Lab M1.03 - sklearn Model Training + Evaluation
Anson Knausenberger

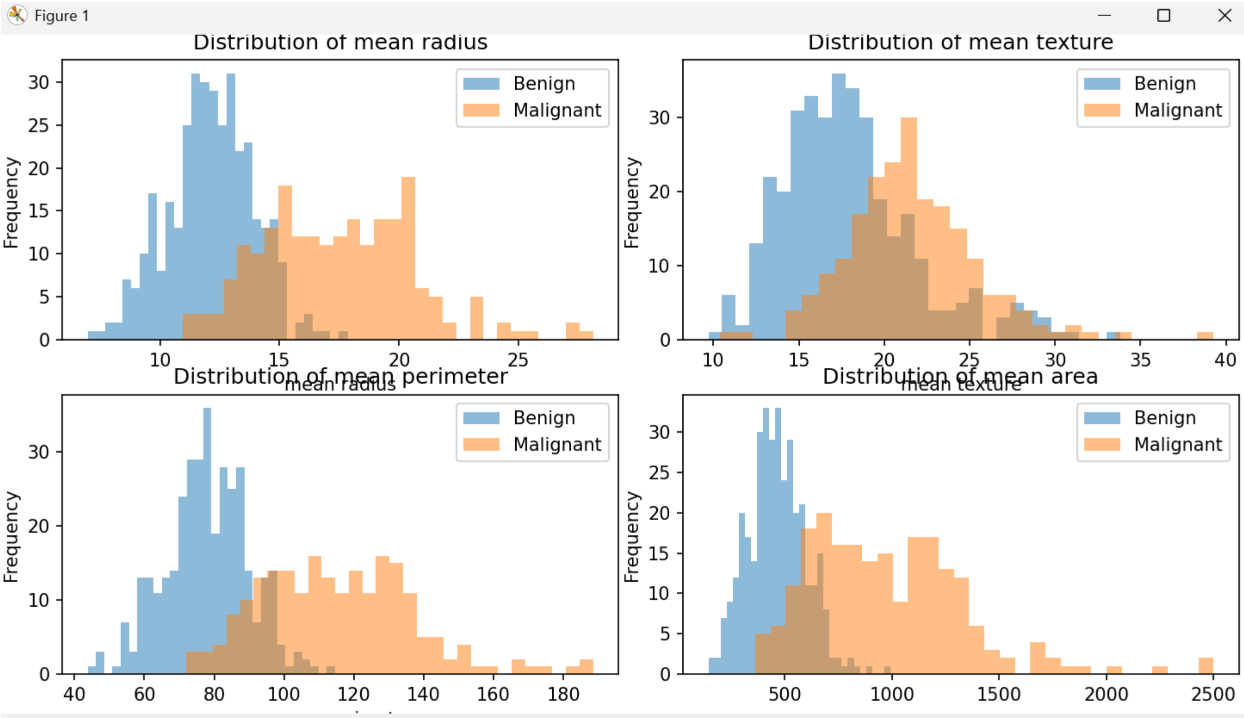
Part 1 (Guided Exercise):

Screenshots or output showing:

Dataset exploration

```
==================================================
BASIC STATISTICS
==================================================
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       mean symmetry  mean fractal dimension  ...  worst texture  \
...
==================================================
FEATURE DISTRIBUTIONS
==================================================
Saved visualization to 'feature_distributions.png'
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



Step 2 Observations

- `df.describe()` shows all 30 features are numeric and on different scales (some features like area/perimeter have much larger ranges than others).
- There are no missing values in the dataset.
- The histograms show clear separation: malignant tumors generally have larger values for mean radius, mean perimeter, and mean area than benign tumors (texture overlaps more).

