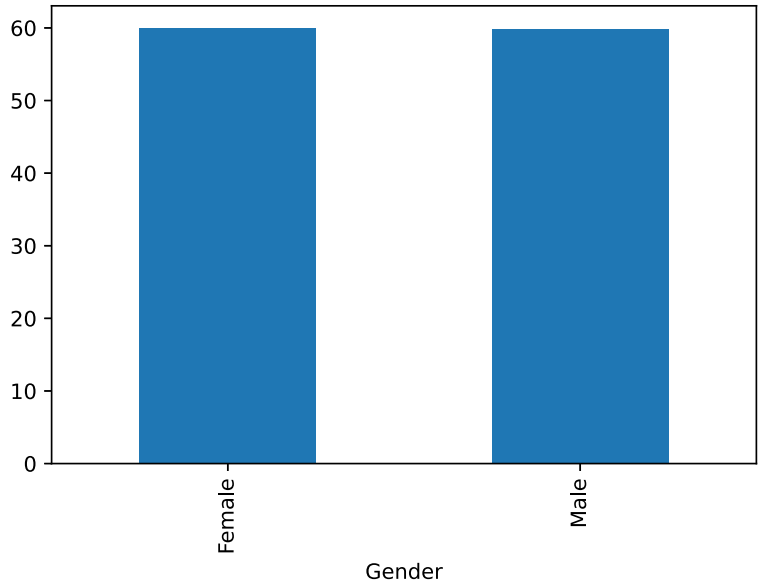
Most customers make 50k->80K annually

In [45]:
```
# Income boxplot
sns.boxplot(df_train["Annual_Income_k$"]);
```



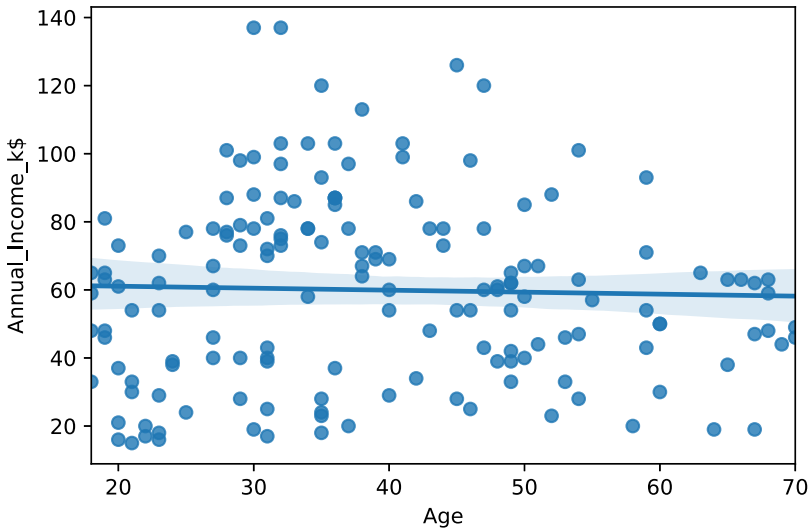
In [46]:
```
# Average income for each gender
print(df_train.groupby("Gender")["Annual_Income_k$"].mean())
df_train.groupby("Gender")["Annual_Income_k$"].mean().plot(kind = "bar");
```

```
Gender
Female    60.041667
Male      59.843750
Name: Annual_Income_k$, dtype: float64
```



Both Gender average incomes are equall.

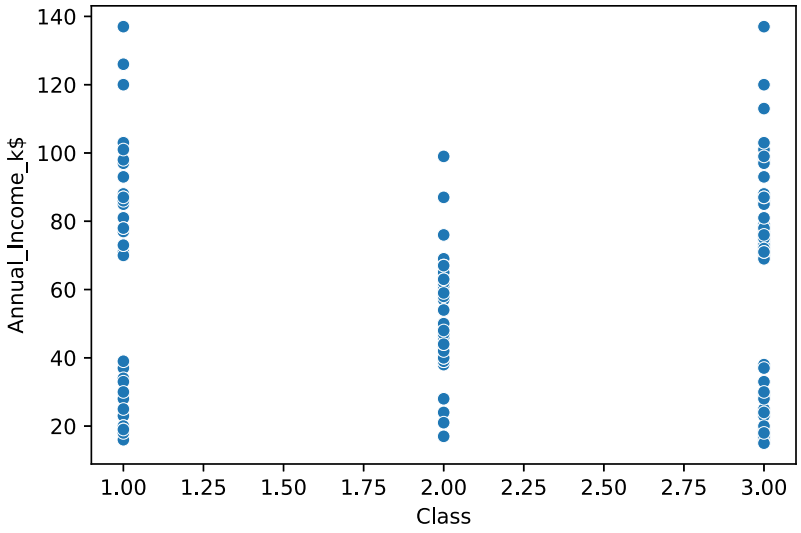
In [47]:
```
# Age and income relation
sns.regplot(df_train["Age"], df_train["Annual_Income_k$"]);
```



There is no clear direct relation but our Data shows that customers at middle ages have more annual income.

