Annapoornima S

225229101

Lab 15: Exploring Marketing Campaign dataset

In [1]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import pickle

from sklearn.preprocessing import LabelEncoder

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from lightgbm import LGBMRegressor

from sklearn.model_selection import GridSearchCV

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

In [4]:

```
# Reading a Source File
data = pd.read_csv("marketing.csv")
```

In [5]:

```
data.head()
```

Out[5]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer
0	5524	1957	Graduation	Single	58138.0	0	0	2012-09-04
1	2174	1954	Graduation	Single	46344.0	1	1	2014-03-08
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-21
3	6182	1984	Graduation	Together	26646.0	1	0	2014-02-10
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-19

5 rows × 29 columns

In [6]:

```
data.rename(columns={'MntWines': 'Wines',
                     'MntFruits': 'Fruits',
                     'MntMeatProducts': 'Meat',
                     'MntFishProducts': 'Fish',
                     'MntSweetProducts': 'Sweet',
                     'MntGoldProds': 'Gold',
                     'NumDealsPurchases': 'Discount_Purchases',
                     'NumWebPurchases': 'Web_Purchases',
                     'NumCatalogPurchases': 'Catalog_Purchases',
                     'NumStorePurchases': 'Store_Purchases'}, inplace=True)
```

In [7]:

```
data.head()
```

Out[7]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer
0	5524	1957	Graduation	Single	58138.0	0	0	2012-09-04
1	2174	1954	Graduation	Single	46344.0	1	1	2014-03-08
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-21
3	6182	1984	Graduation	Together	26646.0	1	0	2014-02-10
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-19

5 rows × 29 columns

In [8]:

```
le = LabelEncoder()
education_label = le.fit_transform(data['Education'])
data['Education'] = education_label
```

In [9]:

```
marital_staus_label = le.fit_transform(data['Marital_Status'])
data['Marital_Status'] = marital_staus_label
```

In [10]:

```
data[['Z_CostContact', 'Z_Revenue']].describe()
```

Out[10]:

	Z_CostContact	Z_Revenue
count	2240.0	2240.0
mean	3.0	11.0
std	0.0	0.0
min	3.0	11.0
25%	3.0	11.0
50%	3.0	11.0
75%	3.0	11.0
max	3.0	11.0

In [11]:

```
data.drop(columns=['Z_CostContact', 'Z_Revenue'], inplace=True)
```

In [12]:

```
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], dayfirst=True)

data['Day'] = data['Dt_Customer'].apply(lambda x: x.day)
data['Month'] = data['Dt_Customer'].apply(lambda x: x.month)
data['Year'] = data['Dt_Customer'].apply(lambda x: x.year)

data.drop(columns='Dt_Customer', inplace=True)
```

In [13]:

```
mean_income = round(data.groupby('Education')['Income'].mean(), 2)
data['Income'] = data.apply(lambda row: mean_income[row['Education']]
                        if np.isnan(row['Income'])
                        else row['Income'], axis=1)
```

In [14]:

```
# amount the customer spent on all product categories in the last 2 years
data['Total_Products'] = data['Wines'] + data['Fruits'] + data['Meat'] + \
                         data['Fish'] + data['Sweet'] + data['Gold']
```

In [16]:

```
# number of accepted offers for all campaigns
data['Total_Accepted'] = data['AcceptedCmp1'] + data['AcceptedCmp2'] + \
                         data['AcceptedCmp3'] + data['AcceptedCmp4'] + data['AcceptedCmp
```

In [17]:

```
# number of customer purchases
data['Total_Purchases'] = data['Discount_Purchases'] + data['Web_Purchases'] + \
                        data['Catalog_Purchases'] + data['Store_Purchases']
```

In [18]:

```
# number of children
data['Children'] = data['Kidhome'] + data['Teenhome']
```

In [19]:

```
# is the client a parent or not (1 - yes, 0 - no)
data['Parents'] = np.where(data['Children'] > 0, 1, 0)
```

In [20]:

```
data
```

Out[20]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Recency
0	5524	1957	2	4	58138.0	0	0	58
1	2174	1954	2	4	46344.0	1	1	38
2	4141	1965	2	5	71613.0	0	0	26
3	6182	1984	2	5	26646.0	1	0	26
4	5324	1981	4	3	58293.0	1	0	94
...	...	...	...	...	...	...	...	...
2235	10870	1967	2	3	61223.0	0	1	46
2236	4001	1946	4	5	64014.0	2	1	56
2237	7270	1981	2	2	56981.0	0	0	91
2238	8235	1956	3	5	69245.0	0	1	8
2239	9405	1954	4	3	52869.0	1	1	40

2240 rows × 34 columns

In [21]:

