STATS/CSE 780 - Homework Assignment 2

Name: Amanpreet Singh (400672477)

2025-09-30

Assignment Questions

1. Dataset Sourcing

- Briefly describe how the dataset was sourced.

- Describe the chosen dataset (what are the variables and observations).

---

2. Application Description

- Clearly describe the application you are investigating based on the selected dataset.

---

3. Exploratory Data Analysis

- Produce numerical and graphical summaries of the data.

- Examine patterns and provide details such as:
  - Data types of the variables
  - Data summaries
  - Correlation and association analysis
  - Outliers and missing values analysis

---

4. Data Splitting

- Split the data into training, validation, and test sets.

- Describe the rationale behind your choice of data splitting.

---

5. Logistic Regression and k-Nearest Neighbor

Using the training and validation sets:

For Each Classifier:

1. Describe if you used any transformations or selected a subset of predictors or categories.

2. Describe the choice of tuning parameters (if any).

3. Identify the most important predictor variable(s) in classifying the response, or explain why this cannot be determined.

---

6. Model Evaluation

1. Use the test set to evaluate the performance of the classifiers using the misclassification error rate.

    - If the classifier needs a cutoff to classify the labels, use sensitivity and specificity analysis to determine the cutoff.

2. Compare and contrast the performance of the classifiers. Provide at least two statements.

---

7. Alternative kNN Strategy

- Perform kNN with an alternative nearest neighbor search strategy or distance metric learning method (different from what you used in (5)), using the training and validation sets.

- Example Python libraries:
  - pyDML
  - metric-learn

Tasks:

1. Describe the search strategy or distance metric learning method you used, including any tuning parameters (if applicable).

2. Compare and contrast the kNN performance with and without the new search strategy or distance metric learning method, using the test set.

---

8. Conclusions

- State at least two conclusions based on your analysis in (4)–(7).

- These conclusions should be clearly connected to your selected dataset and application.

---

9. Generative AI

1. Can generative AI be used to answer one of the questions in (1)–(8)?

2. Provide the prompt(s) used to answer the question.

3. Discuss whether generative AI tools could assist in completing any parts of this assignment.

References

(**pasanisi_ocean_2022-2?**)

Supplemental Material

- Note: GitHub Copilot was used to assist with UI development and error handling.
- The Shiny app involves extensive UI design, and during development, multiple errors occurred which required careful debugging and adjustments.

```
# Core
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing & splitting
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Logistic Regression & kNN
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder

# Metrics & evaluation
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    roc_curve,
    roc_auc_score
)
```

3. Exploratory Data Analysis

```
df = pd.read_csv("spotify_churn_dataset.csv")
df = df.drop(columns=['user_id'])
```

```
df.isna().sum()
df.shape
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8000 entries, 0 to 7999
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   gender                 8000 non-null   object
 1   age                    8000 non-null   int64
 2   country                8000 non-null   object
 3   subscription_type      8000 non-null   object
 4   listening_time         8000 non-null   int64
 5   songs_played_per_day   8000 non-null   int64
 6   skip_rate              8000 non-null   float64
 7   device_type            8000 non-null   object
 8   ads_listened_per_week  8000 non-null   int64
 9   offline_listening      8000 non-null   int64
 10  is_churned             8000 non-null   int64
dtypes: float64(1), int64(6), object(4)
memory usage: 687.6+ KB
```

	age	listening_time	songs_played_per_day	skip_rate	ads_listened_per_week	offline
count	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.0