```
==================================================
DATA SPLIT
==================================================
Training set size: 455 samples
Test set size: 114 samples
Training features: 30
Test features: 30

Training set target distribution:
target
1    285
0    170
Name: count, dtype: int64
  Malignant (0): 170 (37.4%)
  Benign (1): 285 (62.6%)

Test set target distribution:
target
1    72
0    42
Name: count, dtype: int64
  Malignant (0): 42 (36.8%)
  Benign (1): 72 (63.2%)
```

Step 3 Observations

- I separated the dataset into features `X` (30 columns) and target `y` (569 labels).
- I split the data into 455 training samples (80%) and 114 test samples (20%) using `random_state=42` for reproducibility.
- Using `stratify=y` kept the malignant/benign proportions nearly the same in both train and test sets.

(Bootcamp-env) (Bootcamp-env) PS C:\Users\anson\OneDrive\Desktop\!IronHack\AI Labs\Week 1\Lab M1.03> & C:\Users\anson\anaconda3\envs\Bootcamp-env\python.exe "c:/Users/anson/OneDrive/Desktop/!IronHack/AI Labs/Week 1/Lab M1.03/breast_cancer_prediction.py"
Loading breast cancer dataset...

Dataset type: <class 'sklearn.utils._bunch.Bunch'>
Number of samples: 569
Number of features: 30
Target classes: ['malignant' 'benign']

First few rows:

```
   mean radius  mean texture  mean perimeter  mean area  ...  worst concave points  worst symmetry  worst fractal dimension  target
0       17.99         10.38          122.80     1001.0  ...                0.2654          0.4601                  0.11890       0
1       20.57         17.77          132.90     1326.0  ...                0.1860          0.2750                  0.08902       0
2       19.69         21.25          130.00     1203.0  ...                0.2430          0.3613                  0.08758       0
3       11.42         20.38           77.58      386.1  ...                0.2575          0.6638                  0.17300       0
4       20.29         14.34          135.10     1297.0  ...                0.1625          0.2364                  0.07678       0
```

[5 rows x 31 columns]

Dataset info:
<class 'pandas.DataFrame'>
RangeIndex: 569 entries, 0 to 568

```
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
None

Target distribution:
target
1    357
0    212
Name: count, dtype: int64
Malignant (0): 212
Benign (1): 357

==================================================
BASIC STATISTICS
```

```
=====================================================
      mean radius  mean texture  mean perimeter  ...  worst symmetry  worst fractal dimension    target
count   569.000000   569.000000     569.000000  ...      569.000000             569.000000  569.000000
mean     14.127292    19.289649      91.969033  ...        0.290076               0.083946    0.627417
std       3.524049     4.301036      24.298981  ...        0.061867               0.018061    0.483918
min       6.981000     9.710000      43.790000  ...        0.156500               0.055040    0.000000
25%      11.700000    16.170000      75.170000  ...        0.250400               0.071460    0.000000
50%      13.370000    18.840000      86.240000  ...        0.282200               0.080040    1.000000
75%      15.780000    21.800000     104.100000  ...        0.317900               0.092080    1.000000
max      28.110000    39.280000     188.500000  ...        0.663800               0.207500    1.000000

[8 rows x 31 columns]

=====================================================
MISSING VALUES
=====================================================
✓ No missing values found!

=====================================================
FEATURE DISTRIBUTIONS
=====================================================
Saved visualization to 'feature_distributions.png'
```

Model training

```python
print("KNN classifier trained successfully!")
print(f"Number of neighbors (k): {knn.n_neighbors}")

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

print(f"\nTraining predictions: {len(y_train_pred)}")
print(f"Test predictions: {len(y_test_pred)}")
```

Python

```
KNN classifier trained successfully!
Number of neighbors (k): 5

Training predictions: 455
Test predictions: 114
```

Step 4 Observations

- I trained a K-Nearest Neighbors (KNN) classifier with k = 5 using the training dataset.
- KNN predicts a new sample by finding the 5 most similar (closest) training samples and choosing the most common class among them.
- The model produced predictions for all 455 training samples and 114 test samples without errors.

