Solution - Market Research

Solution: Millennial Market Research

Import Libraries and Dataset

In [1]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 50)

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

In [2]:
```
df = pd.read_csv('data/millenial_market_research.csv')
```

Exploratory Analysis and Data Cleaning

We'll start by "getting to know" the dataset. The goal is to understand the dataset at a qualitative level, note anything that should be cleaned up, and spot opportunities for feature engineering.

Tip: In our Machine Learning Masterclass, we separate exploratory analysis and data cleaning into separate steps. However, because take-home challenges are meant to be shorter/condensed analyses, feel free to combine these steps. Just remember to keep your code clean and organized.

In [3]:
```
# Example observations
df.head()
```

Out[3]:

	Age	Gender	Music	Movies/Theaters	Tech/Gadgets	Museums	Food/Dining	Camping/Hiking	Conc
0	17.0	male	7.3	8.1	2.8	1.6	4.5	7.1	0.3
1	21.0	female	9.4	9.3	2.2	2.2	3.2	9.5	5.4
2	19.0	female	6.8	7.5	6.4	2.1	7.8	4.4	1.9
3	26.0	female	4.5	6.8	1.3	8.5	8.0	7.4	1.5
4	19.0	female	9.1	9.8	1.4	3.9	3.1	5.4	8.2

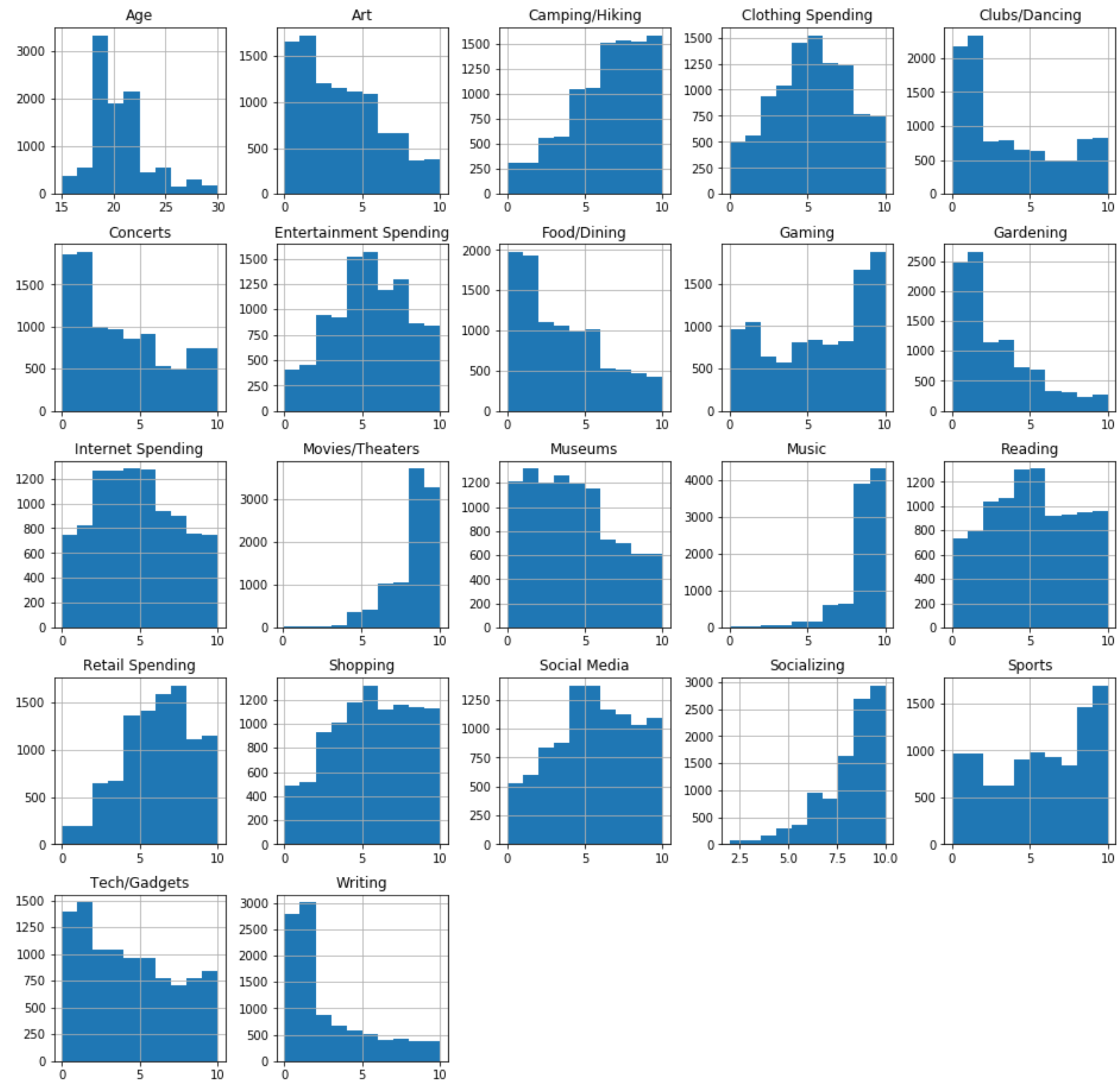
First, we look at some examples observations to get a "feel" for the dataset. We learn that:

