Programming for AI - Final Project

Data Analysis and Data Visualisation

Group Members:

Anmol Zehrah (20K-0199), Hamza Sameer Khan (20K-1744), Arham Shahbaz (20K-1739)

In [1]:
```
# linear algebra and data processing
import numpy as np
import pandas as pd

#ploting libraries
import matplotlib.pyplot as plt
import seaborn as sns

#feature engineering
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

#train test split
from sklearn.model_selection import train_test_split

#metrics
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score as R2
from sklearn.model_selection  import cross_val_score as CVS

#ML models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Lasso

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

from sklearn.naive_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
traindata_df = pd.read_csv(r'D:\Academic Material\Semester 3\PAI Lab Assignments\
traindata_df.head(700)
```

Out[2]:

	Invoice ID	Branch	City	CustomerType	Gender	ProductLine	UnitPrice	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-31-3081	C	Naypyitaw	Normal	Female	NaN	15.28	5	3.8200
2	631-41-3108	NaN	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085
...	...	...	...	...	...	...	...	...	...
694	372-62-5264	NaN	Naypyitaw	Normal	Female	NaN	52.60	9	23.6700
695	800-09-8606	A	Yangon	Member	Female	Home and lifestyle	87.37	5	21.8425
696	182-52-7000	A	Yangon	Member	Female	Sports and travel	27.04	4	5.4080
697	826-58-8051	B	Mandalay	Normal	Male	NaN	62.19	4	12.4380
698	868-06-0466	NaN	Yangon	Member	Male	Electronic accessories	69.58	9	31.3110

699 rows × 17 columns

In [3]:
```
testdata_df = pd.read_csv(r'D:\Academic Material\Semester 3\PAI Lab Assignments\P
testdata_df.head(300)
```

Out[3]:

	Invoice ID	Branch	City	CustomerType	Gender	ProductLine	UnitPrice	Quantity	Tax 5%
0	751-41-9720	C	Naypyitaw	Normal	Male	Home and lifestyle	97.50	10	48.7500
1	626-43-7888	C	Naypyitaw	Normal	Female	Fashion accessories	60.41	8	24.1640
2	176-64-7711	NaN	Mandalay	Normal	Male	NaN	32.32	3	4.8480
3	191-29-0321	B	Mandalay	Member	Female	Fashion accessories	19.77	10	9.8850
4	729-06-2010	B	Mandalay	Member	Male	H and B	80.47	9	36.2115
...	...	...	...	...	...	...	...	...	...
295	652-49-6720	NaN	Naypyitaw	Member	Female	Electronic accessories	60.95	1	3.0475
296	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175
297	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900
298	727-02-1313	A	Yangon	Member	Male	NaN	31.84	1	1.5920
299	347-56-2442	NaN	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910

300 rows × 17 columns

In [4]:
```
# Number of rows and columns in both data files.
print(f"Training Dataset (row, col): {traindata_df.shape}\n\nTesting Dataset (row
```

```
Training Dataset (row, col): (699, 17)

Testing Dataset (row, col): (301, 17)
```

In [5]:
```
#column information
traindata_df.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               699 non-null    object 
 1   Branch                   591 non-null    object 
 2   City                     699 non-null    object 
 3   CustomerType             699 non-null    object 
 4   Gender                   699 non-null    object 
 5   ProductLine              623 non-null    object 
 6   UnitPrice                699 non-null    float64
 7   Quantity                 699 non-null    int64  
 8   Tax 5%                   699 non-null    float64
 9   Total                    699 non-null    float64
 10  Date                     699 non-null    object 
 11  Time                     699 non-null    object 
 12  Payment                  699 non-null    object 
 13  cogs                     699 non-null    float64
 14  gross margin percentage  699 non-null    float64
 15  GrossIncome              699 non-null    float64
 16  Rating                   699 non-null    float64
dtypes: float64(7), int64(1), object(9)
memory usage: 93.0+ KB
```

In [6]:
```
# statistical summary of train data set
traindata_df.describe()
```

Out[6]:

	UnitPrice	Quantity	Tax 5%	Total	cogs	gross margin percentage	GrossIncome
count	699.000000	699.000000	699.000000	699.000000	699.000000	6.990000e+02	699.000000
mean	55.691774	5.517883	15.351590	322.383393	307.031803	4.761905e+00	15.351590
std	26.205441	2.915851	11.507112	241.649343	230.142231	5.332887e-14	11.507112
min	10.130000	1.000000	0.604500	12.694500	12.090000	4.761905e+00	0.604500
25%	33.050000	3.000000	6.352000	133.392000	127.040000	4.761905e+00	6.352000
50%	54.730000	5.000000	12.096000	254.016000	241.920000	4.761905e+00	12.096000
75%	77.620000	8.000000	22.383000	470.043000	447.660000	4.761905e+00	22.383000
max	99.960000	10.000000	49.650000	1042.650000	993.000000	4.761905e+00	49.650000

In [7]:
```python
# statistical summary of test data set
testdata_df.describe()
```

Out[7]:

	UnitPrice	Quantity	Tax 5%	Total	cogs	gross margin percentage	GrossIncome
count	301.000000	301.000000	301.000000	301.000000	301.000000	3.010000e+02	301.000000
mean	55.626512	5.491694	15.443879	324.321453	308.877575	4.761905e+00	15.443879
std	27.198550	2.945748	12.183733	255.858383	243.674650	9.786232e-15	12.183733
min	10.080000	1.000000	0.508500	10.678500	10.170000	4.761905e+00	0.508500
25%	32.320000	3.000000	4.768000	100.128000	95.360000	4.761905e+00	4.768000
50%	56.000000	6.000000	12.080000	253.680000	241.600000	4.761905e+00	12.080000
75%	79.590000	8.000000	22.858500	480.028500	457.170000	4.761905e+00	22.858500
max	99.960000	10.000000	48.750000	1023.750000	975.000000	4.761905e+00	48.750000

In [8]:
```python
# Missing values in ascending order
print("Train Data:\n")
print(traindata_df.isnull().sum().sort_values(ascending=True),"\n\n")
print("Test Data:\n")
print(testdata_df.isnull().sum().sort_values(ascending=True),"\n\n")
```

```
Train Data:

Invoice ID                 0
gross margin percentage    0
cogs                       0
Payment                    0
Time                       0
Date                       0
Total                      0
GrossIncome                0
Tax 5%                     0
UnitPrice                  0
Gender                     0
CustomerType               0
City                       0
Quantity                   0
Rating                     0
ProductLine               76
Branch                   108
```

In [9]:
```
# Value count of both categories
print("Branch Count: \n")
print(traindata_df.Branch.value_counts(), "\n\n")
print("ProductLine Count: \n")
print(traindata_df.ProductLine.value_counts(), "\n\n")
```

```
Branch Count:

B    205
A    197
C    189
Name: Branch, dtype: int64


ProductLine Count:

Sports and travel         116
Fashion accessories       115
Food and beverages        108
Electronic accessories     99
Home and lifestyle         95
Health and beauty          73
H and B                    17
Name: ProductLine, dtype: int64
```

In [10]:
```
# Mode values for both categories in each dataset
print("Branch: \nMode of test values, Mode of train values:\n",[testdata_df['Bran
print("\nProductLine: \nMode of test values, Mode of train values:\n",[testdata_d
```

```
Branch:
Mode of test values, Mode of train values:
 ['B', 'B']

ProductLine:
Mode of test values, Mode of train values:
 ['Electronic accessories', 'Sports and travel']
```

In [11]:
```python
# Information regarding both datasets
print("Train: \n")
print(traindata_df.info())
print("\n\nTest: \n")
print(testdata_df.info())
```

```
Train:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               699 non-null    object 
 1   Branch                   591 non-null    object 
 2   City                     699 non-null    object 
 3   CustomerType             699 non-null    object 
 4   Gender                   699 non-null    object 
 5   ProductLine              623 non-null    object 
 6   UnitPrice                699 non-null    float64
 7   Quantity                 699 non-null    int64  
 8   Tax 5%                   699 non-null    float64
 9   Total                    699 non-null    float64
 10  Date                     699 non-null    object 
 11  Time                     699 non-null    object 
 12  Payment                  699 non-null    object 
 13  cogs                     699 non-null    float64
 14  gross margin percentage  699 non-null    float64
 15  GrossIncome              699 non-null    float64
 16  Rating                   699 non-null    float64
dtypes: float64(7), int64(1), object(9)
memory usage: 93.0+ KB
None


Test:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               301 non-null    object 
 1   Branch                   261 non-null    object 
 2   City                     301 non-null    object 
 3   CustomerType             301 non-null    object 
 4   Gender                   301 non-null    object 
 5   ProductLine              265 non-null    object 
 6   UnitPrice                301 non-null    float64
 7   Quantity                 301 non-null    int64  
 8   Tax 5%                   301 non-null    float64
 9   Total                    301 non-null    float64
 10  Date                     301 non-null    object 
 11  Time                     301 non-null    object 
 12  Payment                  301 non-null    object 
 13  cogs                     301 non-null    float64
```

```
 14  gross margin percentage  301 non-null    float64
 15  GrossIncome              301 non-null    float64
 16  Rating                   301 non-null    float64
dtypes: float64(7), int64(1), object(9)
memory usage: 40.1+ KB
None
```