	age	listening_time	songs_played_per_day	skip_rate	ads_listened_per_week	offline
mean	37.662125	154.068250	50.127250	0.300127	6.943875	0.7477
std	12.740359	84.015596	28.449762	0.173594	13.617953	0.4343
min	16.000000	10.000000	1.000000	0.000000	0.000000	0.0000
25%	26.000000	81.000000	25.000000	0.150000	0.000000	0.0000
50%	38.000000	154.000000	50.000000	0.300000	0.000000	1.0000
75%	49.000000	227.000000	75.000000	0.450000	5.000000	1.0000
max	59.000000	299.000000	99.000000	0.600000	49.000000	1.0000

```
for x in ['gender','subscription_type','device_type','is_churned']:
    print(df[x].value_counts(normalize=True))
    print('-----')
```

```
gender
Male      0.336375
Female    0.332375
Other     0.331250
Name: proportion, dtype: float64
-----
subscription_type
Premium    0.264375
Free       0.252250
Student    0.244875
Family     0.238500
Name: proportion, dtype: float64
-----
device_type
Desktop    0.347250
Web        0.327875
Mobile     0.324875
Name: proportion, dtype: float64
```

```
-----
is_churned
0    0.741125
1    0.258875
Name: proportion, dtype: float64
-----
```

```
col = ['gender', 'age', 'country', 'subscription_type',
    'skip_rate', 'device_type',
       'ads_listened_per_week', 'offline_listening', 'is_churned']
for x in col:
    print(x,df[x].unique())
    print('-----')
```

```
gender ['Female' 'Other' 'Male']
-----
age [54 33 38 22 29 17 39 41 55 44 24 37 53 25 36 58 34 35 49 45 32 19 28 50
 42 51 46 18 56 59 57 48 21 30 23 47 20 52 43 27 31 16 40 26]
-----
country ['CA' 'DE' 'AU' 'US' 'UK' 'IN' 'FR' 'PK']
-----
subscription_type ['Free' 'Family' 'Premium' 'Student']
-----
skip_rate [0.2  0.34 0.04 0.31 0.36 0.46 0.38 0.11 0.29 0.56 0.6  0.44 0.26 0.23
 0.41 0.35 0.45 0.1  0.42 0.59 0.53 0.33 0.3  0.22 0.39 0.17 0.43 0.18
 0.06 0.5  0.54 0.51 0.4  0.48 0.58 0.08 0.   0.32 0.49 0.14 0.21 0.05
 0.19 0.47 0.24 0.03 0.37 0.15 0.25 0.07 0.02 0.12 0.27 0.13 0.16 0.52
 0.09 0.28 0.01 0.55 0.57]
-----
device_type ['Desktop' 'Web' 'Mobile']
-----
ads_listened_per_week [31  0 13  5 44 37 39 23 35  7 28 22 43 15 12 19 38 14 18 10 32 29 20 16
```

