How can we accurately predict churn rates for Netflix users?

Stakeholders: Netflix , Netflix users, Investors, Competitors and Researchers

What is a churn rate?
The annual percentage rate at which customers stop subscribing to a service or employees leave a job.

Churn rate for this dataset is around ~80% meaning **4 out of 5** customers don't stay for a full year or more.

Why is this important?

This is important because multiple parties are involved, it helps **Netflix** know how many of their users they are retaining, it helps the **users** know that their streaming service is interested in keeping them as customers and therefore interested in enhancing user experience and lastly is important to us **researchers** to know what makes a person to stay as paying customer or not in the biggest streaming platform in the world, is it their age? Their country? Their income?

Churn rates were defined in this matter:

1. Customer stayed for x >= 365 days then customer **did not churn.**
2. Customer stayed for x < 365 days then customer **did churn.**

Raw Dataset + Data Cleaning

	User ID	Subscription Type	Monthly Revenue	Join Date	Last Payment Date	Country	Age	Gender	Device	Plan Duration
0	1	Basic	10	15-01-22	10-06-23	United States	28	Male	Smartphone	1 Month
1	2	Premium	15	05-09-21	22-06-23	Canada	35	Female	Tablet	1 Month
2	3	Standard	12	28-02-23	27-06-23	United Kingdom	42	Male	Smart TV	1 Month
3	4	Standard	12	10-07-22	26-06-23	Australia	51	Female	Laptop	1 Month
4	5	Basic	10	01-05-23	28-06-23	Germany	33	Male	Smartphone	1 Month

Important notes about dataset:

1. The **granularity** of the data is costumer, meaning that each row represents one individual customer, therefore we need to drop user ID in the preprocessing part.
2. Since Netflix only offers subscription on a monthly basis (they don't offer multiple months or yearly subscription) we can drop Plan Duration since all entries the same. ('1 Month')
3. Instead of using Join Date and Last Payment Dates as features we transformed it to a binary feature ('Yes', 'No') called **'Churned'**

Methods

Data Preprocessing/ Data Cleaning

```python
# CREATING A CLEANED DATAFRAME WITH ONLY USEFUL FEATURES
df['Last Payment Date'] = pd.to_datetime(df['Last Payment Date'], format = "%d-%m-%y")

df['Join Date'] = pd.to_datetime(df['Join Date'], format = "%d-%m-%y")

max_value = max(df['Last Payment Date'])
min_value = min(df['Last Payment Date'])


df['Churned'] = df.apply(lambda r : 'No' if (r['Last Payment Date'] - r['Join Date']).days >= 365 else 'Yes', axis=1)
data_training = df.drop(columns= ['Plan Duration', 'User ID', 'Join Date', 'Last Payment Date'])
data_training
```

Visualize

	Subscription Type	Monthly Revenue	Country	Age	Gender	Device	Churned
	object Basic 40% Standard 30.7% Premium 29.3%	int64 10 - 15	object United Stat... 18% Spain 18% 8 others 63.9%	int64 26 - 51	object Female 50.3% Male 49.7%	object Laptop 25.4% Tablet 25.3% 2 others 49.2%	object Yes 82.2% No 17.8%
0	Basic	10	United States	28	Male	Smartphone	No
1	Premium	15	Canada	35	Female	Tablet	No
2	Standard	12	United Kingdom	42	Male	Smart TV	Yes
3	Standard	12	Australia	51	Female	Laptop	Yes
4	Basic	10	Germany	33	Male	Smartphone	Yes
5	Premium	15	France	29	Female	Smart TV	No
6	Standard	12	Brazil	46	Male	Tablet	No
7	Basic	10	Mexico	39	Female	Laptop	Yes
8	Standard	12	Spain	37	Male	Smartphone	Yes
9	Premium	15	Italy	44	Female	Smart TV	Yes

1. Putting all dates into a ‘readable’ format using pd.to_datetime.
2. Using a lambda to determine churned rates.
3. Dropping 4 features: Plan Duration, User ID, Join Date, Last Payment Date.

K-means

Why K-means?

Key Points:

- **Clustering Goal:**
  - Segment Netflix users into distinct groups based on behavioral and demographic data for actionable insights.
- **Advantages of K-Means:**
  - Simplicity: Easy to implement and computationally efficient for the dataset size.
  - Interpretability: Clear cluster assignments make it straightforward to analyze user segments.
  - Scalability: Handles a medium-sized dataset (2,500 rows) effectively while maintaining speed.
- **Alignment with Project Goals:**
  - Groups users with similar attributes to understand churn patterns and identify high-risk segments.
  - Provides actionable insights for personalized marketing and retention strategies.

K-means

Data Pre-Processing

Steps Taken:

- Cleaned missing data and converted date fields to datetime.
- Applied ohe for categorical features (country, device).
- Scaled numerical features (e.g., age, revenue) using StandardScaler.