In [12]:
```
#list of all the numeric columns
num = traindata_df.select_dtypes('number').columns.to_list()
#list of all the categoric columns
cat = traindata_df.select_dtypes('object').columns.to_list()

#numeric df
BM_num =  traindata_df[num]
#categoric df
BM_cat = traindata_df[cat]

[traindata_df[category].value_counts() for category in cat[1:]]
```

Out[12]:
```
[B    205
 A    197
 C    189
 Name: Branch, dtype: int64,
 Yangon       236
 Mandalay     232
 Naypyitaw    231
 Name: City, dtype: int64,
 Normal    353
 Member    346
 Name: CustomerType, dtype: int64,
 Male      355
 Female    344
 Name: Gender, dtype: int64,
 Sports and travel         116
 Fashion accessories       115
 Food and beverages        108
 Electronic accessories     99
 Home and lifestyle         95
 Health and beauty          73
 H and B                    17
 Name: ProductLine, dtype: int64,
 1/25/2019    15
 3/5/2019     14
 2/15/2019    14
 3/9/2019     14
 1/26/2019    13
              ..
 2/11/2019     4
 3/17/2019     3
 1/9/2019      3
 2/19/2019     3
 2/28/2019     3
 Name: Date, Length: 89, dtype: int64,
 19:48    6
 10:11    5
 19:39    5
 19:20    5
 10:23    4
         ..
 12:31    1
 19:18    1
```

```
11:27    1
13:56    1
12:42    1
Name: Time, Length: 426, dtype: int64,
Cash           246
Ewallet        234
Credit card    219
Name: Payment, dtype: int64]
```

In [13]:
```
#list of all the numeric columns
num2 = testdata_df.select_dtypes('number').columns.to_list()
#list of all the categoric columns
cat2 = testdata_df.select_dtypes('object').columns.to_list()

#numeric df
BM_num2 =  testdata_df[num]
#categoric df
BM_cat2 = testdata_df[cat]

[testdata_df[category].value_counts() for category in cat[1:]]
```

Out[13]:
```
[B    88
 C    88
 A    85
 Name: Branch, dtype: int64,
 Yangon       104
 Mandalay     100
 Naypyitaw     97
 Name: City, dtype: int64,
 Member    155
 Normal    146
 Name: CustomerType, dtype: int64,
 Female    157
 Male      144
 Name: Gender, dtype: int64,
 Electronic accessories    48
 Home and lifestyle        47
 Fashion accessories       46
 Food and beverages        42
 Health and beauty         37
 Sports and travel         36
 H and B                    9
 Name: ProductLine, dtype: int64,
 3/2/2019     9
 1/8/2019     8
 2/7/2019     7
 1/19/2019    7
 1/24/2019    7
             ..
 3/12/2019    1
 1/16/2019    1
 2/10/2019    1
 1/22/2019    1
 1/21/2019    1
 Name: Date, Length: 85, dtype: int64,
 11:40    4
 11:51    3
 10:33    3
 12:40    3
 10:38    3
         ..
 17:26    1
 10:05    1
 15:05    1
 15:42    1
```

```
14:06    1
Name: Time, Length: 241, dtype: int64,
Ewallet        111
Cash            98
Credit card     92
Name: Payment, dtype: int64]
```

In [14]:
```
#train data replacement to make data more uniform
traindata_df['ProductLine'].replace(['H and B', 'Health and beauty'], ['Health Ar

traindata_df.ProductLine.value_counts()
```

Out[14]:
```
Sports and travel         116
Fashion accessories       115
Food and beverages        108
Electronic accessories     99
Home and lifestyle         95
Health And Beauty          90
Name: ProductLine, dtype: int64
```

In [15]:
```
#test data replacement to make data more uniform
testdata_df['ProductLine'].replace(['H and B', 'Health and beauty'], ['Health And

testdata_df.ProductLine.value_counts()
```

Out[15]:
```
Electronic accessories    48
Home and lifestyle        47
Fashion accessories       46
Health And Beauty         46
Food and beverages        42
Sports and travel         36
Name: ProductLine, dtype: int64
```

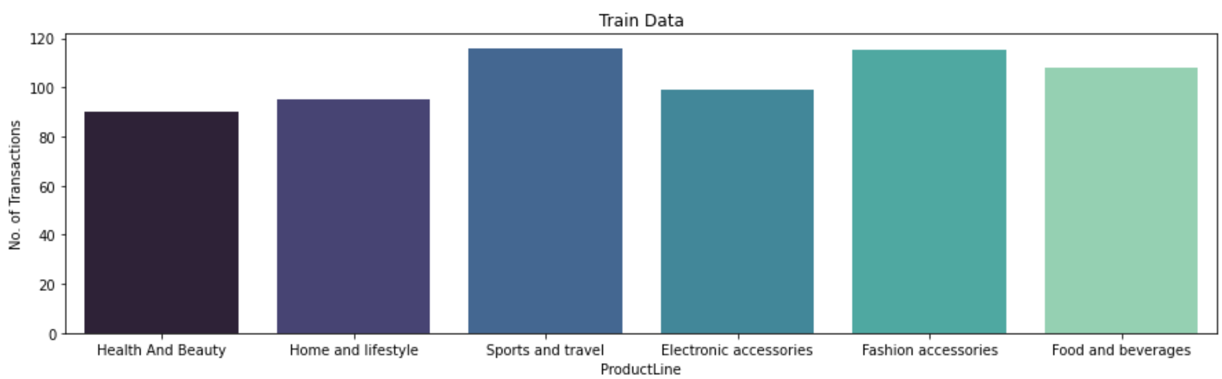
In [16]:
```
#traindata_df.to_csv("Bookupdatedtrain.csv", index = False)
#testdata_df.to_csv("Bookupdatedtest.csv", index = False)
```

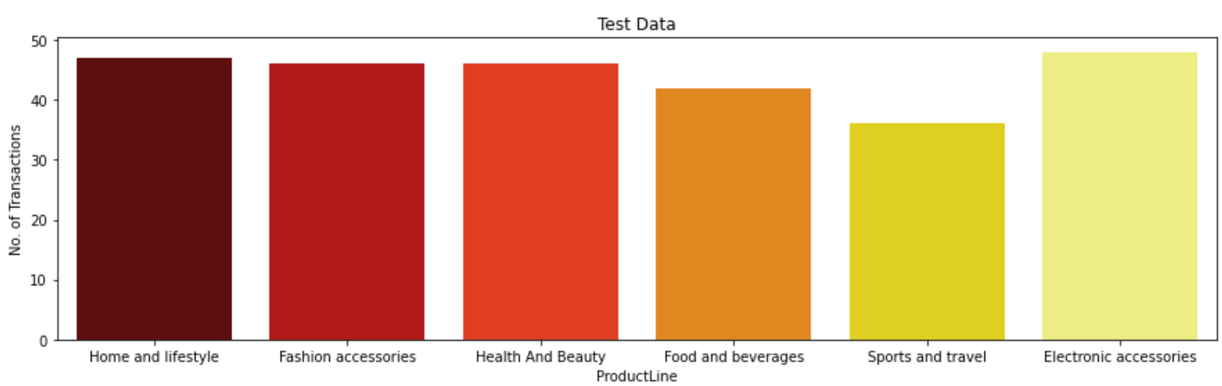
In [17]:
```
# updating the csv files
import csv
col= ['Invoice ID', 'Branch', 'City', 'CustomerType', 'Gender', 'ProductLine', 'U
list = traindata_df.values.tolist()
with open("Bookupdatedtrain.csv", 'w', newline ='') as f:
    writer = csv.writer(f)
    writer.writerow(col)
    writer.writerows(list)
    f.close()
```

In [18]:
```
# updating the csv files
import csv
col= ['Invoice ID', 'Branch', 'City', 'CustomerType', 'Gender', 'ProductLine', 'U
list = testdata_df.values.tolist()
with open("Bookupdatedtest.csv", 'w', newline ='') as f:
    writer = csv.writer(f)
    writer.writerow(col)
    writer.writerows(list)
    f.close()
```

