Preprocessing data

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

scikit-learn requirements

- Numeric data

- No missing values

- With real-world data:
  - This is rarely the case

  - We will often need to preprocess our data first

Dealing with categorical features

- scikit-learn will not accept categorical features by default

- Need to convert categorical features into numeric values

- Convert to binary features called dummy variables
  - 0: Observation was NOT that category

  - 1: Observation was that category

Dummy variables

genre
Alternative
Anime
Blues
Classical
Country
Electronic
Hip-Hop
Jazz
Rap
Rock

Dummy variables

genre
Alternative
Anime
Blues
Classical
Country
Electronic
Hip-Hop
Jazz
Rap
Rock

Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	Blue	0	0	0
0	0	0	1	0	0	Classical	0	0	0
0	0	0	0	1	0	Country	0	0	0
0	0	0	0	0	1	Electronic	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	Jazz	1	0	0
0	0	0	0	0	0	Rap	0	1	0
0	0	0	0	0	0	Rock	0	0	1

Dummy variables

genre
Alternative
Anime
Blues
Classical
Country
Electronic
Hip-Hop
Jazz
Rap
Rock

➡️

Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	B0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	C0	0	0
0	0	0	0	0	1	E0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	Jazz	1	0
0	0	0	0	0	0	R0	0	1
0	0	0	0	0	0	Ro	0	0

Dealing with categorical features in Python

- scikit-learn: `OneHotEncoder()`
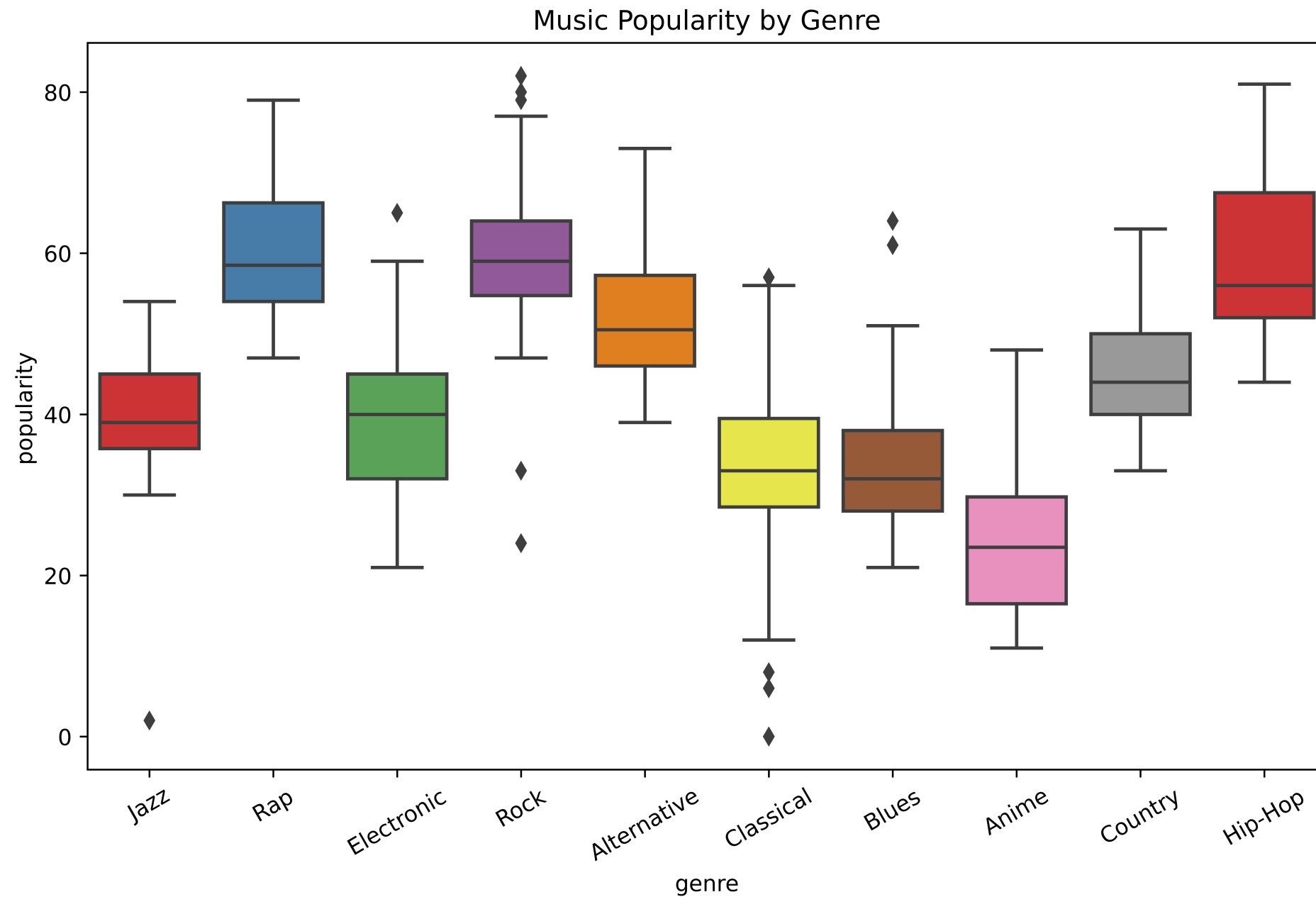
- pandas: `get_dummies()`