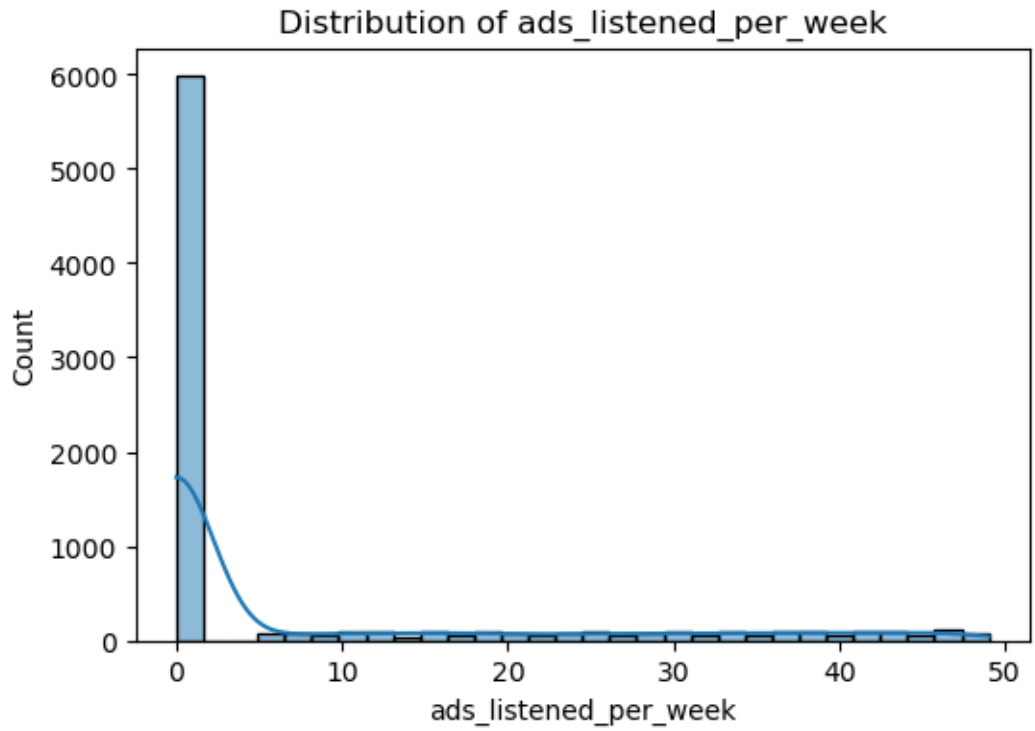
```
  6 36 27 49 40 47  9 45 42 48 26  8 24 30 25 41 21 34 17 46 33 11]
-----
offline_listening [0 1]
-----
is_churned [1 0]
-----
```

```
df['ads_zero_flag'] = df['ads_listened_per_week'].apply(lambda x: 'Zero' if x == 0 else 'Non-ze