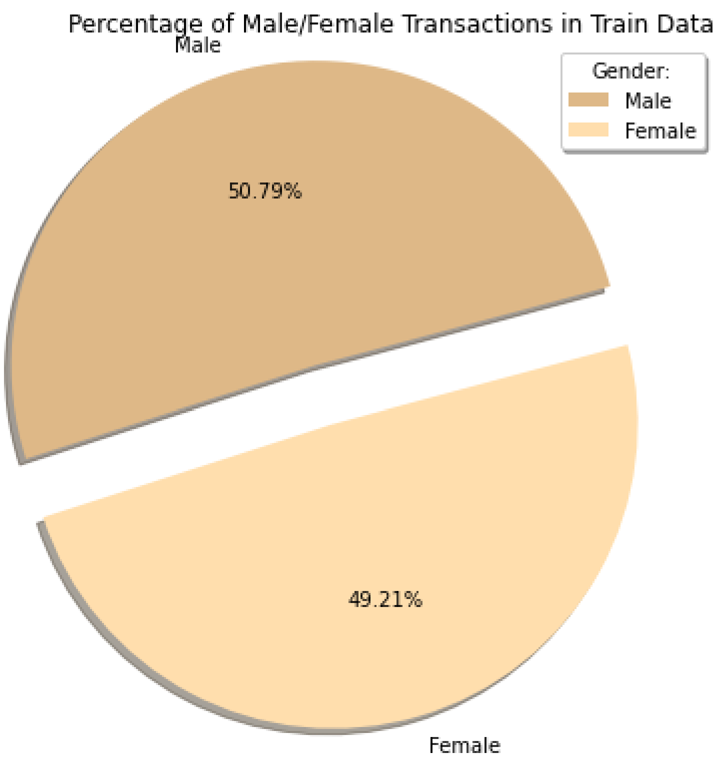
Data Visualization

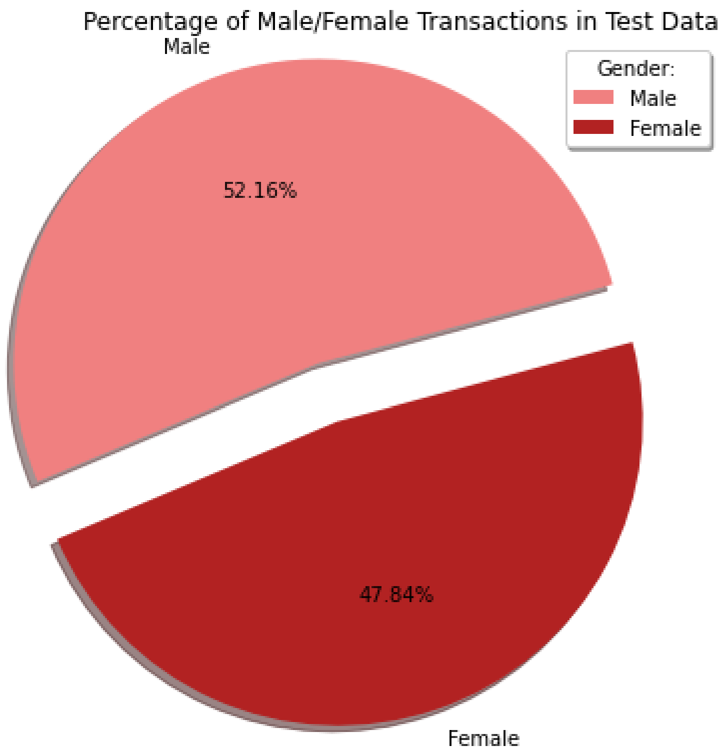
In [19]:
```
#Total number of transactions for each Product Line
plt.figure(figsize=(15,4))
sns.countplot(x=traindata_df.ProductLine, data=traindata_df ,palette='mako')
plt.ylabel("No. of Transactions")
plt.title("Train Data")
plt.show()
plt.figure(figsize=(15,4))
sns.countplot(x=testdata_df.ProductLine, data=testdata_df ,palette='hot')
plt.ylabel("No. of Transactions")
plt.title("Test Data")
plt.show()
```



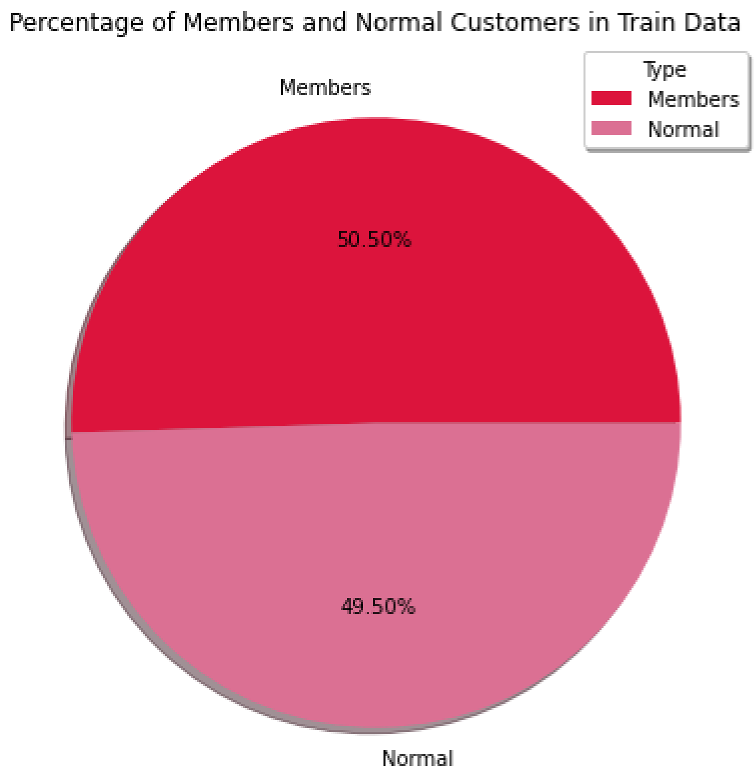


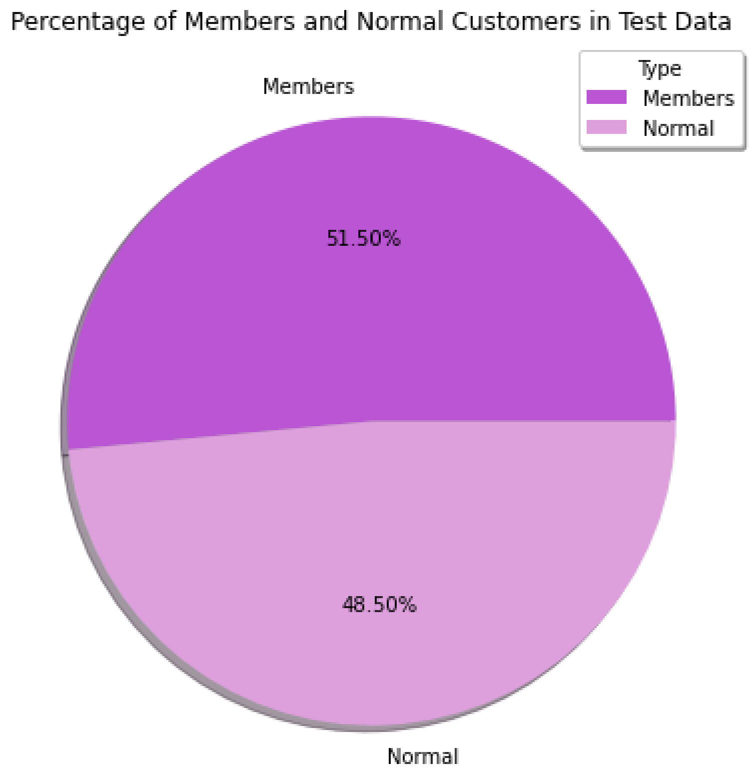
In [20]:
```
# Percentage of Male/Female making the transactions
plt.figure(figsize=(25,7))
plt.pie(traindata_df.Gender.value_counts(),labels=["Male","Female"], shadow = Tru
plt.legend(traindata_df.Gender,labels=["Male","Female"], shadow = True,title = "G
plt.title("Percentage of Male/Female Transactions in Train Data",loc="right")
plt.show()
plt.figure(figsize=(25,7))
plt.pie(testdata_df.Gender.value_counts(),labels=["Male","Female"], shadow = True
plt.legend(testdata_df.Gender,labels=["Male","Female"], shadow = True,title = "Ge
plt.title("Percentage of Male/Female Transactions in Test Data",loc="right")
plt.show()
```



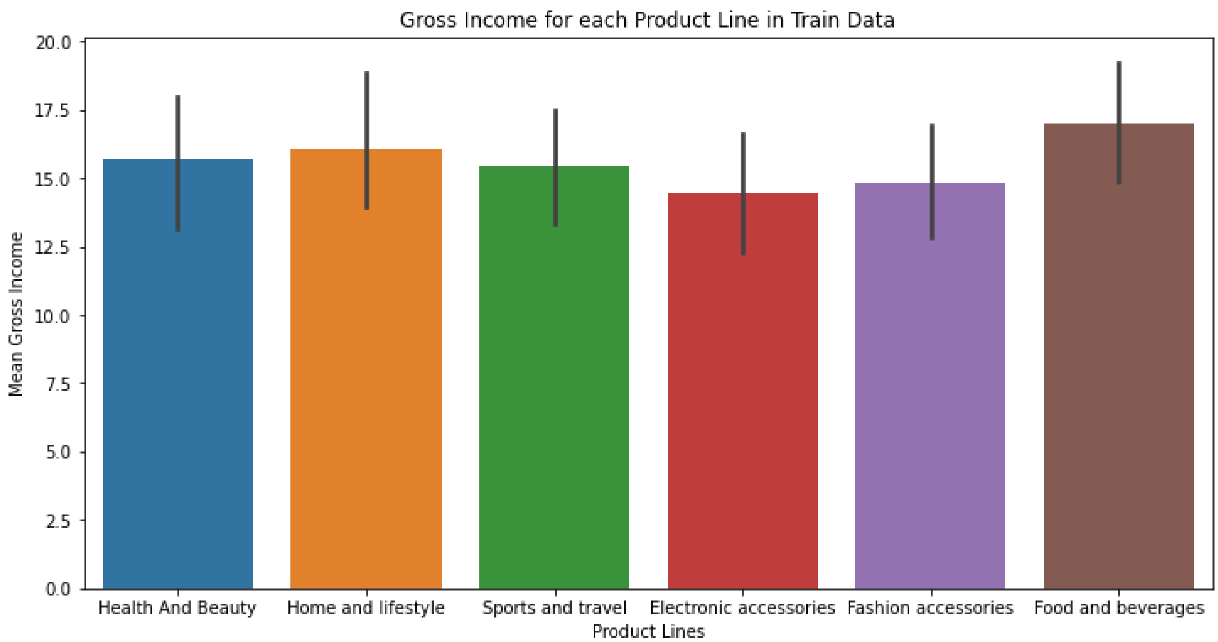

Percentage of Male/Female Transactions in Test Data
Male
Gender:
Male
Female
52.16%
47.84%
Female

In [21]:
```
# Percentage of members and normal Customer Type
plt.figure(figsize=(25,7))
plt.pie(traindata_df.CustomerType.value_counts(),labels=["Members","Normal"], sha
plt.legend(traindata_df.Gender,labels=["Members","Normal"], shadow = True,title =
plt.title("Percentage of Members and Normal Customers in Train Data")
plt.show()
plt.figure(figsize=(25,7))
plt.pie(testdata_df.CustomerType.value_counts(),labels=["Members","Normal"], shad
plt.legend(testdata_df.Gender,labels=["Members","Normal"], shadow = True,title =
plt.title("Percentage of Members and Normal Customers in Test Data")
plt.show()
```

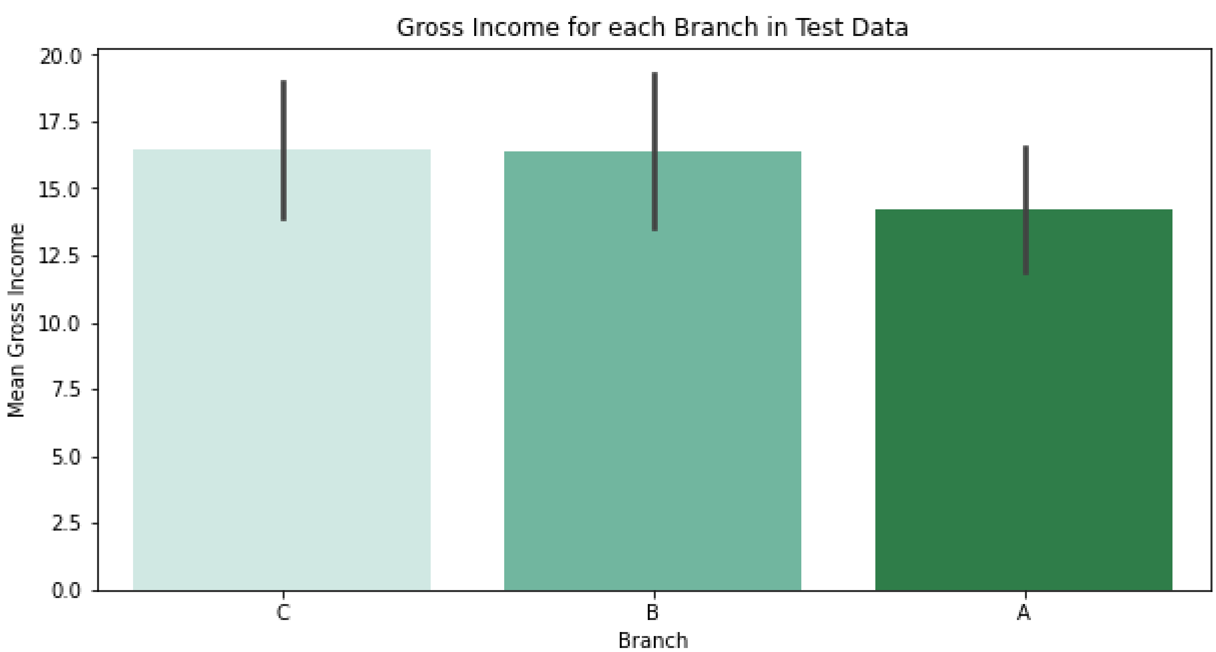



Percentage of Members and Normal Customers in Test Data
Type
Members
Normal
Members
51.50%
48.50%
Normal

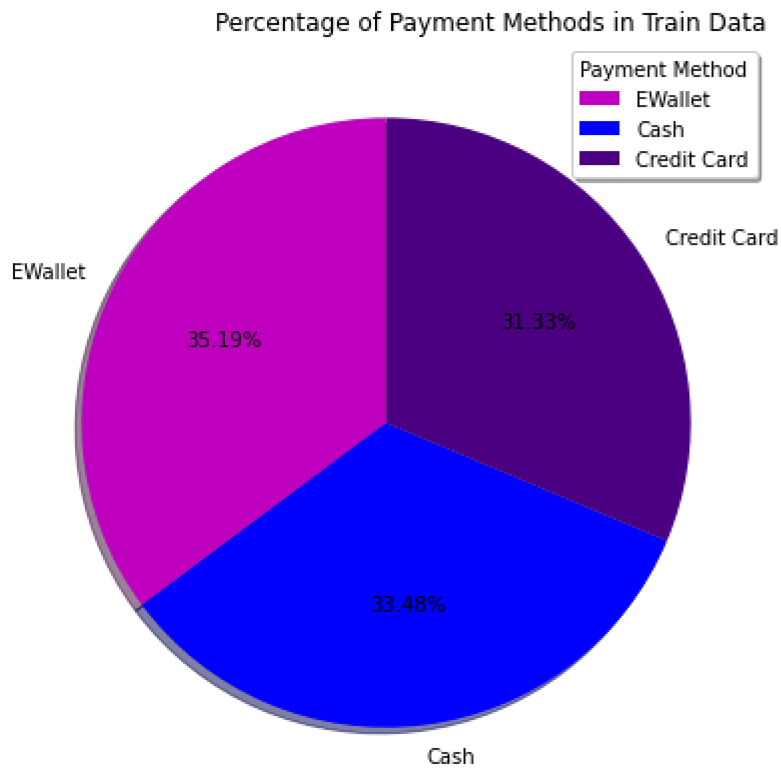
In [22]:
```
# Mean Gross income for each Product Line
plt.figure(figsize= (12,6))
sns.barplot(x = traindata_df['ProductLine'], y = traindata_df['GrossIncome'])
plt.xlabel("Product Lines")
plt.ylabel("Mean Gross Income")
plt.title("Gross Income for each Product Line in Train Data")
plt.show()
plt.figure(figsize= (12,6))
sns.barplot(x = testdata_df['ProductLine'], y = testdata_df['GrossIncome'],palett
plt.title("Gross Income for each Product Line in Test Data")
plt.xlabel("Product Lines")
plt.ylabel("Mean Gross Income")
plt.show()
```

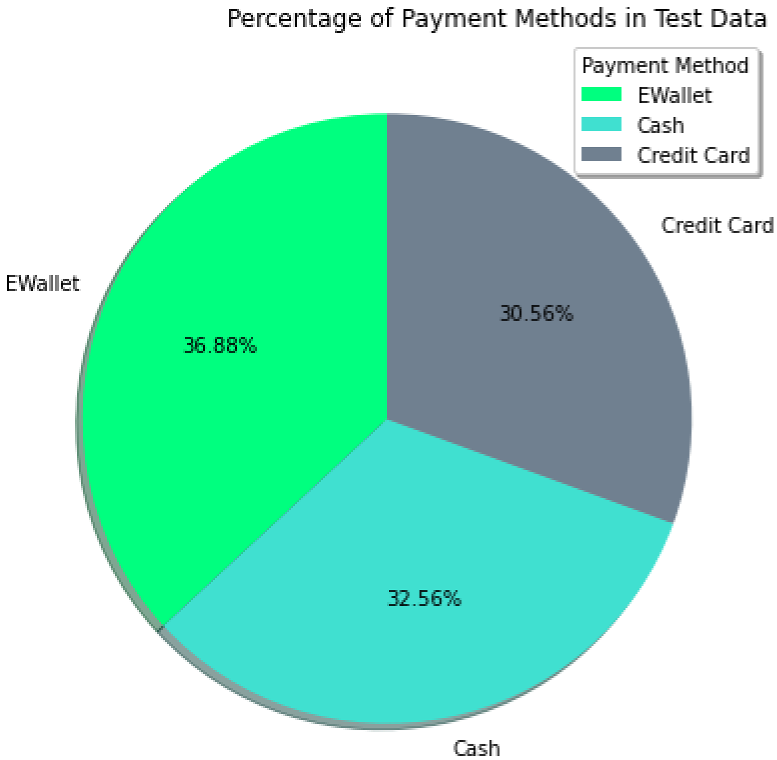


In [23]:
```
# Mean Gross Income for every Branch
plt.figure(figsize= (10,5))
sns.barplot(x = traindata_df['Branch'], y = traindata_df['GrossIncome'],palette='
plt.xlabel("Branch")
plt.ylabel("Mean Gross Income")
plt.title("Gross Income for each Branch in Train Data")
plt.show()
plt.figure(figsize= (10,5))
sns.barplot(x = testdata_df['Branch'], y = testdata_df['GrossIncome'],palette='Bu
plt.title("Gross Income for each Branch in Test Data")
plt.xlabel("Branch")
plt.ylabel("Mean Gross Income")
plt.show()
```

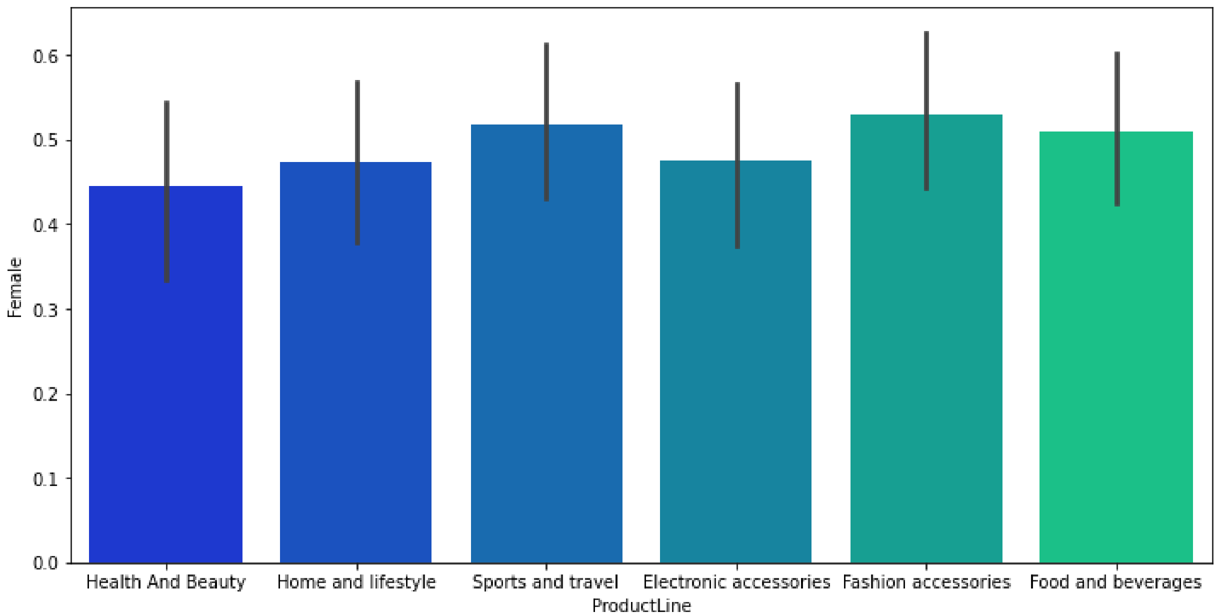


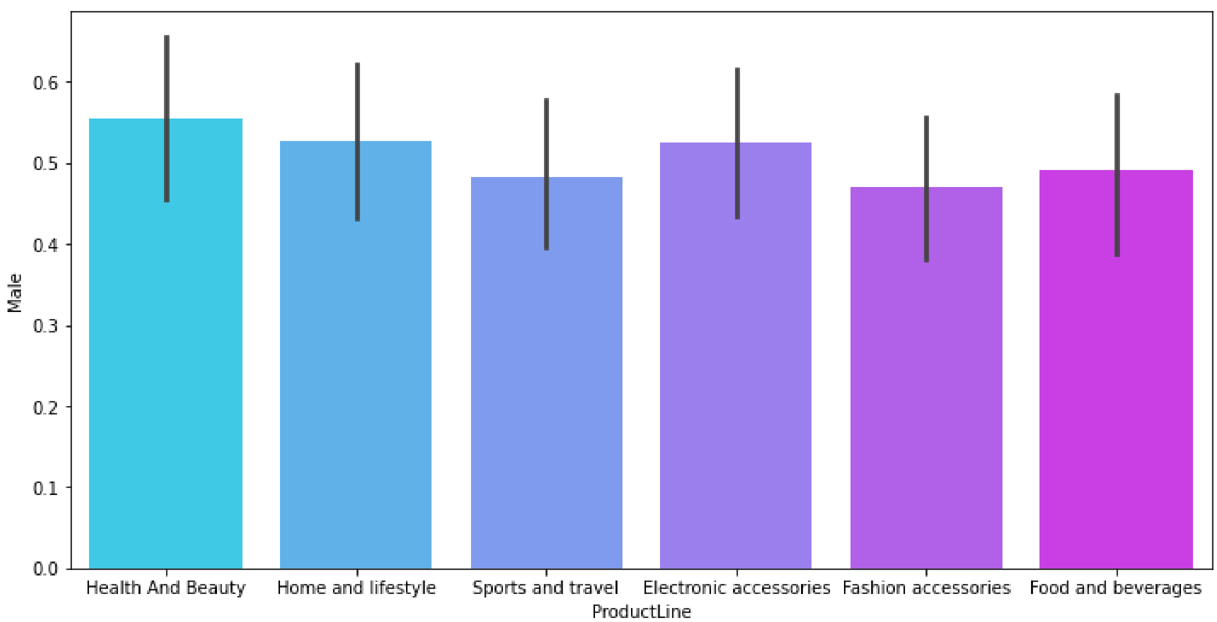
In [24]:
```
# Percentage Composition of Payment Methods in the transactions
plt.figure(figsize=(25,7))
plt.pie(traindata_df.Payment.value_counts(),labels=["EWallet","Cash","Credit Card
plt.legend(traindata_df.Payment.value_counts(),labels=["EWallet","Cash","Credit C
plt.title("Percentage of Payment Methods in Train Data",loc="right")
plt.show()
plt.figure(figsize=(25,7))
plt.pie(testdata_df.Payment.value_counts(),labels=["EWallet","Cash","Credit Card"
plt.legend(testdata_df.Payment.value_counts(),labels=["EWallet","Cash","Credit Ca
plt.title("Percentage of Payment Methods in Test Data",loc="right")
plt.show()
```



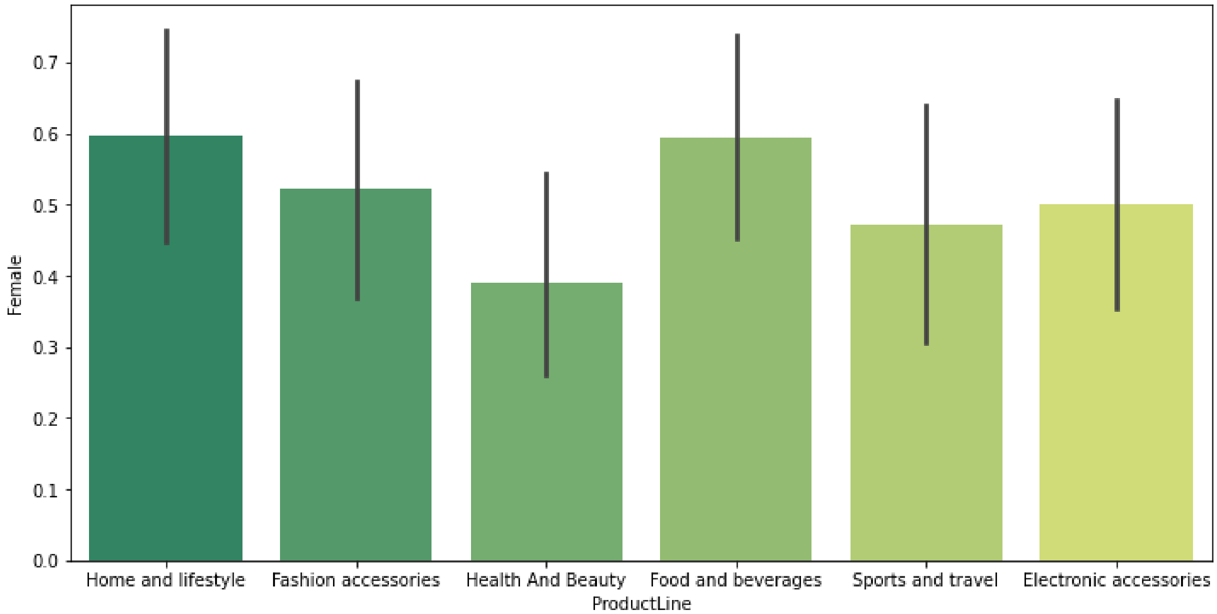

Percentage of Payment Methods in Test Data
Payment Method
EWallet
Cash
Credit Card
EWallet
36.88%
Credit Card
30.56%
32.56%
Cash