- The dataset has a mostly numeric features, and only one categorical feature.

- The dataset has some features that are irrelevant to our analysis (`'Entertainment Spending'` and `'Retail Spending'`).
- The values for the Interests appear to range from 0 to 10. We'll confirm this in a moment.

Next, we'll dive deeper into the distributions and statistics of our features.

In [4]:
```
# Numeric feature distributions
df.hist(figsize=(16,16))
plt.show()
```



In [5]:
```
# Numeric feature summary statistics
df.describe()
```

Out[5]:

	Age	Music	Movies/Theaters	Tech/Gadgets	Museums	Food/Dining	Campi
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.
mean	20.386600	8.485650	8.221250	4.334410	4.226790	3.51436	6.3396

	Age	Music	Movies/Theaters	Tech/Gadgets	Museums	Food/Dining	Campi
std	2.780596	1.401132	1.488302	2.922271	2.702377	2.68117	2.4712
min	15.000000	0.000000	0.100000	0.000000	0.000000	0.00000	0.0000
25%	19.000000	8.200000	7.600000	1.700000	1.900000	1.20000	4.7000
50%	20.000000	8.800000	8.600000	4.000000	4.000000	2.90000	6.7000
75%	21.000000	9.400000	9.300000	6.700000	6.125000	5.40000	8.3000
max	30.000000	10.000000	10.000000	10.000000	10.000000	10.00000	10.000

In [6]:
```
# Categorical feature summary statistics
df.describe(include=['object'])
```

Out[6]:

	Gender
count	10000
unique	2
top	female
freq	5877

A few key learnings:

- First, the dataset includes observations that are **not** in our target audience. We should remove these observations before continuing our analysis.
- Second, the `Gender` feature only has two values, `'female'` and `'male'`. We should convert this into an indicator feature with `0` or `1`.
- Finally, It doesn't look like any distributions are out of the ordinary, and we don't have any missing values.

Before continuing, we will filter our dataset to just our target audience.

In [7]:
```
# Filter to our target audience
df = df[(df['Clothing Spending'] > 5) & (df['Internet Spending'] > 5)]
df.shape
```

Out[7]:
```
(2889, 23)
```

Feature Engineering

For this challenge, feature engineering is pretty simple. First, we'll create an indicator feature for `Female` to replace the categorical feature of `Gender`.

In [8]:
```
# Create indicator feature for Female
```

```
df.rename(columns={'Gender':'Female'}, inplace=True)
df.Female.replace({'female':1, 'male':0}, inplace=True)
```

Then, we'll drop the four behavioral features, since we wish to create marketing clusters based on interests and demographics.

Now we have our **analytical base table** (ABT).

