Project Instructions

Help AZ Watch improve their marketing strategies by answering the following questions about their subscribers.

What is the predicted subscriber churn for AZ Watch?

- Apply the necessary processing steps to prepare the data for modeling.
- Train three standard classification models and save these as model1, model2, and model3.
- Aim to acheive a minimum accuracy of 90% for at least one of the models. Save your best accuracy score as score.

AZ Watch wants to establish more personalized and targeted campaigns to reduce subscriber churn. What subscriber segments can be identified in the data to make their campaigns more targeted?

- Apply a standard clustering method to the numerical features X. Analyze the average values by cluster_id for these numerical features and store them in analysis, rounding values to the nearest whole number.

Project Background

AZ Watch is a popular video streaming platform specialized in educational content, where creators publish online video tutorials and lessons about any topic, from speaking a new language to cooking to learning to play a musical instrument.

Their next goal is to leverage AI-driven solutions to analyze and make predictions about their subscribers and improve their marketing strategy around attracting new subscribers and retaining current ones. This project uses machine learning to predict subscribers likely to churn and find customer segments. This may help AZ Watch find interesting usage patterns to build subscriber personas in future marketing plans!

The `data/AZWatch_subscribers.csv` **dataset** contains information about subscribers and their status over the last year:

Column name	Description
`subscriber_id`	The unique identifier of each subscriber user
`age_group`	The subscriber's age group
`engagement_time`	Average time (in minutes) spent by the subscriber per session
`engagement_frequency`	Average weekly number of times the subscriber logged in the platform (sessions) over a year period

Column name	Description
`subscription_status`	Whether the user remained subscribed to the platform by the end of the year period (subscribed), or unsubscribed and terminated her/his services (churned)

Carefully observe and analyze the features in the dataset, asking yourself if there are any **categorical attributes** requiring pre-processing?

The subscribers dataset from the `data/AZWatch_subscribers.csv` file is already being loaded and split into training and test sets for you:

In [5]:
```
# Import the necessary modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.cluster import KMeans
import seaborn as sns
from matplotlib import pyplot as plt

# Specify the file path of your CSV file
file_path = "../data/subscribers_data.csv"

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)
df
```

Out[5]:

	subscriber_id	age_group	engagement_time	engagement_frequency	subscriptic
0	14451	18-34	5.55	7	s
1	18386	under 18	5.12	12	s
2	12305	35 and over	4.25	4	
3	17546	18-34	8.54	15	s
4	15399	18-34	12.12	20	s
...	...	...	...	...	
995	17439	35 and over	3.12	23	s
996	10112	35 and over	5.25	8	s
997	10692	35 and over	2.37	5	
998	11164	18-34	8.19	8	s
999	14958	35 and over	3.78	8	

1000 rows × 5 columns

In [6]:
```
# Separate predictor variables from class label
X = df.drop(['subscriber_id','subscription_status'], axis=1)
y = df.subscription_status

# Split intro training and test sets (20% test)
X_train, X_test, y_train, y_test = train_test_split(
                        X, y, test_size=.2, random_state=42)

# Data processing: Apply One Hot Encoding on the categorical attribute: age_
X_train_prepared = pd.get_dummies(X_train, columns=['age_group'])

# Data processing:Apply the same one hot encoding transformation on the test
X_test_prepared = pd.get_dummies(X_test, columns=['age_group'])
```

In [7]:
```
# LOGISTIC REGRESSION CLASSIFIER
# Train a logistic regression classifier for subscriber churn prediction
model1 = LogisticRegression()
model1.fit(X_train_prepared, y_train)

# Calculate accuracy score of predictions on test set
score = model1.score(X_test_prepared, y_test)
print("\nLogistic regression accuracy score: ", score)
```

```
Logistic regression accuracy score:  0.925
```

In [8]:
```
# DECISION TREE CLASSIFIER
# Train a decision tree classifier for subscriber churn prediction
```

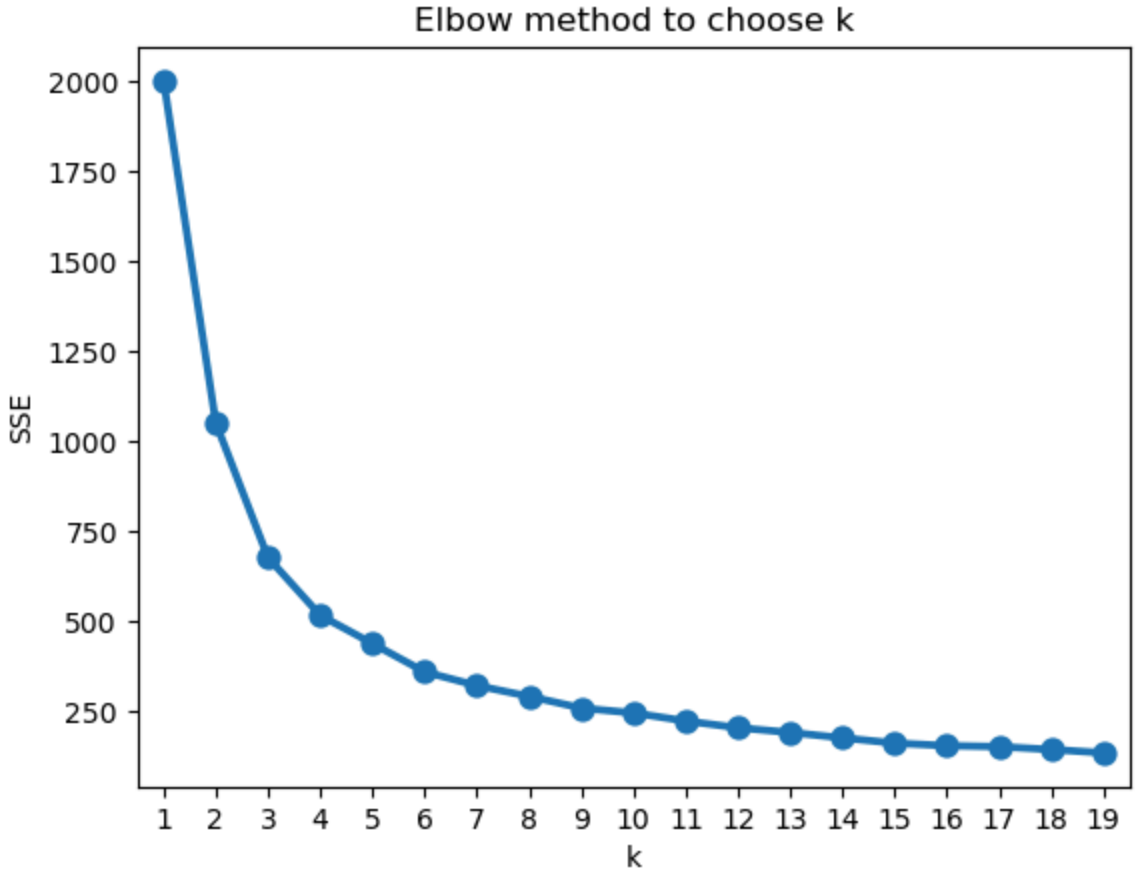
```
model2 = DecisionTreeClassifier(max_depth=3, criterion="gini")
model2.fit(X_train_prepared, y_train)

# Calculate decision tree's accuracy score of predictions on test set
score = model2.score(X_test_prepared, y_test)
print("\nDecision tree accuracy score: ", score)
```

```
Decision tree accuracy score:  0.92
```

In [9]:
```
# RANDOM FOREST ENSEMBLE
# Train a random forest ensemble classifier for subscriber churn prediction
model3 = RandomForestClassifier(n_estimators = 10, max_depth=3)
model3.fit(X_train_prepared, y_train)

# Calculate ensemble's accuracy score of predictions on test set
score = model3.score(X_test_prepared, y_test)
print("\nRandom Forest accuracy score: ", score)
```

```
Random Forest accuracy score:  0.915
```

In [10]:
```
# SUBSCRIBER SEGMENTATION
# You can optionally use a method like the elbow criterion and silhouette ca
segmentation = X.drop(['age_group'], axis=1)

# Scale the two numerical data attributes
scaler = StandardScaler()
scaler.fit(segmentation)
segmentation_normalized = scaler.transform(segmentation)

sse = {} # sum of squared errors (distances) to each cluster
for k in range(1,20):
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(segmentation_normalized)
    sse[k] = kmeans.inertia_

plt.title('Elbow method to choose k')
plt.xlabel('k');plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()

# Apply k-means clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=1)
kmeans.fit_predict(segmentation_normalized)

# Add cluster labels as a new attribute in the dataset before scaling
segmentation["cluster_id"] = kmeans.labels_

# Analyze average feature values and counts per cluster
analysis = segmentation.groupby(['cluster_id']).agg({
    'engagement_time': ['mean'],
    'engagement_frequency':['mean']
}).round(0)
```


Elbow method to choose k
SSE
2000
1750
1500
1250
1000
750
500
250
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
k