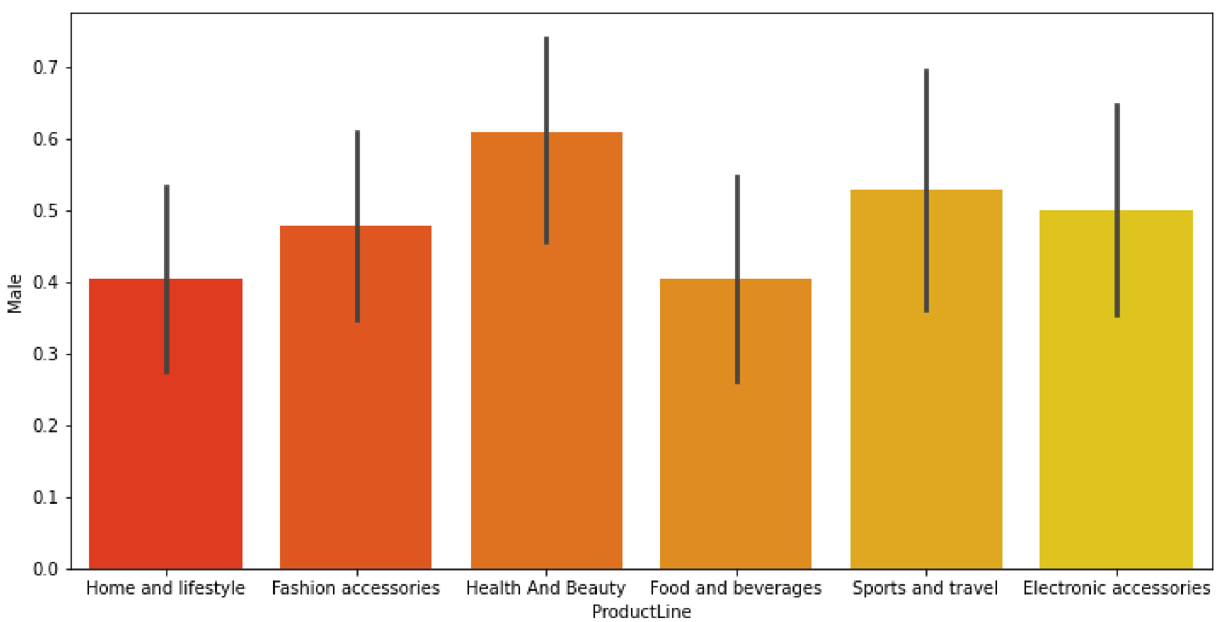
In [25]:
```
#  Composition Ratio of Gender for every Product Line in Train Data
gender_dummies  = pd.get_dummies(traindata_df['Gender'])
gender_dummies.head()
df = pd.concat([traindata_df, gender_dummies], axis = 1)
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'Female', data = df,palette='winter')
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'Male', data = df,palette='cool')
plt.show()
```



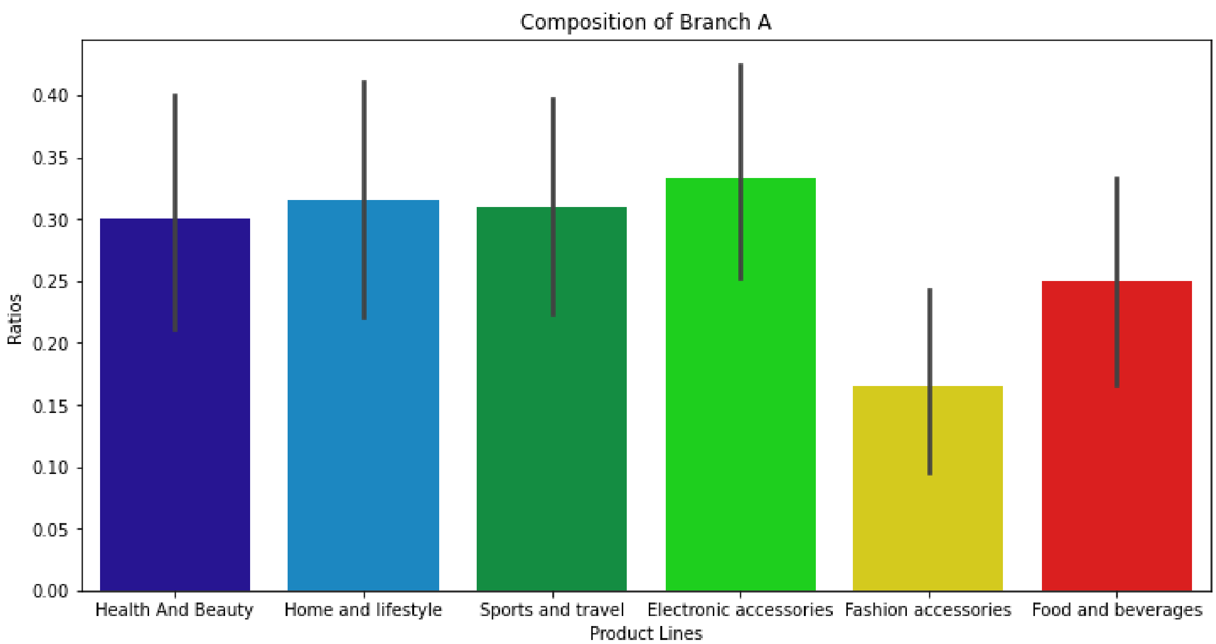


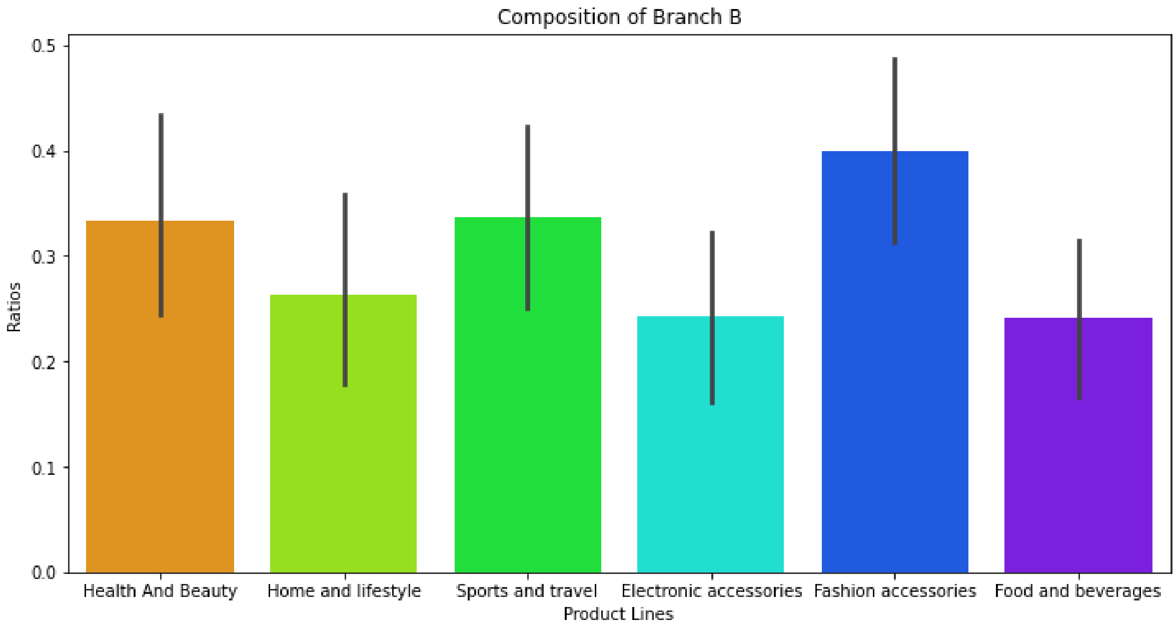
In [26]:
```
#  Composition Ratio of Gender for every Product Line in Test Data
gender_dummies  = pd.get_dummies(testdata_df['Gender'])
gender_dummies.head()
df = pd.concat([testdata_df, gender_dummies], axis = 1)
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'Female', data = df,palette='summer')
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'Male', data = df,palette='autumn')
plt.show()
```