In [9]:
```
abt = df.drop(['Entertainment Spending',
               'Clothing Spending',
               'Internet Spending',
               'Retail Spending'], axis=1)
```

PCA

Next, we'll create Principal Components and keep enough to capture at least 80% of the variance in the dataset. We've already done the heavy lifting, so fitting the PCA transformation is simple:

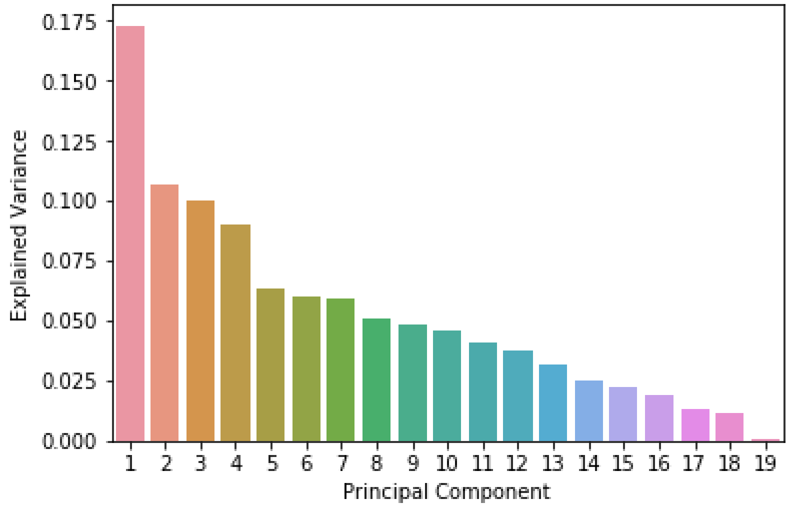
In [10]:
```
from sklearn.decomposition import PCA
```

In [11]:
```
pca = PCA()
pca.fit( abt )
```

Out[11]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=Nc
  svd_solver='auto', tol=0.0, whiten=False)
```

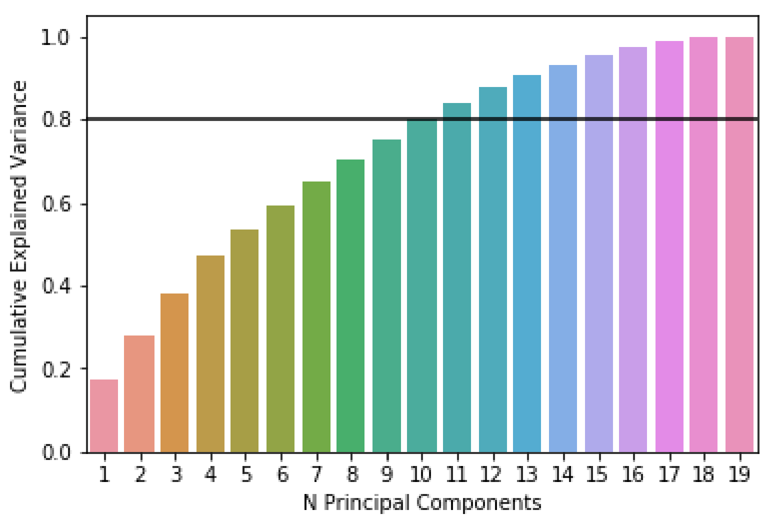
We can take a look how much variance each PC explains.

In [12]:
```
# Explained Variance Ratio
sns.barplot(x=np.arange(pca.n_components_) + 1,
            y=pca.explained_variance_ratio_)
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.show()
```



To decide on the number of PC's to keep, we'll plot the cumulative explained variances.

In [13]:
```
# Cumulative Explained Variance
cumulative = np.cumsum(pca.explained_variance_ratio_)
sns.barplot(x=np.arange(pca.n_components_) + 1,
            y=cumulative)
plt.xlabel('N Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.8, color='k', linestyle='-')
plt.show()
```



It looks like 10 might barely be enough, but let's double-check that.

(Remember that Python is 0-indexed, so `cumulative[9]` is actually the sum of **10** PC's.

In [14]:
```
# 10 Principal Components
cumulative[9]
```

Out[14]: 0.798546055121548

Ah. It looks like 10 PC's doesn't satisfy our requirements, so we'll need to use the first 11 components for our clustering model.

Clustering

Now we're ready to train our clustering model. There are multiple viable clustering algorithms, but the standard K-Means algorithm works fine for most use-cases. It's fast, scalable to large datasets, and produces balanced clusters.

In [15]:
```
from sklearn.cluster import KMeans
```

Remember, we're training the clustering model based on the principal components (PC's), and not the original features.

That means we need to create a new dataframe with the transformed features:

In [16]:
```
# PCA transformation
pc_df = pd.DataFrame( pca.transform(abt) )