Purpose: Ensure the dataset was prepared for clustering and machine learning.

```
netflix_data['Join Date'] = pd.to_datetime(netflix_data['Join Date'], format='%d-%m-%y')
netflix_data['Last Payment Date'] = pd.to_datetime(netflix_data['Last Payment Date'], format='%d-%m-%y')

duplicates_count = netflix_data.duplicated().sum()

updated_data_types = netflix_data.dtypes
updated_data_types, duplicates_count
```

This output has been hidden. Show it.

```
encoded_data = pd.get_dummies(netflix_data, columns=[
    'Subscription Type', 'Country', 'Gender', 'Device', 'Plan Duration'], drop_first=True)

encoded_data.head()
```

This output has been hidden. Show it.

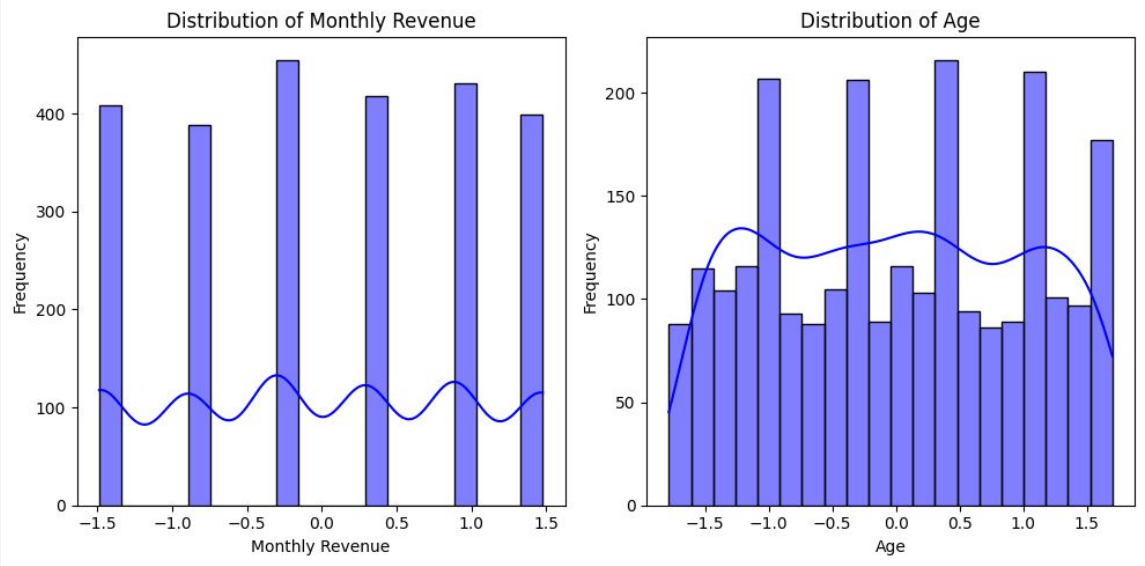
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numerical_columns = ['Age', 'Monthly Revenue']
encoded_data[numerical_columns] = scaler.fit_transform(encoded_data[numerical_columns])
final_data = encoded_data.drop(columns=['User ID', 'Join Date', 'Last Payment Date'])