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [24]:

```
pip install kneed
```

```
Defaulting to user installation because normal site-packages is not write
able
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.14.2 in c:\programdata\anaconda3
\lib\site-packages (from kneed) (1.21.5)
Requirement already satisfied: scipy>=1.0.0 in c:\programdata\anaconda3\l
ib\site-packages (from kneed) (1.7.3)
Installing collected packages: kneed
Successfully installed kneed-0.8.5Note: you may need to restart the kerne
l to use updated packages.
```

In [25]:

```
from kneed import KneeLocator
```

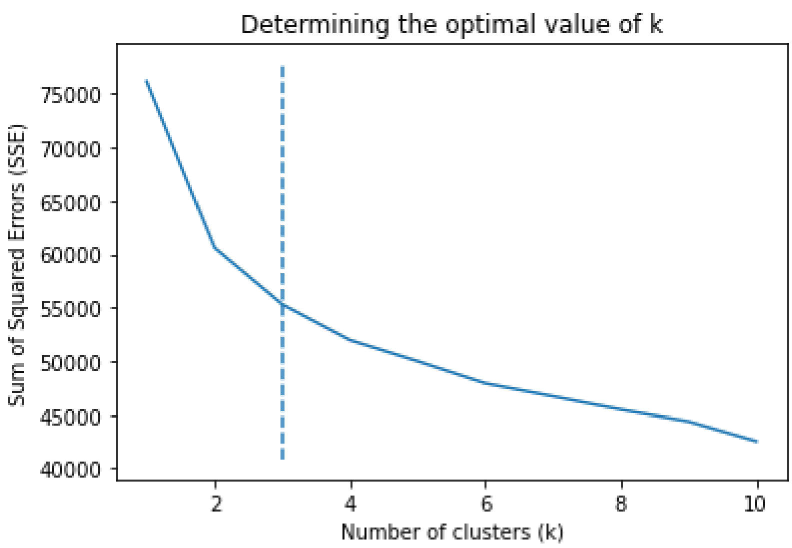
In [26]:

```
# determining the optimal value of K
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    sse.append(kmeans.inertia_)

kl = KneeLocator(range(1, 11), sse, curve="convex", direction="decreasing")
optimal_k = kl.elbow

# result visualization
plt.xlabel('Number of clusters (k)')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.title('Determining the optimal value of k')
plt.plot(range(1, 11), sse)
plt.vlines(optimal_k, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()

print("Optimal number of clusters (k):", optimal_k)
```




```
Optimal number of clusters (k): 3
```

In [27]:

```
# client clustering with K-means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data_scaled)
data['Cluster'] = kmeans.labels_
```

In [28]:

```
# Analyze the resulting groups/clusters
cluster_sizes = data['Cluster'].value_counts()
```

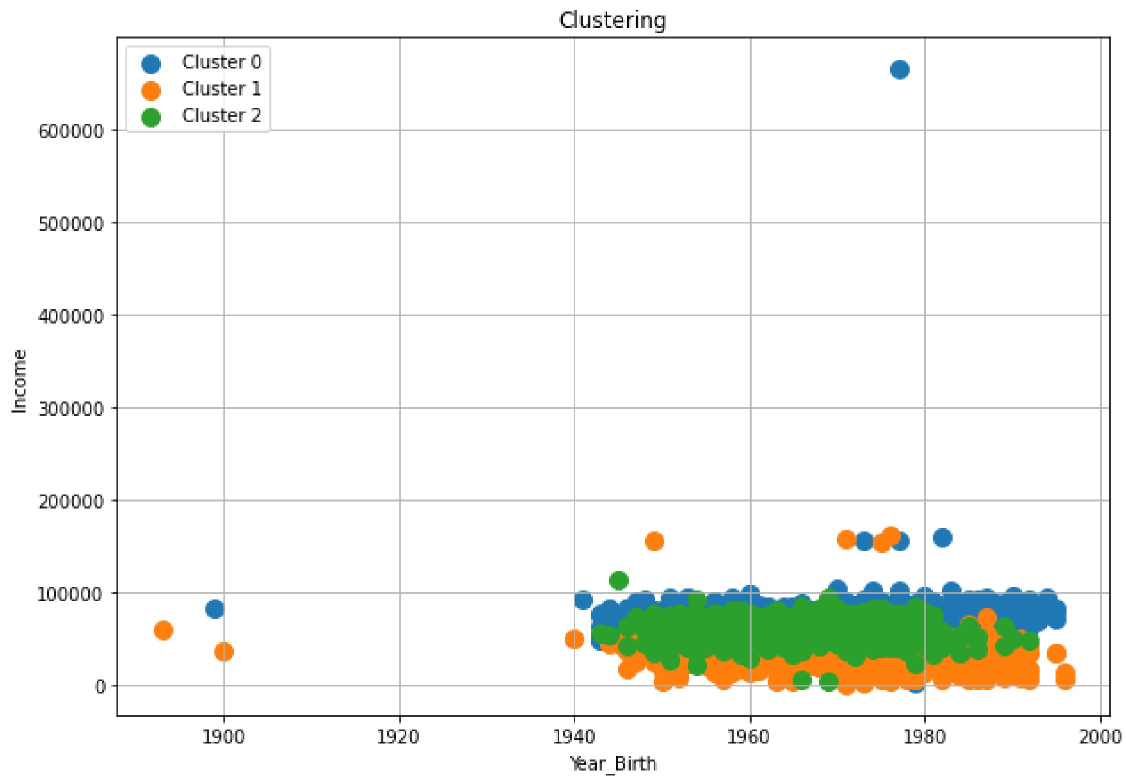
In [29]:

```
# Let's see how many clients are in each group/cluster
for cluster in range(3):
    print(f"Group {cluster} contains {cluster_sizes[cluster]} clients")
```

```
Group 0 contains 514 clients
Group 1 contains 1070 clients
Group 2 contains 656 clients
```