# Rename Columns
pc_df.columns = ['PC{}'.format(n+1) for n in np.arange(pca.n_components_)

pc_df.head()
```

Out[16]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC1
0	-4.718882	-2.960997	1.116481	2.548510	-0.199757	-1.365595	-0.791734	-5.314410	2.321042	-2.1
1	0.491975	1.380983	1.128373	6.301044	-0.359154	0.709515	-1.014473	1.097250	1.166798	0.15
2	-2.770408	-4.774232	1.598379	0.078869	-0.386427	-0.448358	-1.733045	-1.781863	-2.426829	-2.1
3	-3.041096	-3.171436	0.658393	2.253144	0.320050	-0.796833	-2.609572	-2.273643	-1.241404	-5.5
4	2.375304	1.511895	-0.941456	-0.373477	1.079997	-4.262592	0.463675	7.823742	0.350148	1.69

We'll keep just the first 11 for our training data.

In [17]:
```
# Create training set
pcs_to_keep = ['PC{}'.format(n+1) for n in np.arange(11)]
X_train = pc_df[pcs_to_keep]

# Train K-Means clustering algorithm
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
```

Out[17]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

Function

Finally, writing the deliverable function is straightforward, but prone to errors of omission. The key is that you must include every single step in which you transformed the dataset in any way. Another pitfall is accidentally including some object from above inside the function (since the function should be stand-alone).

Tip: To check your function, try saving your trained models to your computer (using pickle or some other library) re-starting your script, importing your libraries and models, and then skipping straight to your function. Your function should be able to run on its own.

In [18]:
```python
def predict_clusters(raw_data, trained_pca, trained_kmeans, n_pc):
    df_new = raw_data.copy()

    # Filter to our target audience
    df_new = df_new[(df_new['Clothing Spending'] > 5)
                    & (df_new['Internet Spending'] > 5)]

    # Engineer Features
    df_new.rename(columns={'Gender':'Female'}, inplace=True)
    df_new.Female.replace({'female':1, 'male':0}, inplace=True)
    abt_new = df_new.drop(['Entertainment Spending',
                           'Clothing Spending',
                           'Internet Spending',
                           'Retail Spending'], axis=1)

    # PCA transformation
    pc_df_new = pd.DataFrame( trained_pca.transform(abt_new) )

    # Rename Columns
    pc_df_new.columns = ['PC{}'.format(n+1)
                         for n in np.arange(trained_pca.n_components_)]

    # Create test set
    pcs_to_keep = ['PC{}'.format(n+1) for n in np.arange(n_pc)]
    X_new = pc_df_new[pcs_to_keep]

    # Predict clusters
    df_new['Cluster'] = trained_kmeans.predict(X_new)

    return df_new[['Age', 'Female', 'Clothing Spending', 'Internet Spendi
```

We'll test this with a new, raw dataset.

In [19]:
```
raw_df = pd.read_csv('data/unseen_raw_data.csv')

raw_df.head()
```

Out[19]:

	Age	Gender	Music	Movies/Theaters	Tech/Gadgets	Museums	Food/Dining	Camping/Hiking	Con
0	21.0	female	9.5	7.2	0.8	3.5	3.8	9.1	0.8
1	23.0	female	9.1	8.6	3.3	5.5	3.8	6.5	6.1
2	19.0	female	8.2	9.5	0.9	4.3	4.1	9.9	2.1
3	19.0	female	8.8	6.3	0.1	3.9	3.2	9.4	9.5
4	22.0	female	9.6	9.9	4.5	5.6	1.0	6.8	1.2

In [20]:
```
pred_df = predict_clusters(raw_df, pca, kmeans, 11)

pred_df
```

Out[20]:

	Age	Female	Clothing Spending	Internet Spending	Cluster
0	21.0	1	7.8	5.7	0
11	20.0	0	5.3	6.7	0
17	18.0	1	8.4	5.9	0
21	28.0	1	7.4	7.3	2
26	18.0	0	5.4	5.1	1
28	22.0	1	6.4	6.7	2
29	19.0	0	6.0	8.9	1
33	30.0	0	6.3	6.1	2
47	26.0	0	7.5	7.1	2
48	23.0	1	7.2	6.0	0
49	17.0	1	9.9	8.7	2