final_data.head()
```

This output has been hidden. Show it.

K-means

Data Visualizations

- **Visual Insights:**
  - Histograms: Highlighted distributions of age and monthly revenue.
  - Heatmap: Showed relationships between features.
  - PCA and t-SNE: Revealed distinct user clusters and dimensionality reduction.
- **Implication**: Identified separable user groups, supporting clustering feasibility.

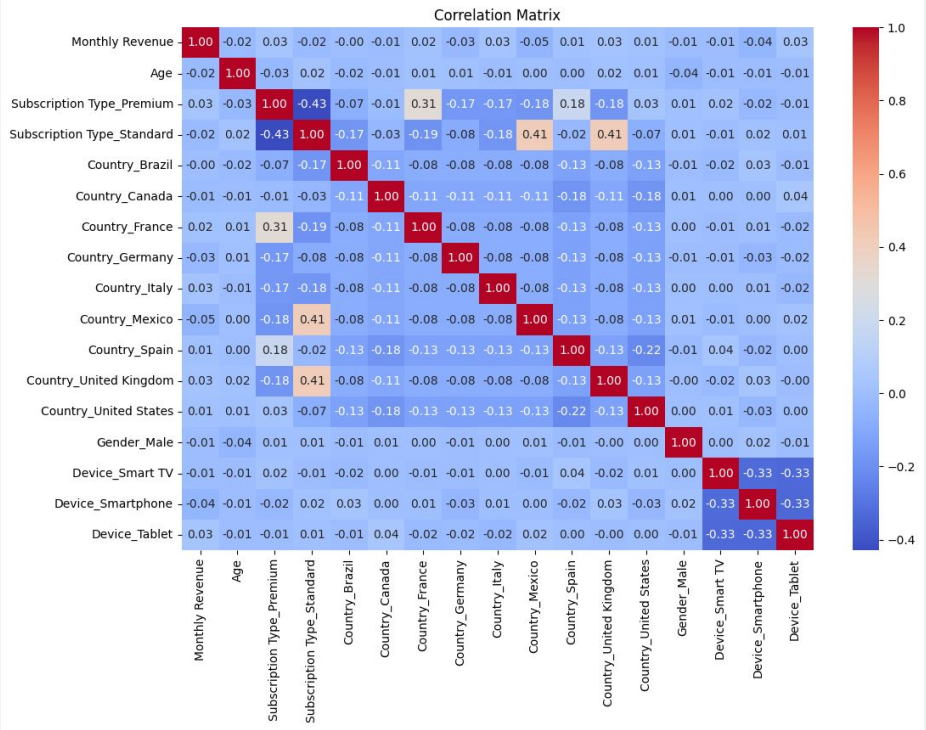


K-means

- **Visual Insights:**
  - Histograms: Highlighted distributions of age and monthly revenue.
  - Heatmap: Showed relationships between features.
  - PCA and t-SNE: Revealed distinct user clusters and dimensionality reduction.

- **Implication:** Identified separable user groups, supporting clustering feasibility.

Data Visualizations

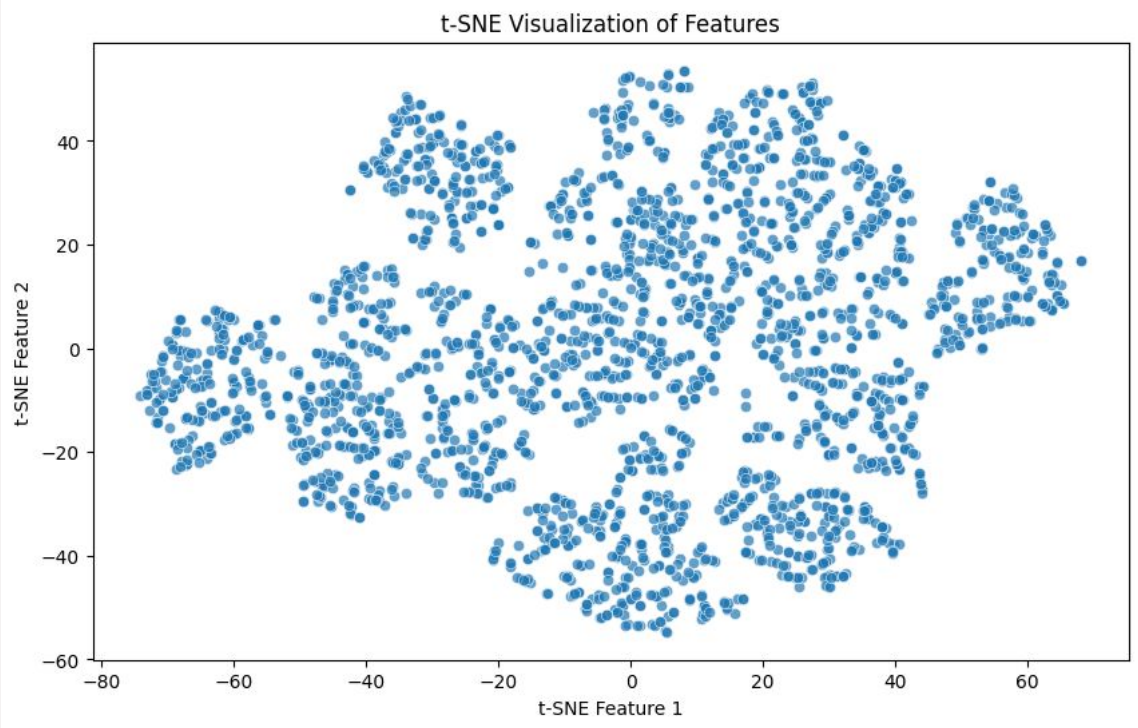


K-means

Data Visualizations

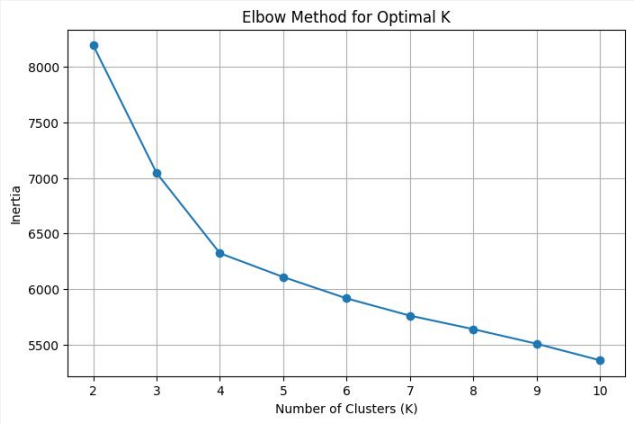
- **Visual Insights**:
  - Histograms: Highlighted distributions of age and monthly revenue.
  - Heatmap: Showed relationships between features.
  - PCA and t-SNE: Revealed distinct user clusters and dimensionality reduction.

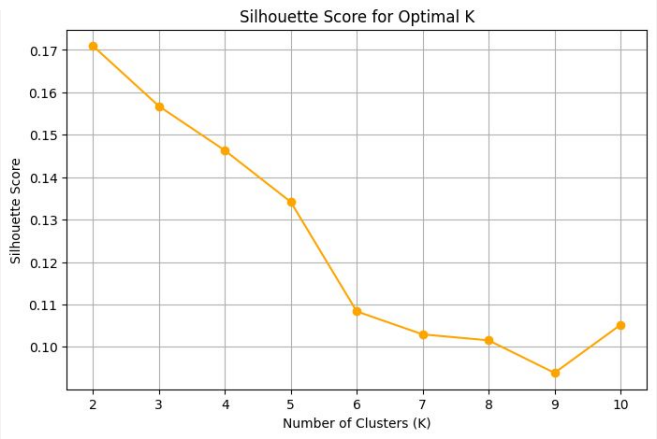
- **Implication:** Identified separable user groups, supporting clustering feasibility.



K-means

K-Means Clustering





Approach:

- Used Elbow Method and Silhouette Scores to determine optimal clusters.
- Chose 4 clusters for balance between interpretability and separability.

Results:

- Clear segmentations identified through PCA visualizations.
- Clusters characterized by attributes like revenue and engagement.

K-means

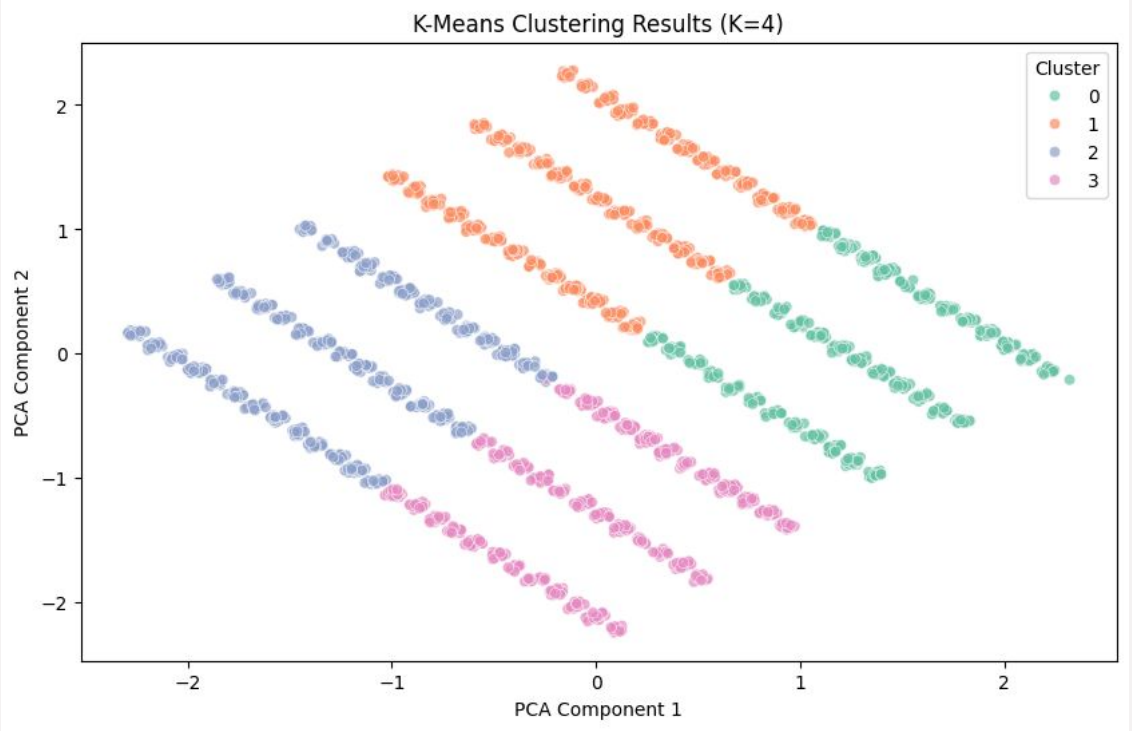
Approach:

- Used Elbow Method and Silhouette Scores to determine optimal clusters.
- Chose 4 clusters for balance between interpretability and separability.