In [30]:

```
fig, ax = plt.subplots(figsize=(10, 7))
legend = []
x_lable='Year_Birth'
y_lable='Income'
ax.set_xlabel(x_lable)
ax.set_ylabel(y_lable)
plt.title('Clustering')
for c, rows in data.groupby('Cluster'):
    plt.scatter(rows[x_lable], rows[y_lable], s = 100)
    legend.append("Cluster %s" % c)
plt.legend(legend, loc="upper left")
plt.grid()
plt.show()
```



In [31]:

```
data.shape
```

Out[31]:

```
(2240, 35)
```

In [32]:

```
# remove noise
data = data[(data['Year_Birth'] > 1900) & (data['Income'] < 600_000)]
```

In [33]:

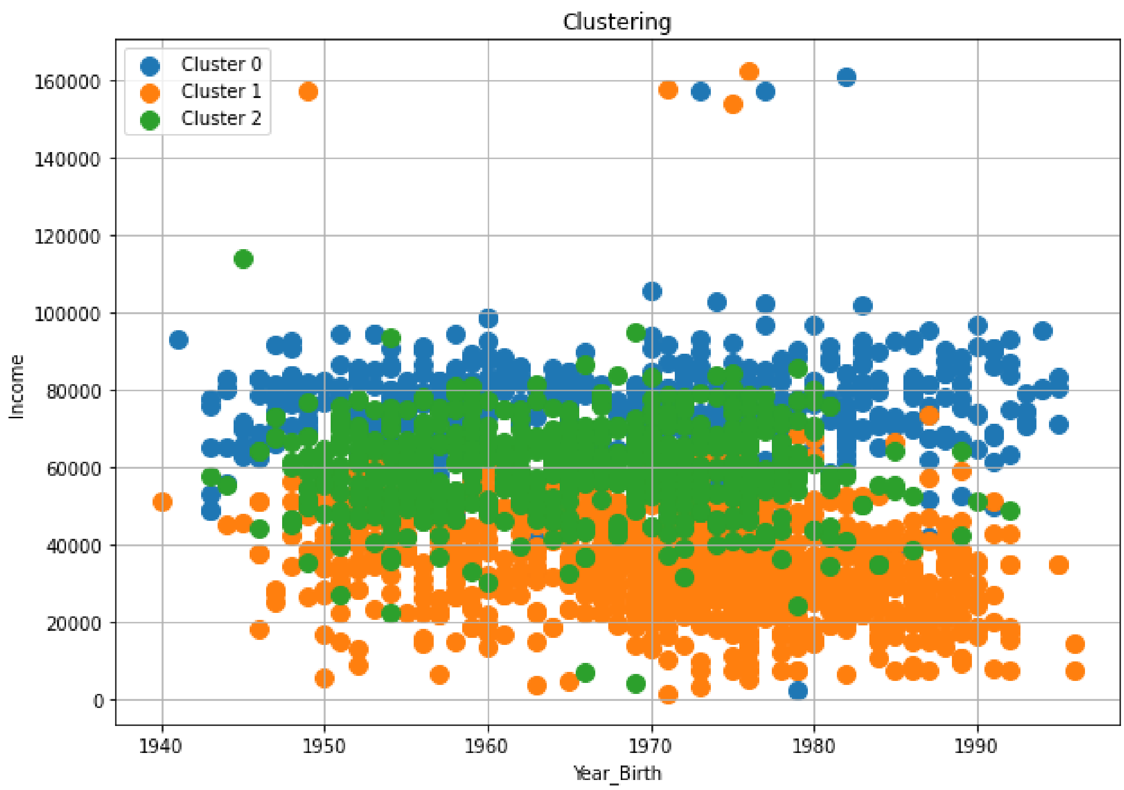
```
data.shape
```

Out[33]:

```
(2236, 35)
```

In [34]:

```
fig, ax = plt.subplots(figsize=(10, 7))
legend = []
x_lable='Year_Birth'
y_lable='Income'
ax.set_xlabel(x_lable)
ax.set_ylabel(y_lable)
plt.title('Clustering')
for c, rows in data.groupby('Cluster'):
    plt.scatter(rows[x_lable], rows[y_lable], s = 100)
    legend.append("Cluster %s" % c)
plt.legend(legend, loc="upper left")
plt.grid()
plt.show()
```



In [35]:

```
cluster_means = data.groupby('Cluster').mean()
cluster_means
```

Out[35]:

Cluster	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenh
0	5684.662109	1968.515625	2.423828	3.765625	76328.001211	0.023438	0.04
1	5601.800562	1971.338951	2.281835	3.722846	35565.028642	0.780899	0.45
2	5493.525915	1965.222561	2.557927	3.711890	59661.036021	0.224085	0.94

3 rows × 34 columns

In [36]:

```
# scale data after removing noise
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [37]:

```
# splitting data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data_scaled[:, :-1],
                                                    data_scaled[:, -1],
                                                    test_size=0.2,
                                                    random_state=42)
```

In [38]:

```
# model training
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[38]:

```
LinearRegression()
```

In [39]:

```
# prediction on the test set
y_pred = model.predict(X_test)
```

In [40]:

```
# evaluation of the quality of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('RMS error:', round(mse, 3))
print('R^2 score:', round(r2, 3))
```

```
RMS error: 0.295
R^2 score: 0.703
```

In [41]:

```
# separate train and test data
X_train, X_test, y_train, y_test = train_test_split(data.drop("Cluster", axis=1),
                                                    data["Cluster"],
                                                    test_size=0.2,
                                                    random_state=42)
```

In [42]:

```
# scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [43]:

```
# train the model
model = LGBMRegressor()