Music dataset

- `popularity` : Target variable

- `genre` : Categorical feature

```
print(music.info())
```

```
   popularity   acousticness   danceability   ...   tempo        valence   genre
0  41.0         0.6440         0.823          ...   102.619000   0.649     Jazz
1  62.0         0.0855         0.686          ...   173.915000   0.636     Rap
2  42.0         0.2390         0.669          ...   145.061000   0.494     Electronic
3  64.0         0.0125         0.522          ...   120.406497   0.595     Rock
4  60.0         0.1210         0.780          ...   96.056000    0.312     Rap
```

EDA w/ categorical feature

Music Popularity by Genre

Encoding dummy variables

```python
import pandas as pd
music_df = pd.read_csv('music.csv')
music_dummies = pd.get_dummies(music_df["genre"], drop_first=True)
print(music_dummies.head())
```

	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock
0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	1	0

```python
music_dummies = pd.concat([music_df, music_dummies], axis=1)
music_dummies = music_dummies.drop("genre", axis=1)
```

Encoding dummy variables

```python
music_dummies = pd.get_dummies(music_df, drop_first=True)
print(music_dummies.columns)
```

```
Index(['popularity', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'genre_Anime', 'genre_Blues', 'genre_Classical',
       'genre_Country', 'genre_Electronic', 'genre_Hip-Hop', 'genre_Jazz',
       'genre_Rap', 'genre_Rock'],
      dtype='object')
```

Linear regression with dummy variables

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
X = music_dummies.drop("popularity", axis=1).values
y = music_dummies["popularity"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
linreg = LinearRegression()
linreg_cv = cross_val_score(linreg, X_train, y_train, cv=kf,
                            scoring="neg_mean_squared_error")
print(np.sqrt(-linreg_cv))
```

```
[8.15792932, 8.63117538, 7.52275279, 8.6205778, 7.91329988]
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Handling missing data

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

Missing data

- No value for a feature in a particular row

- This can occur because:
  - There may have been no observation

  - The data might be corrupt

- We need to deal with missing data

Music dataset

```python
print(music_df.isna().sum().sort_values())
```

```
genre                 8
popularity           31
loudness             44
liveness             46
tempo                46
speechiness          59
duration_ms          91
instrumentalness     91
danceability        143
valence             143
acousticness        200
energy              200
dtype: int64
```

Dropping missing data

```
music_df = music_df.dropna(subset=["genre", "popularity", "loudness", "liveness", "tempo"])
print(music_df.isna().sum().sort_values())
```

```
popularity              0
liveness                0
loudness                0
tempo                   0
genre                   0
duration_ms            29
instrumentalness       29
speechiness            53
danceability          127
valence               127
acousticness          178
energy                178
dtype: int64
```

Imputing values

- Imputation - use subject-matter expertise to replace missing data with educated guesses

- Common to use the mean

- Can also use the median, or another value

- For categorical values, we typically use the most frequent value - the mode

- Must split our data first, to avoid *data leakage*

Imputation with scikit-learn

```python
from sklearn.impute import SimpleImputer
X_cat = music_df["genre"].values.reshape(-1, 1)
X_num = music_df.drop(["genre", "popularity"], axis=1).values
y = music_df["popularity"].values
X_train_cat, X_test_cat, y_train, y_test = train_test_split(X_cat, y, test_size=0.2,
                                                            random_state=12)
X_train_num, X_test_num, y_train, y_test = train_test_split(X_num, y, test_size=0.2,
                                                            random_state=12)
imp_cat = SimpleImputer(strategy="most_frequent")
X_train_cat = imp_cat.fit_transform(X_train_cat)
X_test_cat = imp_cat.transform(X_test_cat)
```

Imputation with scikit-learn

```python
imp_num = SimpleImputer()
X_train_num = imp_num.fit_transform(X_train_num)
X_test_num = imp_num.transform(X_test_num)
X_train = np.append(X_train_num, X_train_cat, axis=1)
X_test = np.append(X_test_num, X_test_cat, axis=1)
```

- Imputers are known as transformers

Imputing within a pipeline

```python
from sklearn.pipeline import Pipeline
music_df = music_df.dropna(subset=["genre", "popularity", "loudness", "liveness", "tempo"])
music_df["genre"] = np.where(music_df["genre"] == "Rock", 1, 0)
X = music_df.drop("genre", axis=1).values
y = music_df["genre"].values
```

Imputing within a pipeline

```python
steps = [("imputation", SimpleImputer()),
         ("logistic_regression", LogisticRegression())]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
```

```
0.7593582887700535
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Centering and scaling

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager

Why scale our data?