```
==================================================
STEP 5: PREDICTIONS vs ACTUAL
==================================================

First 15 predictions vs actual:
    Actual  Predicted Actual_label Predicted_label
0        0          0    malignant       malignant
1        1          1       benign          benign
2        0          0    malignant       malignant
3        1          0       benign       malignant
4        0          0    malignant       malignant
5        1          1       benign          benign
6        1          1       benign          benign
7        0          0    malignant       malignant
8        0          0    malignant       malignant
9        0          0    malignant       malignant
10       1          1       benign          benign
11       0          0    malignant       malignant
12       1          1       benign          benign
13       0          0    malignant       malignant
14       0          0    malignant       malignant

Correct in first 15 examples: 14/15
...
76        0          1    malignant          benign
80        1          0       benign       malignant
99        1          0       benign       malignant
102       0          1    malignant          benign
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Step 5 Observations

- The model successfully generated predictions for every sample in the test set.
- In the first 15 test examples, 14 predictions matched the actual label.
- There were 10 total mismatches in the full test set, which we'll quantify more formally using evaluation metrics next.

Evaluation metrics

```
=== Model Performance ===

Training Accuracy: 0.9473 (94.73%)
Test Accuracy: 0.9123 (91.23%)

(Considering MALIGNANT as the positive class: target=0)
Test Precision (malignant): 0.8636
Test Recall (malignant):    0.9048

=== Confusion Matrix (rows=Actual, cols=Predicted) ===
Order: 0=malignant, 1=benign
[[38  4]
 [ 6 66]]

Interpretation:
Correct malignant (actual 0 → predicted 0): 38
Missed malignant  (actual 0 → predicted 1): 4
Benign flagged malignant (actual 1 → predicted 0): 6
Correct benign    (actual 1 → predicted 1): 66

=== Classification Report ===
              precision    recall  f1-score   support

   malignant       0.86      0.90      0.88        42
      benign       0.94      0.92      0.93        72
...
    accuracy                           0.91       114
   macro avg       0.90      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Step 6 Observations

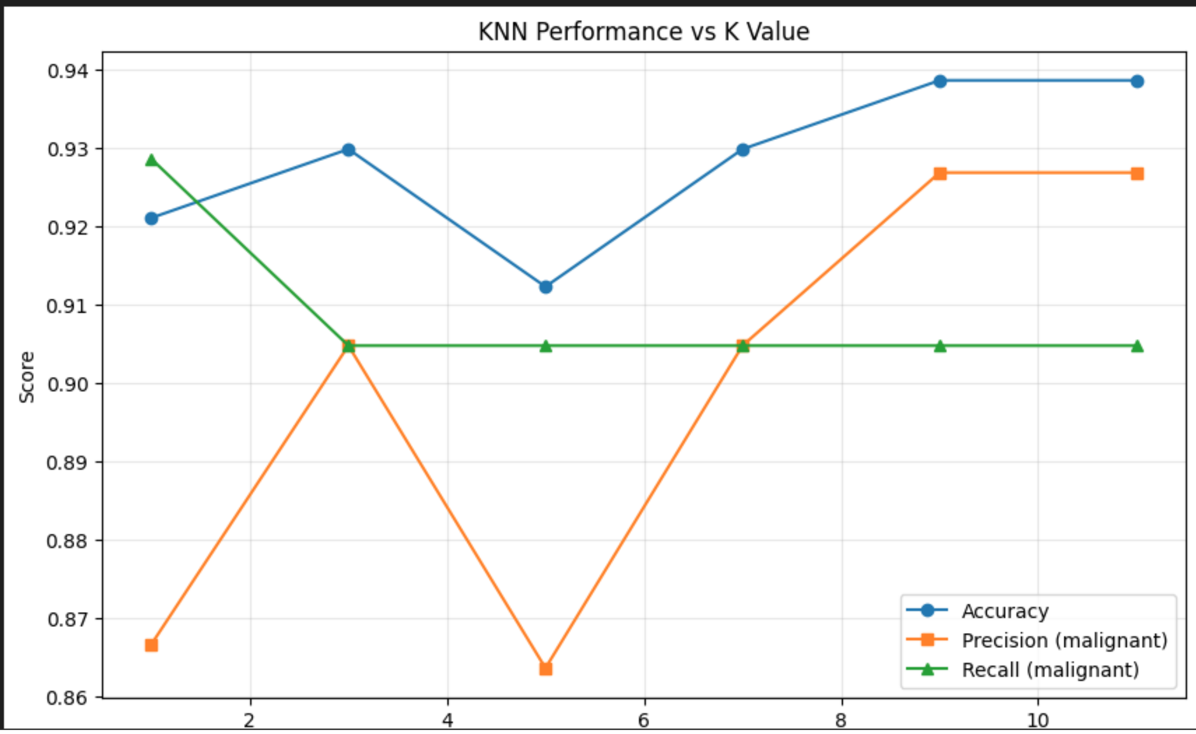
- The model achieved 94.73% training accuracy and 91.23% test accuracy, which suggests decent generalization with some errors on unseen data.
- Treating malignant as the "positive" class (target = 0), precision is 0.86 and recall is 0.90, meaning it correctly identifies most malignant cases but still misses some.
- The confusion matrix shows 4 missed malignant cases (malignant predicted as benign) and 6 false alarms (benign predicted as malignant); missed malignant cases are the most concerning in a medical screening context.

K value comparison results

Generate + Code + Markdown | Run All Clear All Outputs | Outline ··· Select Ke

```python
    # Make predictions
    y_pred_temp = knn_temp.predict(X_test)

    # Calculate metrics
    acc = accuracy_score(y_test, y_pred_temp)

    # malignant is class 0 in this dataset, so treat it as "positive"
    prec_malig = precision_score(y_test, y_pred_temp, pos_label=0)
    rec_malig = recall_score(y_test, y_pred_temp, pos_label=0)

    results.append({
        'K': k,
        'Accuracy': acc,
        'Precision_malignant': prec_malig,
        'Recall_malignant': rec_malig
    })

    print(f"K={k:2d}: Accuracy={acc:.4f}, Precision(malig)={prec_malig:.4f}, Recall(malig)={rec_malig:.4f}")

# Find best K by accuracy (simple approach)
results_df = pd.DataFrame(results)
best_k = results_df.loc[results_df['Accuracy'].idxmax(), 'K']
print(f"\nBest K value by Accuracy: {best_k} (Accuracy: {results_df['Accuracy'].max():.4f})")

# Visualize results
plt.figure(figsize=(10, 6))
plt.plot(results_df['K'], results_df['Accuracy'], marker='o', label='Accuracy')
plt.plot(results_df['K'], results_df['Precision_malignant'], marker='s', label='Precision (malignant)')
plt.plot(results_df['K'], results_df['Recall_malignant'], marker='^', label='Recall (malignant)')
plt.xlabel('K (Number of Neighbors)')
plt.ylabel('Score')
plt.title('KNN Performance vs K Value')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('knn_k_comparison.png', dpi=150, bbox_inches='tight')
print("\nSaved visualization to 'knn_k_comparison.png'")
plt.show()

results_df
```

```
==================================================
EXPERIMENTING WITH DIFFERENT K VALUES
==================================================
K= 1: Accuracy=0.9211, Precision(malig)=0.8667, Recall(malig)=0.9286
K= 3: Accuracy=0.9298, Precision(malig)=0.9048, Recall(malig)=0.9048
K= 5: Accuracy=0.9123, Precision(malig)=0.8636, Recall(malig)=0.9048
K= 7: Accuracy=0.9298, Precision(malig)=0.9048, Recall(malig)=0.9048
K= 9: Accuracy=0.9386, Precision(malig)=0.9268, Recall(malig)=0.9048
K=11: Accuracy=0.9386, Precision(malig)=0.9268, Recall(malig)=0.9048

Best K value by Accuracy: 9 (Accuracy: 0.9386)

Saved visualization to 'knn_k_comparison.png'
```



	K	Accuracy	Precision_malignant	Recall_malignant
0	1	0.921053	0.866667	0.928571
1	3	0.929825	0.904762	0.904762
2	5	0.912281	0.863636	0.904762
3	7	0.929825	0.904762	0.904762
4	9	0.938596	0.926829	0.904762
5	11	0.938596	0.926829	0.904762

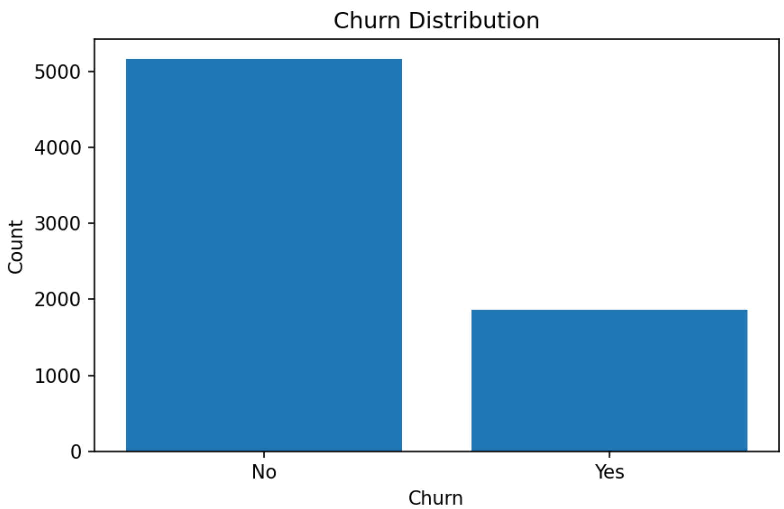
Step 7 Observations

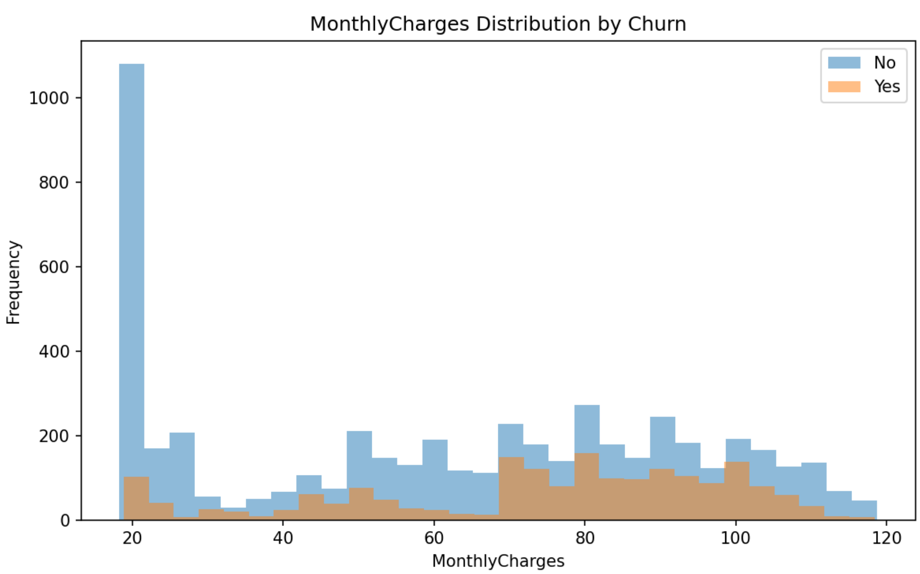
- Changing K changes performance: smaller K can be more sensitive (higher recall) but may create more false alarms, while larger K smooths decisions and can improve stability.
- In my results, K=9 (and K=11) achieved the highest test accuracy (~0.939) and the highest precision for malignant (~0.927), while recall for malignant stayed ~0.905.
- If the priority is to catch as many malignant cases as possible, K=1 had the highest malignant recall (~0.929), but with lower precision (more false positives).

Part 2 (Unguided Exercise):

Screenshots or output showing:

- Data exploration and preprocessing
- Model training and evaluation
- Comparison of different K values





```
====================================================
STEP 2: PREPROCESSING
====================================================
Missing values after TotalCharges conversion:
TotalCharges      11
gender             0
Partner            0
SeniorCitizen      0
Dependents         0
tenure             0
MultipleLines      0
PhoneService       0
OnlineSecurity     0
OnlineBackup       0
dtype: int64