model.fit(X_train, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000576 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2164
[LightGBM] [Info] Number of data points in the train set: 1788, number of
used features: 33
[LightGBM] [Info] Start training from score 1.069351
[LightGBM] [Warning] No further splits with positive gain, best gain: -in
f
[LightGBM] [Warning] No further splits with positive gain, best gain: -in
f
```

Out[43]:

```
LGBMRegressor()
```

In [44]:

```
# Predict on test set
y_pred = model.predict(X_test)
```

In [45]:

```
# Assess the quality of the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMS error:", round(mse, 3))
print("R^2 Score:", round(r2, 3))
```

```
RMS error: 0.069
R^2 Score: 0.866
```

In [46]:

```
data.shape
```

Out[46]:

```
(2236, 35)
```

In [47]:

```
pd.DataFrame(data={'features': data.drop(columns='Cluster').columns,
                   'importances': model.feature_importances_}).sort_values(by='importanc
                                                                          ascending=Fal
```

Out[47]:

	features	importances
29	Total_Products	280
4	Income	251
8	Wines	227
12	Sweet	194
11	Fish	187
10	Meat	186
9	Fruits	145
13	Gold	137
31	Total_Purchases	120
15	Web_Purchases	113

In [48]:

```
# Defining the hyperparameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 1],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'num_leaves': [10, 20, 30]
}
```

In [49]:

```
# Selection of hyperparameters using GridSearchCV
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
of testing was 0.000293 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2115
[LightGBM] [Info] Number of data points in the train set: 1431, number
of used features: 33
[LightGBM] [Info] Start training from score 1.066387
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000272 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2122
[LightGBM] [Info] Number of data points in the train set: 1431, number
of used features: 32
[LightGBM] [Info] Start training from score 1.045423
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000299 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2118
[LightGBM] [Info] Number of data points in the train set: 1430, number
of used features: 33
[LightGBM] [Info] Start training from score 1.074126
```

In [50]:

```
# Display results
print("Best params:", grid_search.best_params_)
print("Best score:", round(-grid_search.best_score_, 3))
```

```
Best params: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200,
'num_leaves': 30}
Best score: 0.063
```

In [51]:

```
# Assessing model quality on test data
y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMS error:", round(mse, 3))
print("R^2 score:", round(r2, 3))
```

```
RMS error: 0.071
R^2 score: 0.862
```

In [ ]: