In [1]:
```
# ============================================================================
# Title: Assignment Week 9 – Best Model Selection and Hyperparameter Tuning
# Author: Pankaj Yadav
# Date: 22 Feb 2026
# Description: Find the best model and tune hyperparameters
# ============================================================================
```

In [2]:
```
# import important libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In this exercise, you will work with the Loan_Train.csv dataset which can be downloaded from this link: Loan Approval Data Set.

Import the dataset and ensure that it loaded properly.
Prepare the data for modeling by performing the following steps:

1. Drop the column "Load_ID."
2. Drop any rows with missing data.
3. Convert the categorical features into dummy variables.
4. Split the data into a training and test set, where the "Loan_Status" column is the target.
5. Create a pipeline with a min-max scaler and a KNN classifier (see section 15.3 in the Machine Learning with Python Cookbook).
6. Fit a default KNN classifier to the data with this pipeline. Report the model accuracy on the test set. Note: Fitting a pipeline model works just like fitting a regular model.
7. Create a search space for your KNN classifier where your "n_neighbors" parameter varies from 1 to 10. (see section 15.3 in the Machine Learning with Python Cookbook).

8. Fit a grid search with your pipeline, search space, and 5-fold cross-validation to find the best value for the “n_neighbors” parameter.
9. Find the accuracy of the grid search best model on the test set. Note: It is possible that this will not be an improvement over the default model, but likely it will be.
10. Now, repeat steps 6 and 7 with the same pipeline, but expand your search space to include logistic regression and random forest models with the hyperparameter values in section 12.3 of the Machine Learning with Python Cookbook.
11. What are the best model and hyperparameters found in the grid search? Find the accuracy of this model on the test set.
12. Summarize your results.

In [3]:
```
# Import the dataset and ensure that it loaded properly.

# Load the dataset
loan_data = pd.read_csv('Loan_Train.csv')

# Display the first few rows of the dataset
print('Shape:',loan_data.shape)

# Drop the column “Load_ID.”
loan_data = loan_data.drop(columns=['Loan_ID'])
print(loan_data.head(2))
```

```
Shape: (614, 13)
  Gender Married Dependents Education Self_Employed  ApplicantIncome  \
0   Male      No          0  Graduate            No             5849
1   Male     Yes          1  Graduate            No             4583

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                0.0         NaN             360.0             1.0
1             1508.0       128.0             360.0             1.0

  Property_Area Loan_Status
0         Urban           Y
1         Rural           N
```

In [4]:
```
# Key initial checks on dataset

# Drop any rows with missing data.
loan_data = loan_data.dropna()

# Check for missings
print(loan_data.isnull().sum())

# Check duplicate records
print('\nAny duplicate rows:',loan_data.duplicated().sum())
```

```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64

Any duplicate rows: 0
```

In [5]:
```
# Describe basic dataset stats

# numerical columns first
print('\n numerical columns:\n')
print(loan_data.describe(include='int'))

# Now categorical columns
print('\n categorical columns:\n')
print(loan_data.describe(include='object'))
```

```
 numerical columns:

       ApplicantIncome
count       480.000000
mean       5364.231250
std        5668.251251
min         150.000000
25%        2898.750000
50%        3859.000000
75%        5852.500000
max       81000.000000

 categorical columns:

       Gender Married Dependents Education Self_Employed Property_Area  \
count     480     480        480       480           480           480
unique      2       2          4         2             2             3
top      Male     Yes          0  Graduate            No     Semiurban
freq      394     311        274       383           414           191

       Loan_Status
count          480
unique           2
top              Y
freq           332
```

In [6]:

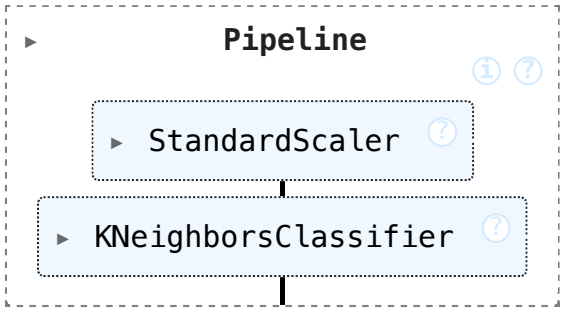
```python
# Convert the categorical features into dummy variables and
# Split the data into a training and test set, where the “Loan_Status” column is the target.