In [48]:
```
# Class and Income
sns.scatterplot(df_train["Class"], df_train["Annual_Income_k$"])
```

Out[48]:
```
<AxesSubplot:xlabel='Class', ylabel='Annual_Income_k$'>
```



Customers whose income about 60k (Average Income) tends to be in 2nd class (Average Spender)

Data Preprocessing

In this part I'll do the following:

1. encode categorical features (Gender)
2. drop customerID as it is unique for each one
3. normalize Income and age columns

In [49]:
```
# function to preprocess our data
def preprocess(data):

    # encode gender col
    data.Gender.replace({"Male":0, "Female":1}, inplace=True)

    # drop customer id
    data.drop("CustomerID", axis=1, inplace=True)

    # Normalize Annual income and age
    scaler_income = MinMaxScaler()
    data["Annual_Income_k$"] = scaler_income.fit_transform(data["Annual_Income_k$"].values.reshape(-1,1))
    scaler_age = MinMaxScaler()
    data["Age"] = scaler_age.fit_transform(data["Age"].values.reshape(-1,1))
```

In [50]:
```
# preprocess data
preprocess(df_train)
df_train.head()
```

Out[50]:

	Gender	Age	Annual_Income_k$	Class
0	1	0.230769	0.032787	3
1	1	0.326923	0.065574	3
2	0	0.230769	1.000000	3
3	1	0.500000	0.516393	1
4	1	0.096154	0.008197	3

Model building

In [51]:
```
# split the data
X = df_train.drop("Class", axis=1)
y = df_train["Class"]

# dict to contain our models
```

```
models = {"KNN": KNeighborsClassifier(),
          "RFC": RandomForestClassifier(),
          "LR": LogisticRegression(),
          "GBC":GradientBoostingClassifier(),
          }


# function for training and evaluating given models
def train_and_evaluate(models, X, y):

    scores = {}
    for name, model in models.items():
        scores[name] =  cross_val_score(model, X, y, cv=5)

    print(pd.DataFrame(scores))
```

In [52]:

```
# training and evaluating the models
train_and_evaluate(models, X, y);
```

```
       KNN      RFC       LR      GBC
0  0.65625  0.65625  0.56250  0.62500
1  0.71875  0.90625  0.50000  0.84375
2  0.84375  0.78125  0.53125  0.84375
3  0.75000  0.87500  0.53125  0.81250
4  0.78125  0.81250  0.50000  0.71875
```

Random forest classifier has the best results.

In [53]:

```
# Tuning Hyperparameters
model_params = {
    'n_estimators': randint(4,200),
    'max_features': truncnorm(a=0, b=1, loc=0.25, scale=0.1),
    'min_samples_split': uniform(0.01, 0.199)
}

rfc = RandomForestClassifier()

# set up random search
clf = RandomizedSearchCV(rfc, model_params, n_iter=100, cv=5, random_state=1)

# train the random search to find the best model
model = clf.fit(X, y)
print(model.score(X,y))

# print winning set of hyperparameters
from pprint import pprint
pprint(model.best_estimator_.get_params())
```

```
0.9
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 0.2911213586696189,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 0.06353120920030778,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 60,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [54]:

```
# Create prediction dataframe
sub = pd.DataFrame()
sub["CustomerID"] = df_test["CustomerID"]
```

In [55]:

```
# preprocessing test data
preprocess(df_test)
```

In [56]:
```
# predict the classes of test set
predictions = model.predict(df_test)
```

In [57]:
```
sub["Class"] = predictions
sub
```

Out[57]:

	CustomerID	Class
0	1	2
1	147	1
2	159	3
3	177	1
4	198	3
5	83	2
6	76	2
7	86	2
8	81	2
9	158	3
10	72	2
11	96	2
12	139	1
13	110	2
14	148	3
15	193	3
16	127	1
17	17	1
18	88	2
19	104	2
20	178	3
21	128	1
22	12	1
23	114	2
24	95	2
25	141	1
26	171	1
27	14	3
28	70	2
29	106	2
30	82	2
31	131	1
32	161	1
33	55	2
34	38	3
35	133	3
36	65	2
37	152	1
38	52	2

	CustomerID	Class
39	154	3

In [58]:
```
sub.to_csv("sub.csv", index=False)
```

Competition Accuracy Score

In [59]:
```
from sklearn.metrics import accuracy_score

test_class = pd.read_csv('test_class.csv')

print(accuracy_score(predictions, test_class.iloc[:, 1]))
```

```
0.75
```