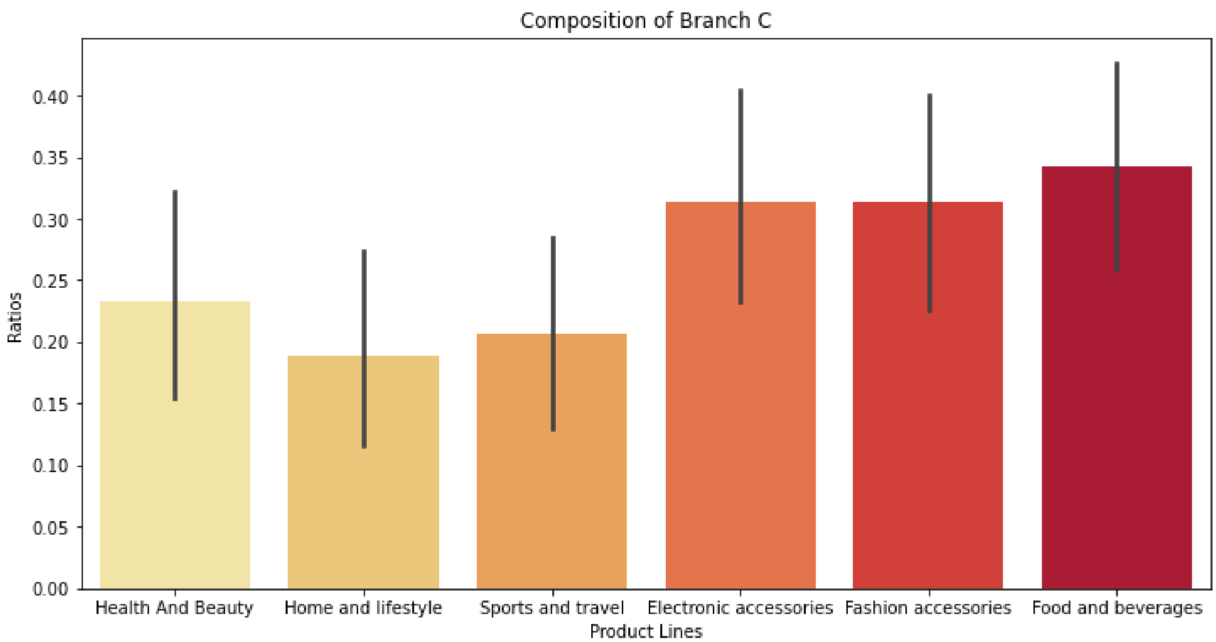




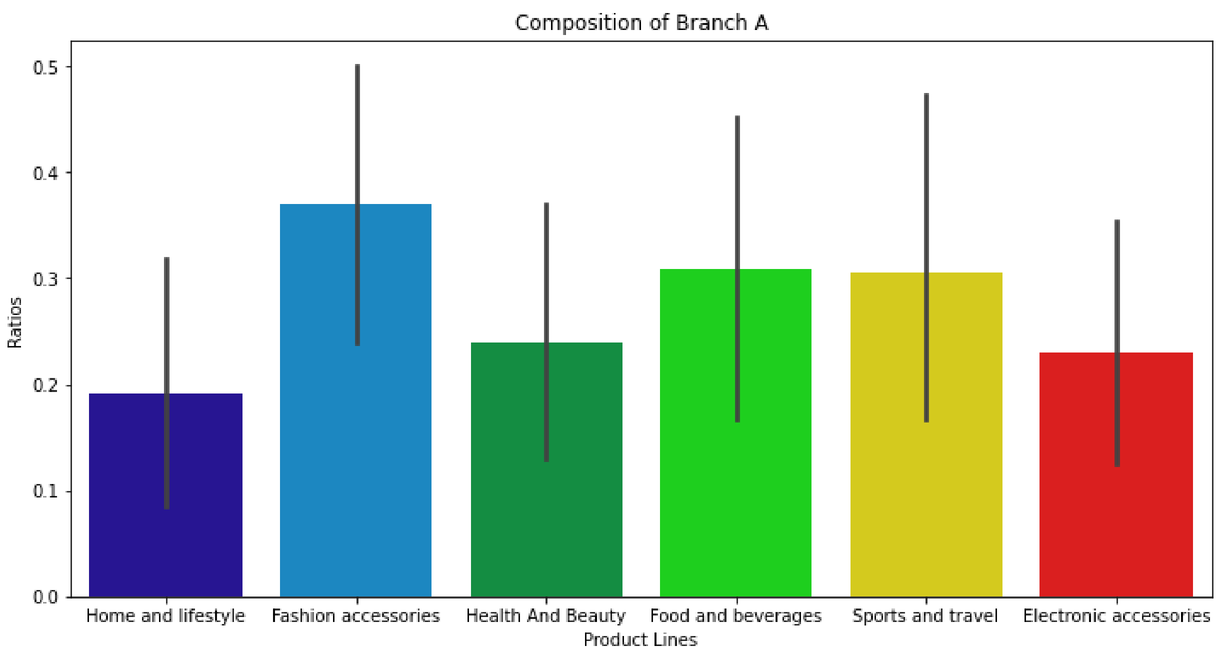
In [27]:
```
#  Composition Ratio of each Branch for every Product Line in Train Data
branch_dummies  = pd.get_dummies(traindata_df['Branch'])
branch_dummies.head()
df = pd.concat([traindata_df, branch_dummies], axis = 1)
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'A', data = df,palette='nipy_spectral')
plt.title("Composition of Branch A")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'B', data = df,palette='gist_rainbow')
plt.title("Composition of Branch B")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'C', data = df,palette='YlOrRd')
plt.title("Composition of Branch C")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
```

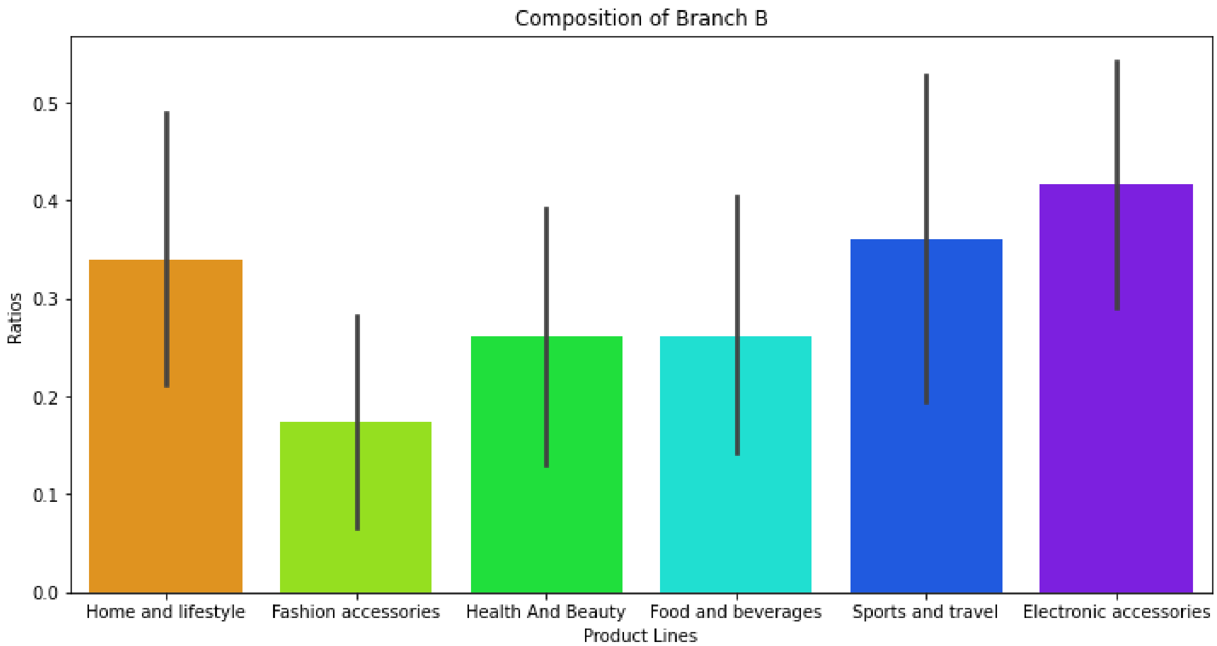


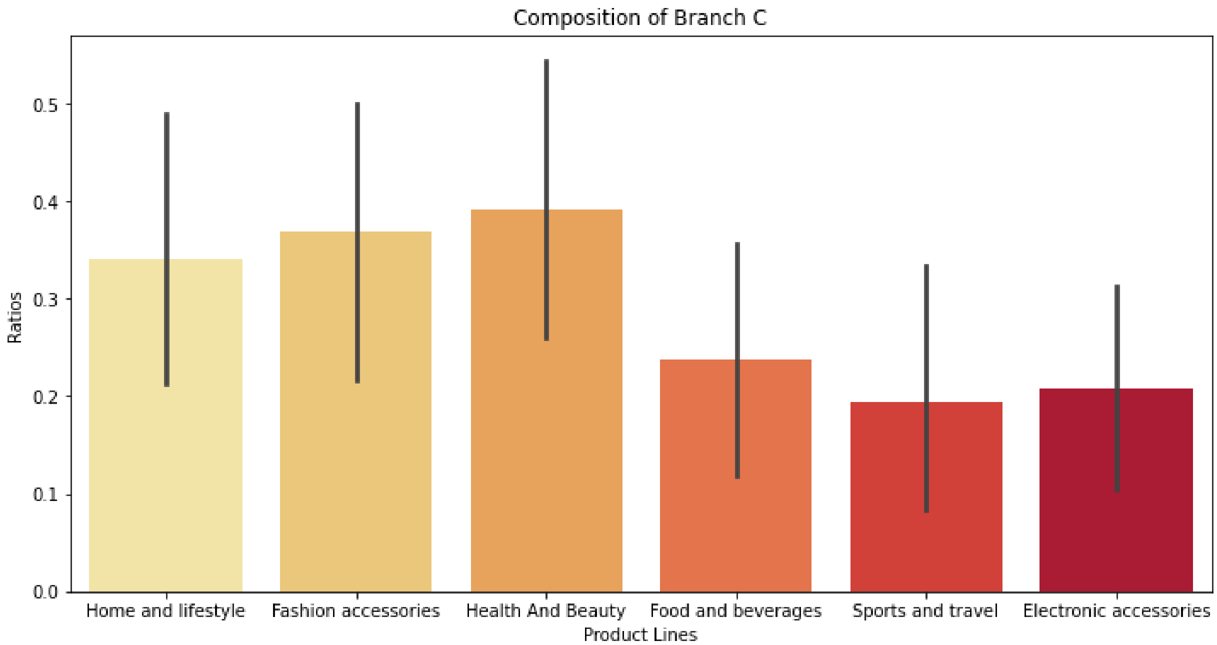

Composition of Branch B
Ratios
0.0
0.1
0.2
0.3
0.4
0.5
Health And Beauty
Home and lifestyle
Sports and travel
Electronic accessories
Fashion accessories
Food and beverages
Product Lines


Composition of Branch C
Ratios
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Health And Beauty
Home and lifestyle
Sports and travel
Electronic accessories
Fashion accessories
Food and beverages
Product Lines