Results:

- Clear segmentations identified through PCA visualizations.
- Clusters characterized by attributes like revenue and engagement.

K-Means Clustering



K-means

Churn Prediction

Methodology:

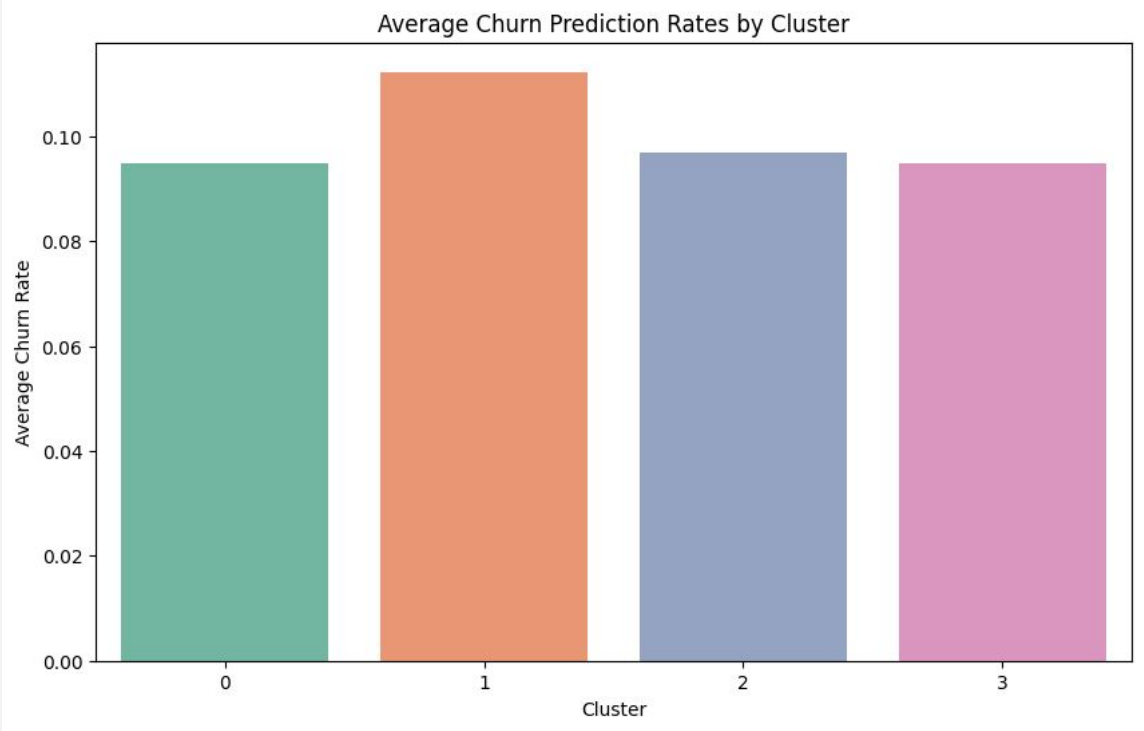
- Integrated Isolation Forest for churn detection.
- Analyzed cluster-level churn prediction rates.

Findings:

- Cluster 1 exhibited the highest churn rate (~10.9%).
- Other clusters showed lower churn rates (~9.5%).

Business Insight:

- Cluster 1 indicates at-risk customers needing targeted retention strategies.



K-means

Was K-means Successful?

Key Takeaways:

- K-Means effectively segmented users into meaningful groups.
- Combined clustering and isolation models identified churn risk areas.

Strengths:

- K-Means successfully segmented the user base into meaningful clusters with clear demographic and behavioral differences.
- These clusters enable targeted analysis of churn risk, which can inform retention strategies.

Limitations:

- K-Means alone does not inherently predict churn; it groups users based on feature similarity.
- The addition of an unsupervised model like Isolation Forest helped assign churn probabilities to these clusters, extending the functionality of K-Means.

Decision Trees

Decision Trees are a foundational supervised learning algorithm, used here for churn prediction.
Conducted hyperparameter tuning with GridSearchCV:

- Tested different values for max_depth, min_samples_split, min_samples_leaf, and criterion.
- Best parameters identified: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2}.

Achieved an accuracy of 66%, indicating room for improvement in predictive performance.
Decision Trees are interpretable, allowing us to pinpoint specific decision paths influencing churn.

```python
from sklearn.model_selection import GridSearchCV

params = {
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid=params, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best parameters for decision trees:", grid_search.best_params_)
best_model = grid_search.best_estimator_
```

```
Best parameters for decision trees: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2}
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

X = pd.get_dummies(data_training.drop(columns=['Churned']))
y = data_training['Churned']

y = y.map({'Yes':1, 'No':0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

dt_model = DecisionTreeClassifier(max_depth=5, random_state=42, class_weight='balanced')
dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Decision tree accuracy: {accuracy:.2f}")
```

```
Decision tree accuracy: 0.66
```

Random Forest

Random Forest builds on Decision Trees by combining multiple trees for better accuracy and robustness.
Hyperparameter tuning involved a larger search space for:

- Best parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}.

Improved accuracy to **82.2%**, showing the value of ensemble methods for predicting churn.
Random Forest reduces overfitting while providing feature importance scores for deeper insights.

```python
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 5, 10],
    'max_features': ['sqrt', 'log2', None]
}

grid_search_rf = GridSearchCV(
    RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy'
)
grid_search_rf.fit(X_train, y_train)

print("Best parameters for random forest:", grid_search_rf.best_params_)
best_rf_model = grid_search_rf.best_estimator_
```

```
Best parameters for random forest: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_es
```

```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42, min_samples_leaf=1, min_samples_spl
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("Random forest accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random forest accuracy: 0.822
```

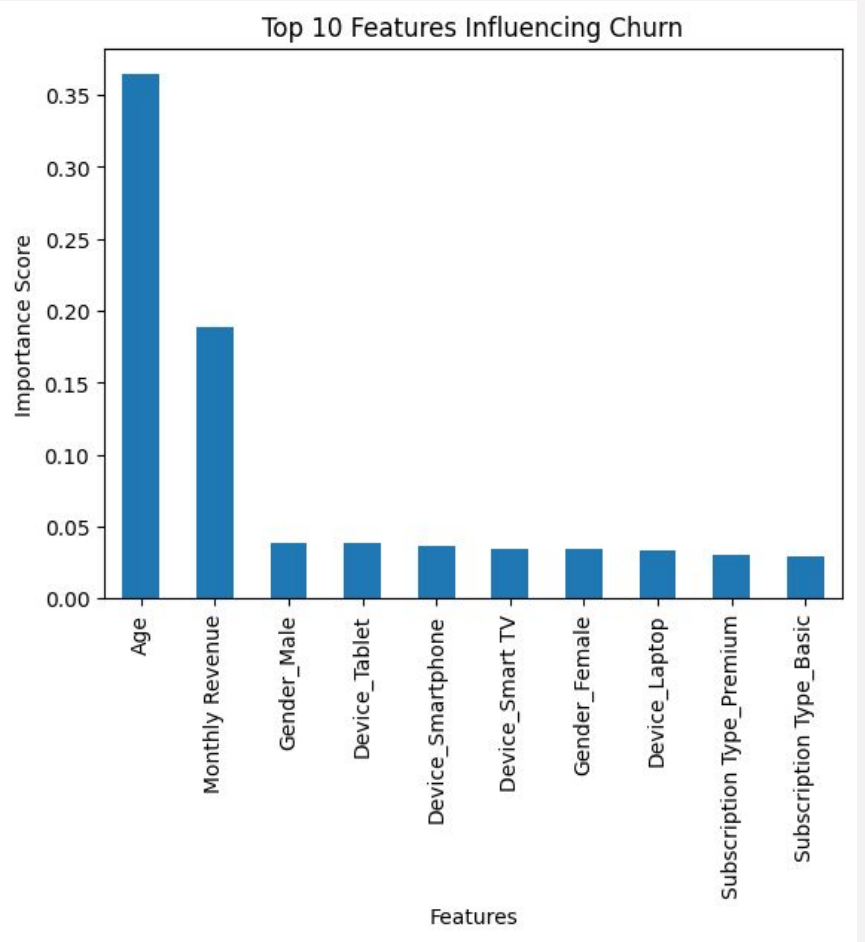
Top 10 Features Influencing Churn Rate

Feature importance from the Random Forest model highlights the key drivers of churn:

- **Age** is the most significant factor, followed by **Monthly Revenue**.
- Device type and subscription type also have notable contributions, indicating differences in user behavior.

Insights suggest that user demographics (e.g., Age) and revenue patterns are critical churn predictors.

Understanding feature importance informs targeted interventions and prioritizes areas for improvement.



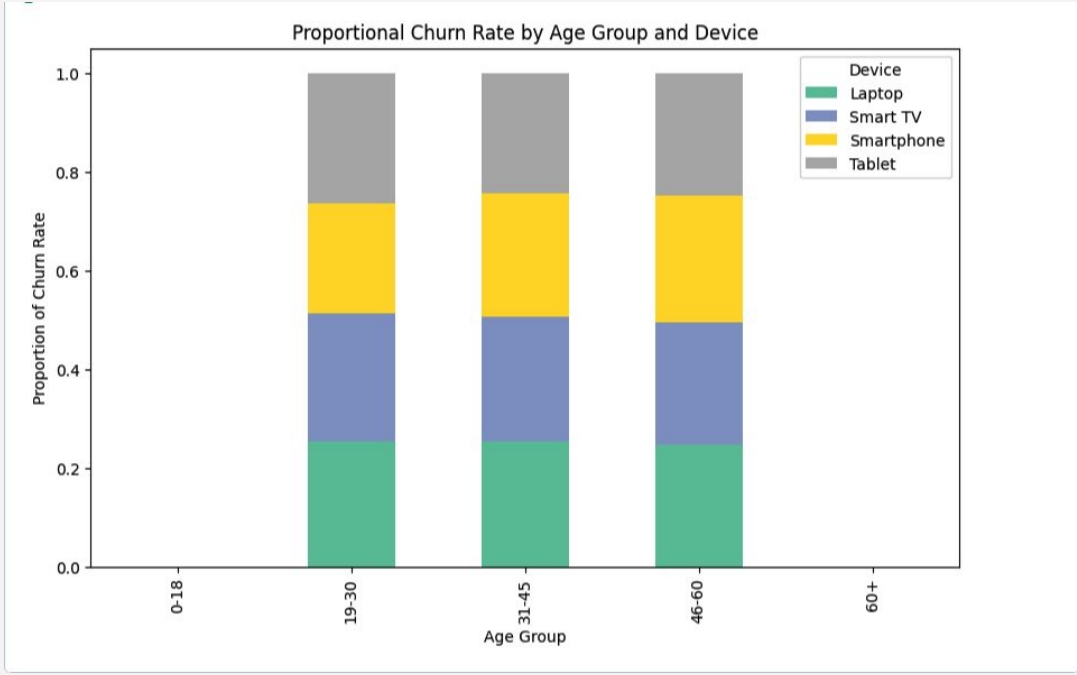
Churn Rate by Age Group and Device

This visualization explores churn distribution across **age groups** and **devices**:

- Mid-age users (31-45) show higher churn proportions on **Smartphones**.
- Older users (46-60) exhibit balanced churn across **Tablets** and **Laptops**.

Device-specific churn highlights potential usability or engagement issues.

Insight: Retention strategies should be customized based on age and device usage patterns.



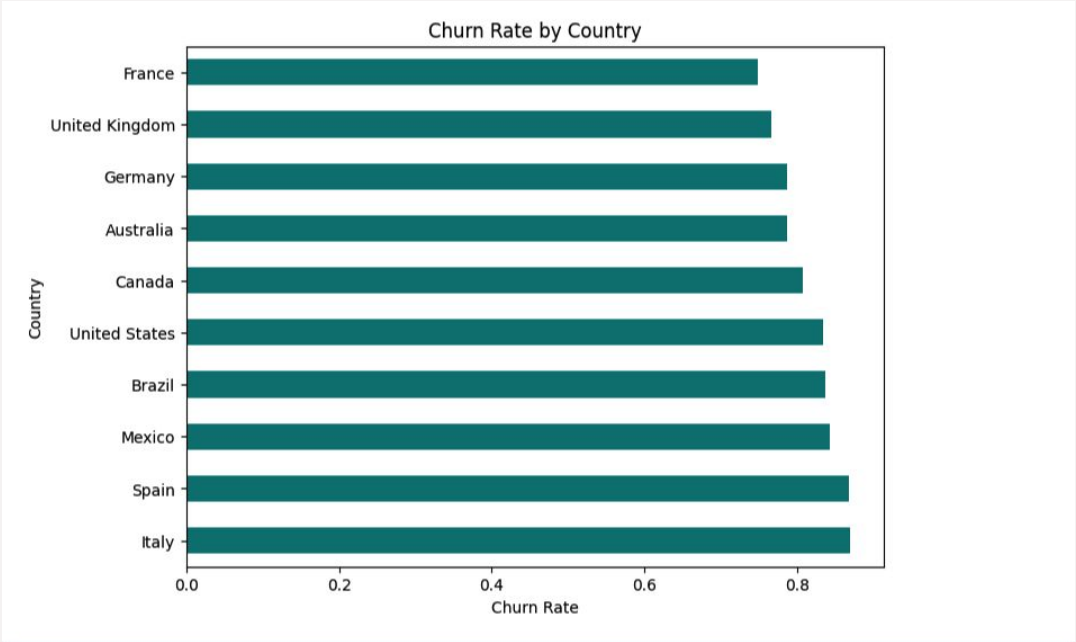
Churn Rate by Country

Horizontal bar chart compares churn rates across countries:

- Countries like **Italy**, **Spain**, and **Mexico** have the highest churn rates.
- Countries like **France** and the **United Kingdom** show relatively lower churn.

Regional differences suggest cultural, content, or pricing factors influencing churn.

Insights: support localized strategies, such as tailored content or competitive pricing, in high-churn regions.



Neural Networks

```
# NEURAL NETWORK 1
X = data_training[['Age']].values
y = data_training['Churned']

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)


le = LabelEncoder()

Y_train = le.fit_transform(Y_train)

Y_test = le.fit_transform(Y_test)



#neural network
clf = MLPClassifier(hidden_layer_sizes = 100, solver='lbfgs', activation = 'relu')

clf.fit(X_train, Y_train)
yp = clf.predict(X_test)

accuracy = accuracy_score(Y_test, yp)
accuracy
```

```
0.806
```

```
#NEURAL NETWORK USING CONTRY AS A FEATURE

#put the data in binary

X =  pd.get_dummies(data_training['Country'])
y = data_training['Churned']

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)


le = LabelEncoder()

Y_train = le.fit_transform(Y_train)

Y_test = le.fit_transform(Y_test)


#sample_weights = compute_sample_weight(class_weight= 'balanced', y=Y_train)
#neural network
clf = MLPClassifier(hidden_layer_sizes = (10,), solver='lbfgs', activation = 'relu')

clf.fit(X_train, Y_train)
yp = clf.predict(X_test)

accuracy = accuracy_score(Y_test, yp)
accuracy
```

```
0.804
```

Starting out with one-feature Neural Networks using only Age in NN1 and Country in NN2 (Country needed to be one-hot-encoded since it's a nominal feature)

Some annotations:

-Hidden layer size didn't seemed to increase/decrease accuracy of the model.

-After some trails it was determined optimal train/test split was 80/20.

Neural Networks

```python
#NEURAL NETWORK USING Subscription Type AND Monthly Revenue

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report



#put the data in binary

X1= pd.get_dummies(data_training['Subscription Type'])
X2 = pd.get_dummies(data_training['Monthly Revenue'])
#X3 = pd.get_dummies(data_training['Age'])
#X4 = pd.get_dummies(data_training['Subscription Type'])
X = np.concatenate([X1, X2], axis=1)

y = data_training['Churned']