```

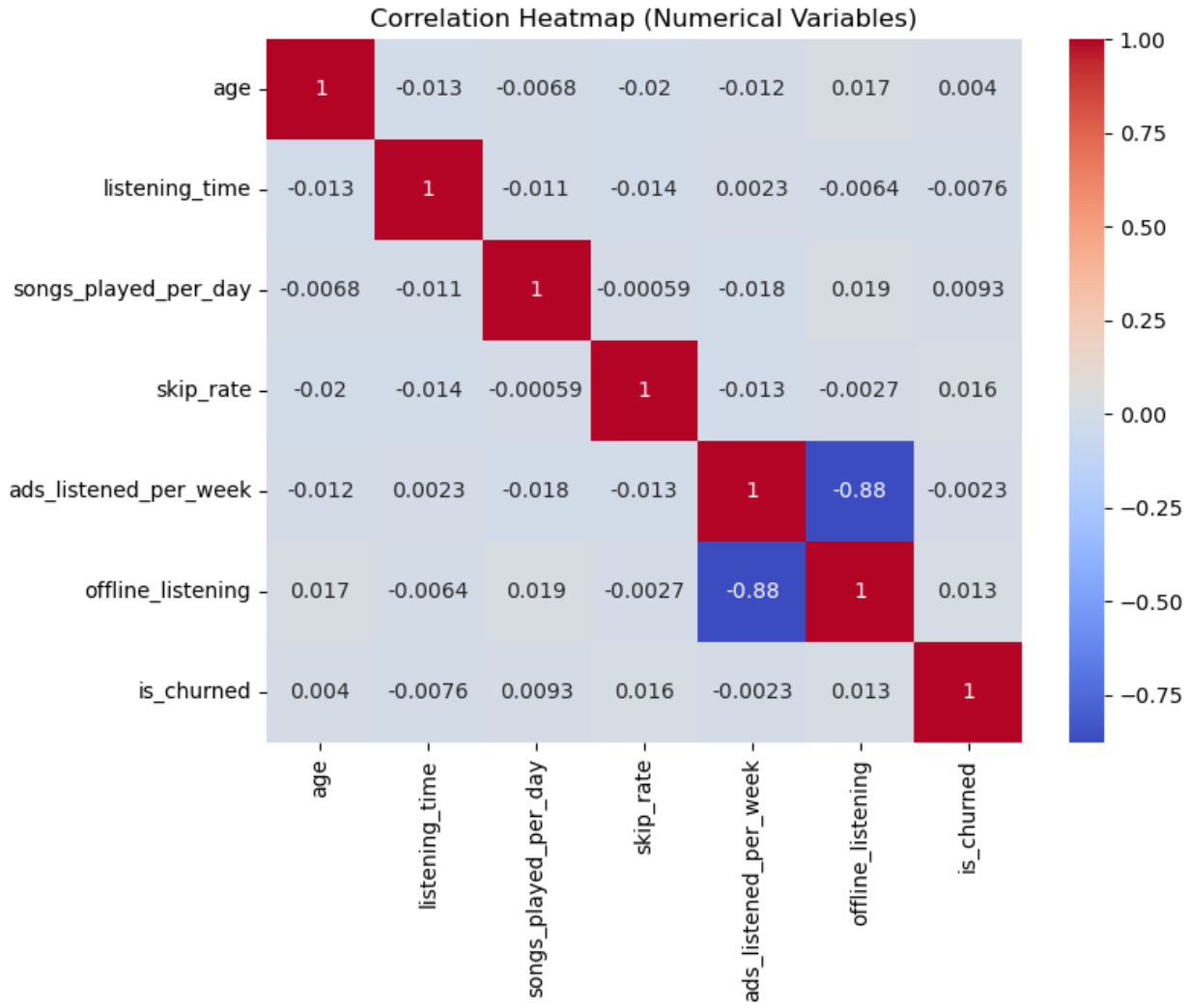
```
# Group by subscription type and ads_zero_flag
counts_by_sub = df.groupby('subscription_type')['ads_zero_flag'].value_counts().unstack(fill_va
df = df.drop(columns=['ads_zero_flag'])
print(counts_by_sub)
```