```
print(music_df[["duration_ms", "loudness", "speechiness"]].describe())
```

	duration_ms	loudness	speechiness
count	1.000000e+03	1000.000000	1000.000000
mean	2.176493e+05	-8.284354	0.078642
std	1.137703e+05	5.065447	0.088291
min	-1.000000e+00	-38.718000	0.023400
25%	1.831070e+05	-9.658500	0.033700
50%	2.176493e+05	-7.033500	0.045000
75%	2.564468e+05	-5.034000	0.078642
max	1.617333e+06	-0.883000	0.710000

Why scale our data?

- Many models use some form of distance to inform them

- Features on larger scales can disproportionately influence the model

- Example: KNN uses distance explicitly when making predictions

- We want features to be on a similar scale

- Normalizing or standardizing (scaling and centering)

How to scale our data

- Subtract the mean and divide by variance
  - All features are centered around zero and have a variance of one

  - This is called **standardization**

- Can also subtract the minimum and divide by the range
  - Minimum zero and maximum one

- Can also *normalize* so the data ranges from -1 to +1

- See scikit-learn docs for further details

Scaling in scikit-learn

```python
from sklearn.preprocessing import StandardScaler
X = music_df.drop("genre", axis=1).values
y = music_df["genre"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(np.mean(X), np.std(X))
print(np.mean(X_train_scaled), np.std(X_train_scaled))
```

```
19801.42536120538, 71343.52910125865
2.260817795600319e-17, 1.0
```

Scaling in a pipeline

```python
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier(n_neighbors=6))]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)
knn_scaled = pipeline.fit(X_train, y_train)
y_pred = knn_scaled.predict(X_test)
print(knn_scaled.score(X_test, y_test))
```

```
0.81
```

Comparing performance using unscaled data

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)
knn_unscaled = KNeighborsClassifier(n_neighbors=6).fit(X_train, y_train)
print(knn_unscaled.score(X_test, y_test))
```

```
0.53
```

CV and scaling in a pipeline

```python
from sklearn.model_selection import GridSearchCV
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
parameters = {"knn__n_neighbors": np.arange(1, 50)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)
cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)
y_pred = cv.predict(X_test)
```

Checking model parameters

```python
print(cv.best_score_)
```

```
0.819999999999999
```

```python
print(cv.best_params_)
```

```
{'knn__n_neighbors': 12}
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Evaluating multiple models

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

Different models for different problems

Some guiding principles

- Size of the dataset
  - Fewer features = simpler model, faster training time
  - Some models require large amounts of data to perform well

- Interpretability
  - Some models are easier to explain, which can be important for stakeholders
  - Linear regression has high interpretability, as we can understand the coefficients

- Flexibility
  - May improve accuracy, by making fewer assumptions about data
  - KNN is a more flexible model, doesn't assume any linear relationships

It's all in the metrics

- Regression model performance:
  - RMSE
  - R-squared
- Classification model performance:
  - Accuracy
  - Confusion matrix
  - Precision, recall, F1-score
  - ROC AUC
- Train several models and evaluate performance out of the box

A note on scaling

- Models affected by scaling:
  - KNN

  - Linear Regression (plus Ridge, Lasso)

  - Logistic Regression

  - Artificial Neural Network


- Best to scale our data before evaluating models

Evaluating classification models

```python
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
X = music.drop("genre", axis=1).values
y = music["genre"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Evaluating classification models

```python
models = {"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(),
          "Decision Tree": DecisionTreeClassifier()}
results = []
for model in models.values():
    kf = KFold(n_splits=6, random_state=42, shuffle=True)
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
    results.append(cv_results)
plt.boxplot(results, labels=models.keys())
plt.show()
```