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)


le = LabelEncoder()

Y_train = le.fit_transform(Y_train)

Y_test = le.fit_transform(Y_test)



#neural network
clf = MLPClassifier(hidden_layer_sizes = (10,), solver='lbfgs', activation = 'relu')

clf.fit(X_train, Y_train)
yp = clf.predict(X_test)

accuracy = accuracy_score(Y_test, yp)
accuracy
```

```
0.822
```

```python

#NEURAL NETWORK USING CONTRY AND GENDER

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report



#put the data in binary

X1= pd.get_dummies(data_training['Country'])
X2 = pd.get_dummies(data_training['Gender'])
#X3 = pd.get_dummies(data_training['Age'])
#X4 = pd.get_dummies(data_training['Subscription Type'])
X = np.concatenate([X1, X2], axis=1)

y = data_training['Churned']

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)


le = LabelEncoder()

Y_train = le.fit_transform(Y_train)

Y_test = le.fit_transform(Y_test)



#neural network
clf = MLPClassifier(hidden_layer_sizes = (10,), solver='lbfgs', activation = 'relu')

clf.fit(X_train, Y_train)
yp = clf.predict(X_test)

accuracy = accuracy_score(Y_test, yp)
accuracy
```

```
0.818
```

Started to concatenate features (after ohe them), after some trials found that 3 or more features overfitted the model leading to accuracy rates of around 70-75%, therefore the course of action was to leave it to two.

Accuracy was increased to 81% for NN3 (using Country and Gender as features) and 82% (using Subscription Type and Monthly Revenue as features)

Neural Networks

```
history=model.fit(X, y, epochs=100, validation_split=0.2, batch_size=32)

accuracy = model.evaluate(X_test, Y_test, verbose=0)[1]
accuracy
```

```
Epoch 86/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2591 - val_accuracy: 0.992
Epoch 87/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2582 - val_accuracy: 0.992
Epoch 88/100
63/63 [==============================] - 0s 4ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2559 - val_accuracy: 0.992
Epoch 89/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5219 - accuracy: 0.7790 - val_loss: 0.2545 - val_accuracy: 0.992
Epoch 90/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5218 - accuracy: 0.7790 - val_loss: 0.2563 - val_accuracy: 0.992
Epoch 91/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2618 - val_accuracy: 0.992
Epoch 92/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2600 - val_accuracy: 0.992
Epoch 93/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5219 - accuracy: 0.7790 - val_loss: 0.2588 - val_accuracy: 0.992
Epoch 94/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2610 - val_accuracy: 0.992
Epoch 95/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2534 - val_accuracy: 0.992
Epoch 96/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5218 - accuracy: 0.7790 - val_loss: 0.2568 - val_accuracy: 0.992
Epoch 97/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5216 - accuracy: 0.7790 - val_loss: 0.2593 - val_accuracy: 0.992
Epoch 98/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2582 - val_accuracy: 0.992
Epoch 99/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5217 - accuracy: 0.7790 - val_loss: 0.2569 - val_accuracy: 0.992
Epoch 100/100
63/63 [==============================] - 0s 3ms/step - loss: 0.5218 - accuracy: 0.7790 - val_loss: 0.2543 - val_accuracy: 0.992
```

```
0.8360000252723694
```

```
#NEURAL NETWORK USING CONTRY AND AGE AS A FEATURE, THIS TIME USING A SEQUENTIAL MODEL AND USING EPOCHS

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import tensorflow as tf

from tensorflow.keras.layers import Dense


X1= pd.get_dummies(data_training['Country'])
X2 = pd.get_dummies(data_training['Gender'])
X3 = pd.get_dummies(data_training['Age'])
#X4 = pd.get_dummies(data_training['Subscription Type'])
X = np.concatenate([X1, X3], axis=1)

y = pd.get_dummies(data_training['Churned'])

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)


model = Sequential()
#model.add(Dropout(0.2))

model.add(Dense(10, activation='relu', input_dim = X.shape[1]))
model.add(Dense(2, activation='softmax'))



model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history=model.fit(X, y, epochs=100, validation_split=0.25, batch_size=32)

accuracy = model.evaluate(X_test, Y_test, verbose=0)[1]
accuracy
```

This time I did a more advanced Neural Network using two features but importing Keras to initialize a Sequential model adding a dense layer with 2 neurons and another dense layer with 10 neurons, with relu and softmax activation features, and finally did a validation split of 25% and got the accuracy to almost 85%

Results and Conclusions

1. Although one our initial assumption was that all three methods used (NN, Random Forests and K-means) would lead us to successful results only Random Forests and NN performed well while K-means didn't.

2. The K-Means clustering effectively segmented users into distinct groups, with Cluster 1 identified as the highest churn risk (10.9%) due to unique demographic and behavioral attributes, enabling targeted retention strategies, while additional models like Isolation Forest enhanced churn prediction accuracy.

3. Actionable insights: address regional disparities by optimizing content offerings, pricing models, and support services in high-churn countries like Italy and Spain, while reinforcing successful practices in low-churn regions like France and the UK.