ads_zero_flag	Non-zero	Zero
subscription_type		
Family	0	1908
Free	2018	0
Premium	0	2115
Student	0	1959

```
plt.figure(figsize=(6,4))
sns.histplot(df['ads_listened_per_week'], kde=True, bins=30)
plt.title(f"Distribution of {col}")
plt.show()
```




```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap (Numerical Variables)")
plt.show()
```




```
offline_counts_by_sub = df.groupby('subscription_type')['offline_listening'].value_counts().uns
print(offline_counts_by_sub)
```

```
offline_listening      0     1
subscription_type
Family                 0  1908
Free                2018     0
Premium                0  2115
Student                0  1959
```

4. Data Splitting

```
y = df['is_churned']

# Features: drop the target
X = df.drop(columns=['is_churned'])

# Optional: check the shapes
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (8000, 10)
y shape: (8000,)
```

```
X
```

	gender	age	country	subscription_type	listening_time	songs_played_per_day	skip_rate	devi
0	Female	54	CA	Free	26	23	0.20	Desk
1	Other	33	DE	Family	141	62	0.34	Web
2	Male	38	AU	Premium	199	38	0.04	Mob
3	Female	22	CA	Student	36	2	0.31	Mob
4	Other	29	US	Family	250	57	0.36	Mob
...	...	...	...	...	...	...	...	...
7995	Other	44	DE	Student	237	36	0.30	Mob
7996	Male	34	AU	Premium	61	64	0.59	Mob
7997	Female	17	US	Free	81	62	0.33	Desk
7998	Female	34	IN	Student	245	94	0.27	Desk
7999	Other	45	AU	Free	210	68	0.46	Desk

```
from sklearn.model_selection import train_test_split

# First, spliting off the test set (20%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Then, spliting the remaining 80% into training and validation (75% train, 25% val → 60/20 ove
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=42, stratify=y_temp
)

print("Training size:", X_train.shape[0])
print("Validation size:", X_val.shape[0])
print("Test size:", X_test.shape[0])
```

```
Training size: 4800
Validation size: 1600
Test size: 1600
```

```
all_cols
```

```
['age',
 'listening_time',
 'songs_played_per_day',
 'skip_rate',
 'ads_listened_per_week',
 'offline_listening',
 'gender',
 'country',
 'subscription_type',
 'device_type']
```

```
# 3. Identify categorical and numeric columns
cat_cols = X.select_dtypes(include=['object']).columns.tolist()
num_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# 4. One-hot encode categorical columns
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
X_train_cat = ohe.fit_transform(X_train[cat_cols])
X_val_cat = ohe.transform(X_val[cat_cols])
X_test_cat = ohe.transform(X_test[cat_cols])

# 5. Extract numeric features
X_train_num = X_train[num_cols].values
X_val_num = X_val[num_cols].values
X_test_num = X_test[num_cols].values

# 6. Combine numeric + encoded categorical features
X_train_final = np.hstack([X_train_num, X_train_cat])
X_val_final = np.hstack([X_val_num, X_val_cat])
X_test_final = np.hstack([X_test_num, X_test_cat])

# 7. Convert to DataFrame
cat_cols_encoded = ohe.get_feature_names_out(cat_cols)
all_cols = num_cols + cat_cols_encoded.tolist()
X_train_final_df = pd.DataFrame(X_train_final, columns=all_cols, index=X_train.index)
X_val_final_df = pd.DataFrame(X_val_final, columns=all_cols, index=X_val.index)
X_test_final_df = pd.DataFrame(X_test_final, columns=all_cols, index=X_test.index)

# 8. Shapes
print("X_train:", X_train_final_df.shape)
print("X_val:", X_val_final_df.shape)
print("X_test:", X_test_final_df.shape)
```

```
X_train: (4800, 24)
X_val: (1600, 24)
X_test: (1600, 24)
```

```
# 3. One-hot encode categorical variables directly (returns DataFrame)
X_train_final = pd.get_dummies(X_train, dtype=int, columns=X_train.select_dtypes(include=['obj
X_val_final = pd.get_dummies(X_val, dtype=int,columns=X_val.select_dtypes(include=['object']).
X_test_final = pd.get_dummies(X_test, dtype=int,columns=X_test.select_dtypes(include=['object'

# 4. Align columns (in case some categories are missing in val/test)
X_val_final = X_val_final.reindex(columns=X_train_final.columns, fill_value=0)
X_test_final = X_test_final.reindex(columns=X_train_final.columns, fill_value=0)

# 5. Shapes
print("X_train:", X_train_final.shape)
print("X_val:", X_val_final.shape)
print("X_test:", X_test_final.shape)
```

```
X_train: (4800, 24)
X_val: (1600, 24)
X_test: (1600, 24)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# -------------------------
# 1. Logistic Regression
# -------------------------
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train_final, y_train)
```

```
# Predictions
y_val_pred_lr = logreg.predict(X_val_final)
y_test_pred_lr = logreg.predict(X_test_final)

# Evaluation
print("Logistic Regression - Validation Accuracy:", accuracy_score(y_val, y_val_pred_lr))
print("Logistic Regression - Test Accuracy:", accuracy_score(y_test, y_test_pred_lr))
print(classification_report(y_test, y_test_pred_lr))

# -------------------------
# 2. K-Nearest Neighbors
# -------------------------
knn = KNeighborsClassifier(n_neighbors=5)  # you can tune k
knn.fit(X_train_final, y_train)

# Predictions
y_val_pred_knn = knn.predict(X_val_final)
y_test_pred_knn = knn.predict(X_test_final)

# Evaluation
print("KNN - Validation Accuracy:", accuracy_score(y_val, y_val_pred_knn))
print("KNN - Test Accuracy:", accuracy_score(y_test, y_test_pred_knn))
print(classification_report(y_test, y_test_pred_knn))
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_final)
X_val_scaled = scaler.transform(X_val_final)
X_test_scaled = scaler.transform(X_test_final)
```