# Convert categorical features to dummy variables
loan_data = pd.get_dummies(loan_data, drop_first=True)

# Rename the target variable columns
loan_data.rename(columns={'Loan_Status_Y': 'Loan_Status'}, inplace=True)

# Define features and target
X = loan_data.drop('Loan_Status', axis=1)
y = loan_data['Loan_Status']
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print('Training set shape:', X_train.shape)
print('Test set shape:', X_test.shape)
```

```
Training set shape: (384, 14)
Test set shape: (96, 14)
```

In [7]:
```
# Create a pipeline with a min-max scaler and a KNN classifier (see
# section 15.3 in the Machine Learning with Python Cookbook).

# Create standardizer
standardizer = StandardScaler()

# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)

# Create a pipeline
pipe = Pipeline([("standardizer", standardizer), ("classifier", knn)])

#Fit a default KNN classifier to the data with this pipeline.
# Report the model accuracy on the test set.
# Note: Fitting a pipeline model works just like fitting a regular model.
pipe.fit(X_train, y_train)
```

Out[7]:



In [8]:
```
# Create a search space for your KNN classifier where
# your “n_neighbors” parameter varies from 1 to 10.
# (see section 15.3 in the Machine Learning with Python Cookbook).
```

```
search_space = [{"classifier__n_neighbors": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}]

# Fit a grid search with your pipeline, search space, and 5-fold
# cross-validation to find the best value for the “n_neighbors” parameter.

classifier = GridSearchCV(pipe, search_space, cv=5, verbose=0)
classifier.fit(X_train, y_train)

# Best neighborhood size (k)
print(f"Best KNN size: {classifier.best_estimator_.get_params()['classifier__n_neighbors']}")
```

```
Best KNN size: 10
```

In [9]:

```
# Find the accuracy of the grid search best model on the test set.
# Note: It is possible that this will not be an improvement over
# the default model, but likely it will be.

# Find the accuracy of the grid search best model on the test set
test_accuracy = classifier.score(X_test, y_test)
print(f"Test set accuracy of the best model: {test_accuracy}")
```

```
Test set accuracy of the best model: 0.7916666666666666
```

In [10]:

```
# Now, repeat steps 6 and 7 with the same pipeline,
# but expand your search space to include logistic
# regression and random forest models with the hyperparameter
# values in section 12.3 of the Machine Learning with Python Cookbook.

# create a search space for logistic regression and random forest
search_space = [{"classifier": [LogisticRegression(max_iter=500,
       solver='liblinear')],
                 "classifier__penalty": ['l1', 'l2'],
                 "classifier__C": np.logspace(0, 4, 10)},
                {"classifier": [RandomForestClassifier()],
                 "classifier__n_estimators": [10, 100, 1000],
                 "classifier__max_features": [1, 2, 3]}]


# fit the pipeline
classifier = GridSearchCV(pipe, search_space, cv=5, verbose=0)
```

```python
classifier.fit(X_train, y_train)

# What are the best model and hyperparameters found in the grid search?
# Find the accuracy of this model on the test set.

# Best model
print(f"Best model: {classifier.best_estimator_}")

# Best hyperparameters
print(f"Best hyperparameters: {classifier.best_params_}")


# Accuracy on the test set
test_accuracy = classifier.score(X_test, y_test)
print(f"Test set accuracy: {test_accuracy}")
```

```
Best model: Pipeline(steps=[('standardizer', StandardScaler()),
                ('classifier',
                 LogisticRegression(C=np.float64(2.7825594022071245),
                                    max_iter=500, penalty='l1',
                                    solver='liblinear'))])
Best hyperparameters: {'classifier': LogisticRegression(max_iter=500, solver='liblinear'), 'classif
ier__C': np.float64(2.7825594022071245), 'classifier__penalty': 'l1'}
Test set accuracy: 0.8229166666666666
```

Summarize your results.

The best KNN model (after grid search) used n_neighbors=10 and achieved a test set accuracy of 0.79.
Expanding the search to include Logistic Regression and Random Forest, the best model was Logistic Regression with C=2.78, penalty=l1, and solver=liblinear.
This Logistic Regression model achieved a higher test set accuracy of 0.82.
Hyperparameter tuning and model selection improved performance, with Logistic Regression outperforming KNN and Random Forest for this dataset.