Encoded shape (rows, columns): (7043, 31)
X shape: (7043, 30)
y shape: (7043,)

Target distribution after encoding:
Churn
0    5174
1    1869
Name: count, dtype: int64
Churn rate (%): 26.54
```

```
====================================================
STEP 3: TRAIN/TEST SPLIT
====================================================
Train size: 5634
Test size: 1409

Train churn distribution:
Churn
0    4139
1    1495
Name: count, dtype: int64
Train churn rate (%): 26.54

Test churn distribution:
Churn
0    1035
1     374
Name: count, dtype: int64
Test churn rate (%): 26.54
```

```
======================================================
STEP 4: TRAIN KNN (k=5)
======================================================
Trained KNN with k=5
Train predictions: 5634
Test predictions: 1409
```

```
======================================================
STEP 5: EVALUATION (positive = churn=1)
======================================================
Training Accuracy: 0.8277 (82.77%)
Test Accuracy:     0.7658 (76.58%)
Test Precision (churn): 0.5791
Test Recall (churn):    0.4305

Confusion Matrix (rows=Actual, cols=Predicted)
Order: 0=No churn, 1=Churn
[[918 117]
 [213 161]]

Classification Report:
              precision    recall  f1-score   support

          No       0.81      0.89      0.85      1035
         Yes       0.58      0.43      0.49       374

    accuracy                           0.77      1409
   macro avg       0.70      0.66      0.67      1409
weighted avg       0.75      0.77      0.75      1409

(Bootcamp-env) (Bootcamp-env) PS C:\Users\anson\OneDrive\Desktop\!IronHack\AI Labs\Week 1\Lab M1.03>
```

```
==================================================
STEP 6: K COMPARISON
==================================================
K= 1: Accuracy=0.7119, Precision=0.4570, Recall=0.4545
K= 3: Accuracy=0.7622, Precision=0.5657, Recall=0.4492
K= 5: Accuracy=0.7658, Precision=0.5791, Recall=0.4305
K= 7: Accuracy=0.7814, Precision=0.6260, Recall=0.4385
K= 9: Accuracy=0.7885, Precision=0.6532, Recall=0.4332
K=11: Accuracy=0.7871, Precision=0.6623, Recall=0.4037
K=15: Accuracy=0.7885, Precision=0.6681, Recall=0.4037

Results table:
    K  Accuracy  Precision    Recall
0   1  0.711852   0.456989  0.454545
1   3  0.762243   0.565657  0.449198
2   5  0.765791   0.579137  0.430481
3   7  0.781405   0.625954  0.438503
4   9  0.788502   0.653226  0.433155
5  11  0.787083   0.662281  0.403743
6  15  0.788502   0.668142  0.403743

