Oregon State University

COLLEGE OF ENGINEERING

Image created using DALL-E

SPACESHIP TITANIC

Melek Derman

12/09/2024

What is Spaceship Titanic dataset?

- Spaceship Titanic is a Kaggle competition to help learners understand the basics of Machine Learning.
- It is a classification problem.
- It has more than 8000 data in train set.

Problem Description


Image created using DALL-E

- It is the year 2912. A transmission from 4 lightyears away has been received.
- An interstellar passenger liner, the Spaceship Titanic, was carrying passengers from our solar system to three newly habitable exoplanets orbiting nearby stars.
- Devastatingly, the ship collided with a spacetime anomaly hidden within a dust cloud, resulting in half of the passengers being transported to an alternate dimension!
- Our goal is to retrieve the lost passengers by predicting which passengers were transported using the records. Help save them and change history!

Oregon State University
College of Engineering

Dataset and Features

- There are 8,693 passenger records
- 6 numerical and 6 categorical features
- Numerical Features:
  - Age, RoomService, FoodCourt, ShoppingMall, Spa, VRDeck
- Categorical Features:
  - HomePlanet, CryoSleep, Cabin, Destination, VIP, Name
- The features are mostly unbalanced.
- There are missing values in all categories.

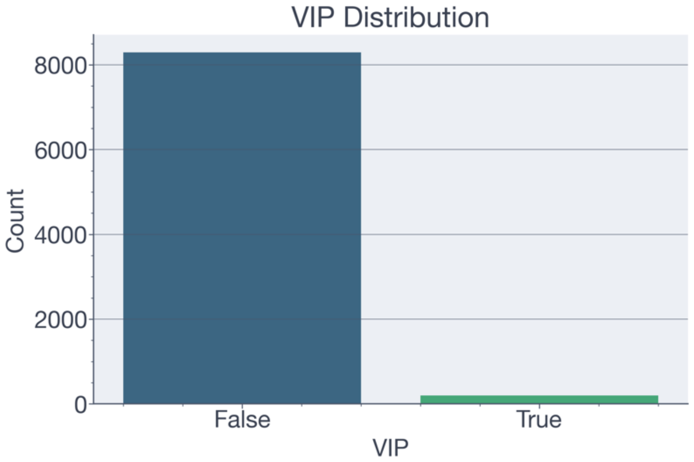
```
Train data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   8693 non-null   object
 1   HomePlanet    8492 non-null   object
 2   CryoSleep     8476 non-null   object
 3   Cabin         8494 non-null   object
 4   Destination   8511 non-null   object
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object
```

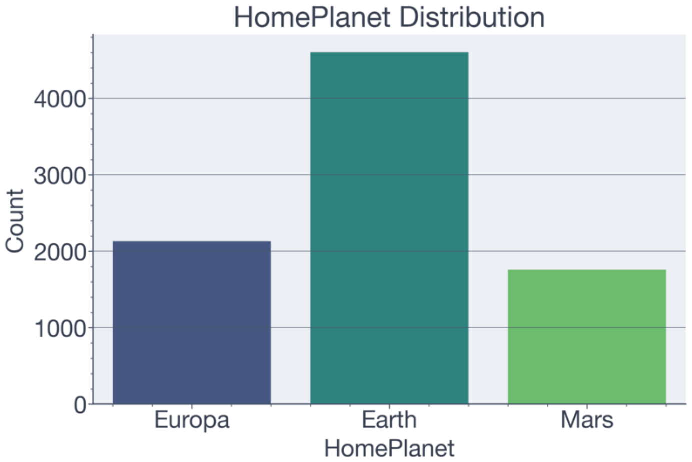
```
Number of unique values in each column:
PassengerId     8693
HomePlanet         3
CryoSleep          2
Cabin           6560
Destination        3
Age               80
VIP                2
RoomService     1273
FoodCourt       1507
ShoppingMall    1115
Spa             1327
VRDeck          1306
Name            8473
Transported        2
```