In [28]:
```
#  Composition Ratio of each Branch for every Product Line in Test Data
branch_dummies  = pd.get_dummies(testdata_df['Branch'])
branch_dummies.head()
df = pd.concat([testdata_df, branch_dummies], axis = 1)
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'A', data = df,palette='nipy_spectral')
plt.title("Composition of Branch A")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'B', data = df,palette='gist_rainbow')
plt.title("Composition of Branch B")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
plt.figure(figsize = (12,6))
sns.barplot(x = 'ProductLine', y = 'C', data = df,palette='YlOrRd')
plt.title("Composition of Branch C")
plt.xlabel("Product Lines")
plt.ylabel("Ratios")
plt.show()
```




Composition of Branch B
Ratios
0.0
0.1
0.2
0.3
0.4
0.5
Home and lifestyle
Fashion accessories
Health And Beauty
Food and beverages
Sports and travel
Electronic accessories
Product Lines



In [ ]:

Machine Learning

In [29]:
```
#load data
data = pd.read_csv(r'D:\Academic Material\Semester 3\PAI Lab Assignments\PAI Proj
data.head()
```

Out[29]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785

In [30]:
```
#label encoding

le = LabelEncoder()
Label = ['Customer type','Gender','Payment']

for i in Label:
    data[i] = le.fit_transform(data[i])

data.head()
```

Out[30]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
0	750-67-8428	A	Yangon	0	0	Health and beauty	74.69	7	26.1415	548.9715
1	226-31-3081	C	Naypyitaw	1	0	Electronic accessories	15.28	5	3.8200	80.2200
2	631-41-3108	A	Yangon	1	1	Home and lifestyle	46.33	7	16.2155	340.5255
3	123-19-1176	A	Yangon	0	1	Health and beauty	58.22	8	23.2880	489.0480
4	373-73-7910	A	Yangon	1	1	Sports and travel	86.31	7	30.2085	634.3785

In [31]:
```
#one hot encoding
cols = ['Branch','City','Product line']
# Apply one-hot encoder
OH_encoder = OneHotEncoder(sparse=False)
data_oh = pd.DataFrame(OH_encoder.fit_transform(data[cols])).astype('int64')


#get feature columns
data_oh.columns = OH_encoder.get_feature_names(cols)

# One-hot encoding removed index; put it back
data_oh.index = data.index

# Add one-hot encoded columns to our main df new name: tr_fe, te_fe (means featur
data_fe = pd.concat([data, data_oh], axis=1)
```

In [32]:
```
# Dropping irrelevant columns

data_fe  = data_fe.drop(['Invoice ID', 'Branch', 'City', 'Product line', 'Tax 5%'
```

In [33]:
```
data_fe.head()
```

Out[33]:

	Customer type	Gender	Unit price	Quantity	Payment	gross income	Branch_A	Branch_B	Branch_C	City_M
0	0	0	74.69	7	2	26.1415	1	0	0	
1	1	0	15.28	5	0	3.8200	0	0	1	
2	1	1	46.33	7	1	16.2155	1	0	0	
3	0	1	58.22	8	2	23.2880	1	0	0	
4	1	1	86.31	7	2	30.2085	1	0	0	

Linear Regression

In [34]:
```
y = data_fe['Quantity']
X = data_fe.drop('Quantity', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, randon
```

In [35]:
```
def cross_val(model_name,model,X,y,cv):

    scores = CVS(model, X, y, cv=cv)
    print(f'{model_name} Scores:')
    for i in scores:
        print(round(i*100,2),"%")
    print(f'Average {model_name} score: {round(scores.mean()*100,4)} %')
```

In [36]:
```
#model
LR = LinearRegression()

#fit
LR.fit(X_train, y_train)

#predict
y_predict = LR.predict(X_test)

#score variables
LR_MAE = round(MAE(y_test, y_predict),2)
LR_MSE = round(MSE(y_test, y_predict),2)
LR_R_2 = round(R2(y_test, y_predict),4)
LR_CS  = round(CVS(LR, X, y, cv=5).mean(),4)

print(f" Mean Absolute Error: {LR_MAE}\n")
print(f" Mean Squared Error: {LR_MSE}\n")
print(f" R^2 Score: {LR_R_2*100}%\n")
cross_val(LR,LinearRegression(),X,y,5)
```

```
 Mean Absolute Error: 0.91

 Mean Squared Error: 1.55

 R^2 Score: 81.58%

LinearRegression() Scores:
81.31 %
77.71 %
80.13 %
84.4 %
80.76 %
Average LinearRegression() score: 80.8626 %
```

In [39]:
```
MAE= [LR_MAE]
MSE= [LR_MSE]
R_2= [LR_R_2]
Cross_score= [LR_CS]

Models = pd.DataFrame({
    'odels': ["Linear Regression"],
    'MAE': MAE, 'MSE': MSE, 'R^2':R_2, 'Cross Validation Score':Cross_score})
Models.sort_values(by='MAE', ascending=True)
```

Out[39]:

	Models	MAE	MSE	R^2	Cross Validation Score
0	Linear Regression	0.91	1.55	0.8158	0.8086

K-Means Clustering Algorithm:

In [47]:
```
# standardizing the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_fe)


pd.DataFrame(data_scaled)

# statistics of scaled data
pd.DataFrame(data_scaled).describe()
```

Out[47]:

	0	1	2	3	4	5	
count	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.(
mean	-4.596323e-17	9.292567e-17	-1.187939e-16	5.562217e-17	3.770317e-16	5.173639e-17	
std	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.(
min	-9.980020e-01	-9.980020e-01	-1.721668e+00	-1.543480e+00	-1.205937e+00	-1.270692e+00	
25%	-9.980020e-01	-9.980020e-01	-8.608740e-01	-8.590099e-01	-1.205937e+00	-8.078714e-01	
50%	-9.980020e-01	-9.980020e-01	-1.669588e-02	-1.745399e-01	-1.204733e-03	-2.812422e-01	
75%	1.002002e+00	1.002002e+00	8.406991e-01	8.521652e-01	1.203528e+00	6.037682e-01	1.:
max	1.002002e+00	1.002002e+00	1.672416e+00	1.536635e+00	1.203528e+00	2.928371e+00	1.:

In [64]:
```
# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=6, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)
```

Out[64]: `KMeans(n_clusters=6)`

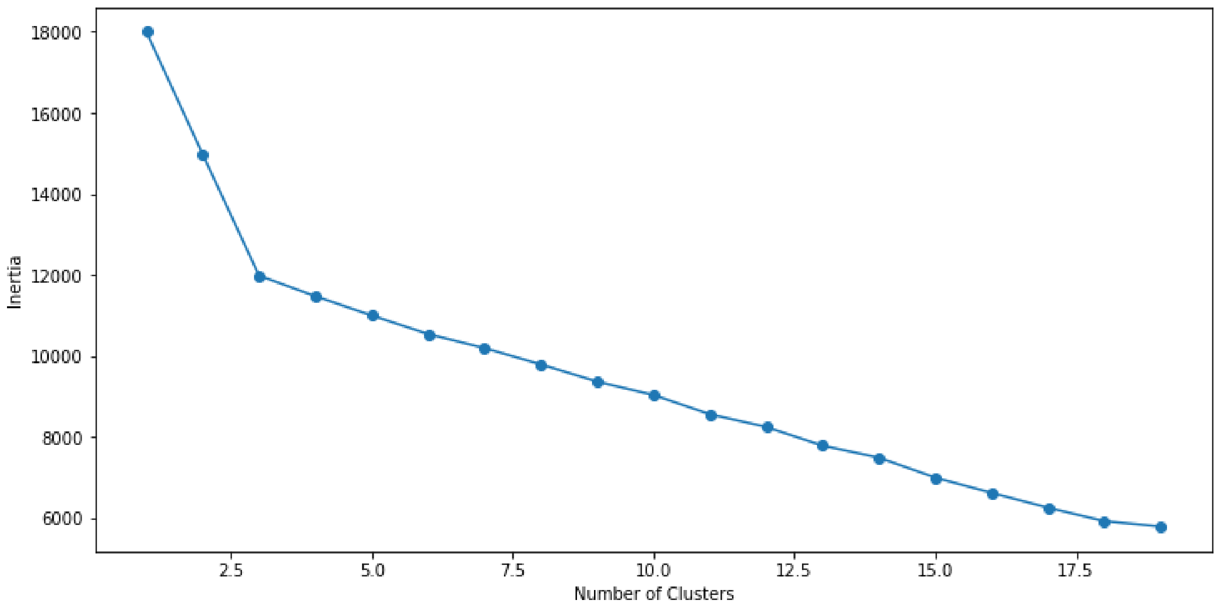
In [60]:
```
# how well clustering has been done through k-means
kmeans.inertia_
```

Out[60]: `1160.611633504169`

In [66]:
```
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_clusters = cluster, init='k-means++')
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
```

Out[66]:
```
Text(0, 0.5, 'Inertia')
```



In [69]:
```
# k means using 5 clusters and k-means++ initialization
kmeans = KMeans(n_clusters = 5, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)
```

In [70]:
```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[70]:
```
0    340
4    203
1    202
3    129
2    126
Name: cluster, dtype: int64
```

In [ ]:
```
```