Visualizing results
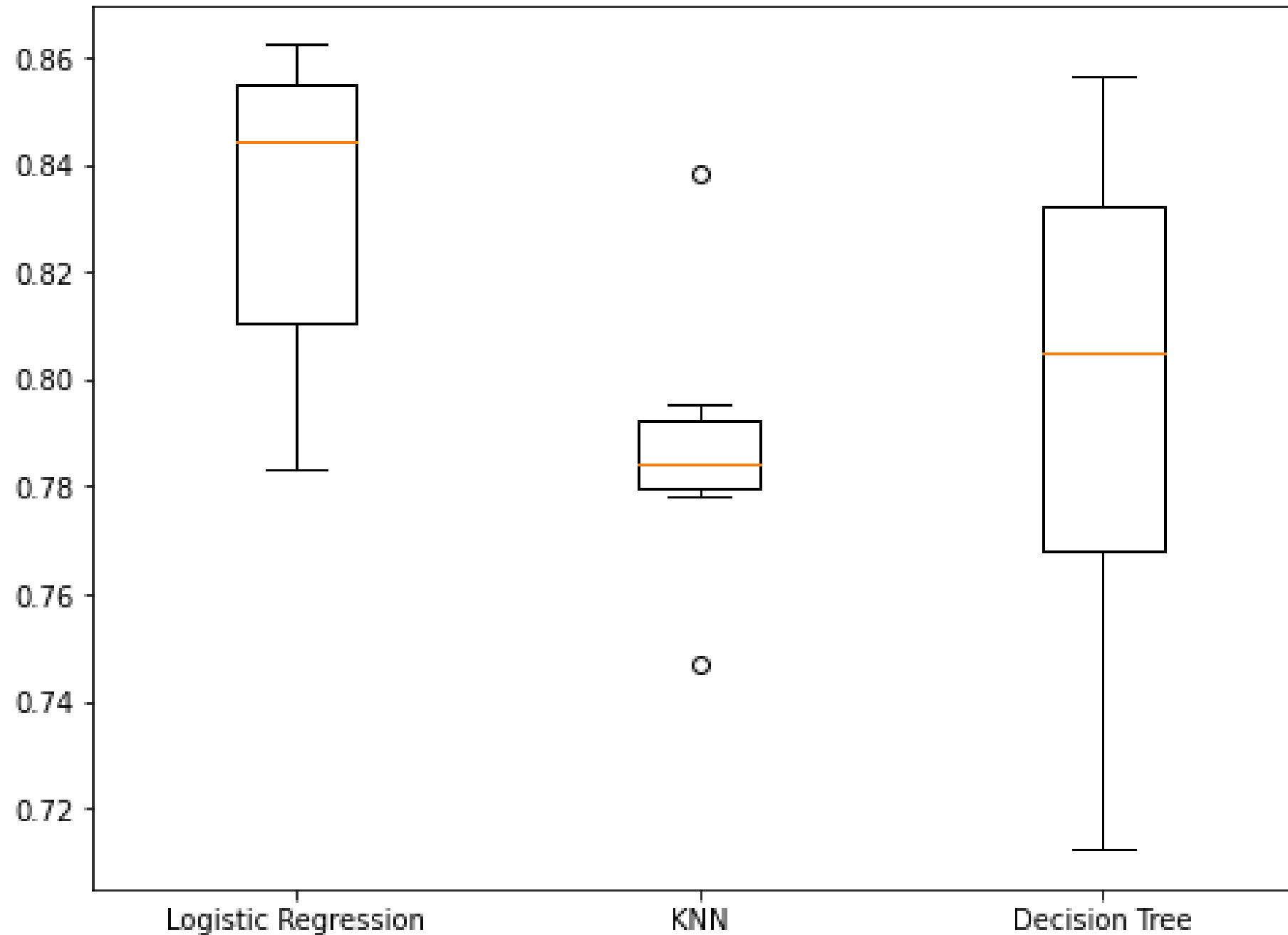
Test set performance

```python
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    test_score = model.score(X_test_scaled, y_test)
    print("{} Test Set Accuracy: {}".format(name, test_score))
```

```
Logistic Regression Test Set Accuracy: 0.844
KNN Test Set Accuracy: 0.82
Decision Tree Test Set Accuracy: 0.832
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Congratulations

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

What you've covered

- Using supervised learning techniques to build predictive models

- For both regression and classification problems

- Underfitting and overfitting

- How to split data

- Cross-validation

What you've covered

- Data preprocessing techniques

- Model selection

- Hyperparameter tuning

- Model performance evaluation

- Using pipelines

Where to go from here?

- **Machine Learning with Tree-Based Models in Python**

- **Preprocessing for Machine Learning in Python**

- **Model Validation in Python**

- **Feature Engineering for Machine Learning in Python**

- **Unsupervised Learning in Python**

- **Machine Learning Projects**

Thank you!

SUPERVISED LEARNING WITH SCIKIT-LEARN