Best K by Recall (catching churners): 1
Best K by Accuracy: 9
```

A brief report (1-2 pages) including:

Summary of approach

- Model performance metrics
- Key findings about customer churn
- Business recommendations
- Limitations and future improvements

Telco Customer Churn Prediction (KNN) — Brief Report

1) Summary of approach

I built a supervised classification model to predict whether a telecom customer will churn (Yes/No) using the Telco Customer Churn dataset (7,043 customers; 21 columns). I followed a standard ML workflow: load and explore the dataset, preprocess/encode features, split into training and test sets (80/20 with stratification), train a K-Nearest Neighbors (KNN) classifier, evaluate using accuracy/precision/recall + confusion matrix, and then tune the number of neighbors (K). Because KNN is distance-based and sensitive to feature scale, I also implemented an improved sklearn Pipeline that standardizes numeric features and one-hot encodes categoricals in a single reproducible workflow.

2) Data exploration (what I observed)

- The target variable is imbalanced: **Churn = Yes: 1,869 (26.54%)**, **Churn = No: 5,174 (73.46%)**.

- TotalCharges was loaded as a string; converting it to numeric revealed **11 missing/blank values**, which were filled using the median.
- A quick histogram comparison suggested **churners tend to have higher MonthlyCharges** (the churn “Yes” distribution is more concentrated at higher monthly costs), even though the total count of churners is lower.

3) Preprocessing

- Dropped customerID (identifier, not predictive).
- Converted TotalCharges to numeric and filled missing values with the median.
- Converted the target Churn to binary: **No = 0**, **Yes = 1**.
- Encoded categorical variables using one-hot encoding (either manually via get_dummies or via OneHotEncoder in a Pipeline).
- Train/test split: **Train = 5,634**, **Test = 1,409**, with identical churn rate (**26.54%**) due to stratification.

4) Model performance metrics

Baseline KNN (manual one-hot, no scaling)

Using KNN with **K=5**:

- **Test Accuracy: 0.7658**
- **Precision (Churn=Yes): 0.5791**
- **Recall (Churn=Yes): 0.4305**
- **Confusion matrix** (rows=actual, cols=pred; 0=No, 1=Yes):
  [
  \begin{bmatrix}
  918 & 117 \
  213 & 161
  \end{bmatrix}
  ]
  Interpretation: the model correctly flagged **161 churners**, but **missed 213 churners** (low recall).

K sweep (no scaling) showed:

- Best recall was **K=1** with **Recall ~0.4545**
- Best accuracy was **K=9/15** with **Accuracy ~0.7885** (but recall stayed ~0.40–0.43)

Improved KNN (Pipeline: scaling numeric + one-hot encoding categoricals)

With a proper preprocessing Pipeline, performance improved substantially. Best results were at **K=9**:

- **Accuracy: 0.7800**
- **Precision (Churn=Yes): 0.5856**
- **Recall (Churn=Yes): 0.5856**

This is a major improvement in recall (catching churners) versus the unscaled version (~0.43–0.46 recall).

5) Key findings about customer churn

- **Churn is meaningfully imbalanced** (~26.5% churn), so accuracy alone is not sufficient; recall/precision for churners is more important for retention actions.
- **MonthlyCharges appears associated with churn** (churners skew toward higher monthly costs). This is consistent with the idea that higher monthly bills can increase churn risk, especially if perceived value is low.
- **Preprocessing quality strongly affects KNN**, especially scaling numeric features. The pipeline approach improved churn recall from ~0.43 to ~0.59.

6) Business recommendations

1. **Use the pipeline KNN (K=9) as the best baseline** from this exercise. It catches ~59% of churners, which is meaningfully better than the unscaled KNN.
2. **Operationalize around recall vs precision tradeoffs**:
    - If the business wants to catch more churners (maximize retention saves), prioritize **recall** even if precision drops (more outreach).
    - If outreach capacity is limited, optimize for a better precision/recall balance and target the highest-risk segment only.
3. **Target interventions where the model suggests higher churn risk**, especially customers with **higher MonthlyCharges**. Possible actions: proactive discounts, service upgrades, contract incentives, improved support, or "value reinforcement" messaging.
4. **Next analytics step (quick wins):** segment churn rates by Contract type, tenure bands, PaymentMethod, and InternetService to identify the strongest levers for retention campaigns.

7) Limitations and future improvements

- **Model limitation (KNN):** KNN can struggle with high-dimensional one-hot encoded data and is sensitive to scaling; while the pipeline helps, KNN may still be suboptimal for this dataset.
- **Metric limitation:** We did not optimize thresholds (KNN outputs class labels by default). For retention, probability scores + threshold tuning are often better.
- **Feature importance:** KNN is not easily interpretable for feature importance. For clearer drivers of churn, consider more interpretable models.
- **Future improvements:**
    1. Try **Logistic Regression** (interpretable coefficients), **Random Forest**, or **Gradient Boosting** and compare recall/precision/F1 for churners.
    2. Use **cross-validation** and a more systematic hyperparameter search.
    3. Evaluate **ROC-AUC / PR-AUC** and tune decision thresholds based on business cost of false negatives vs false positives.
    4. Consider handling class imbalance explicitly (e.g., class weights in other models, resampling methods).