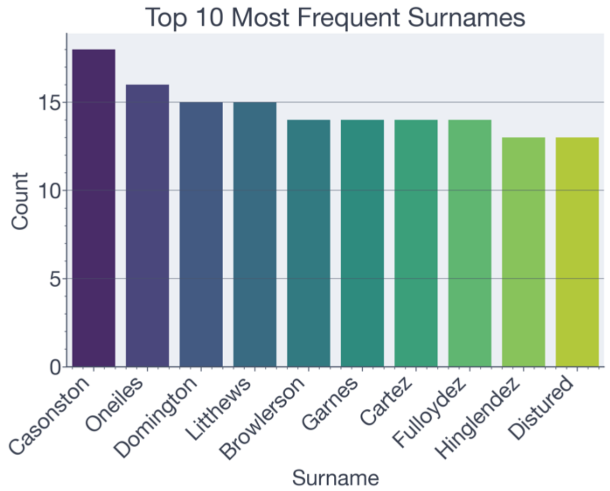
Missing Values:

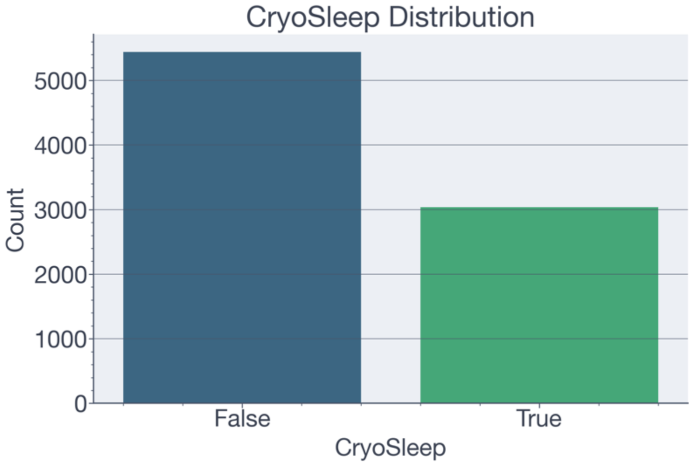
```
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
```

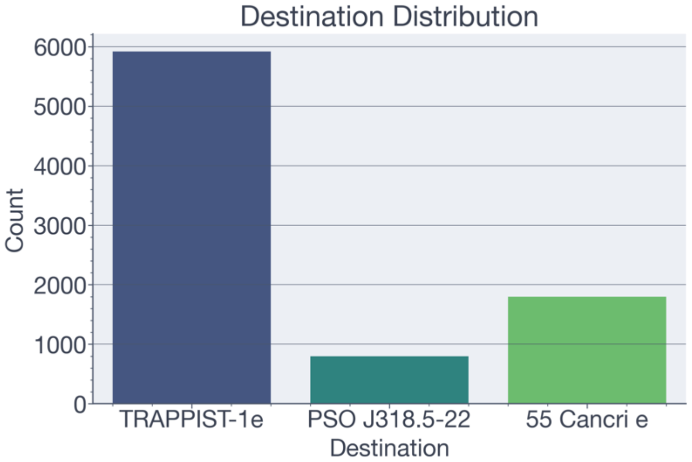
- Categorical Features

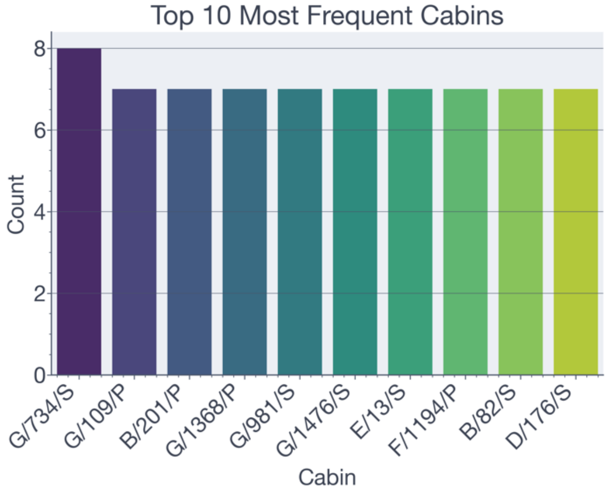






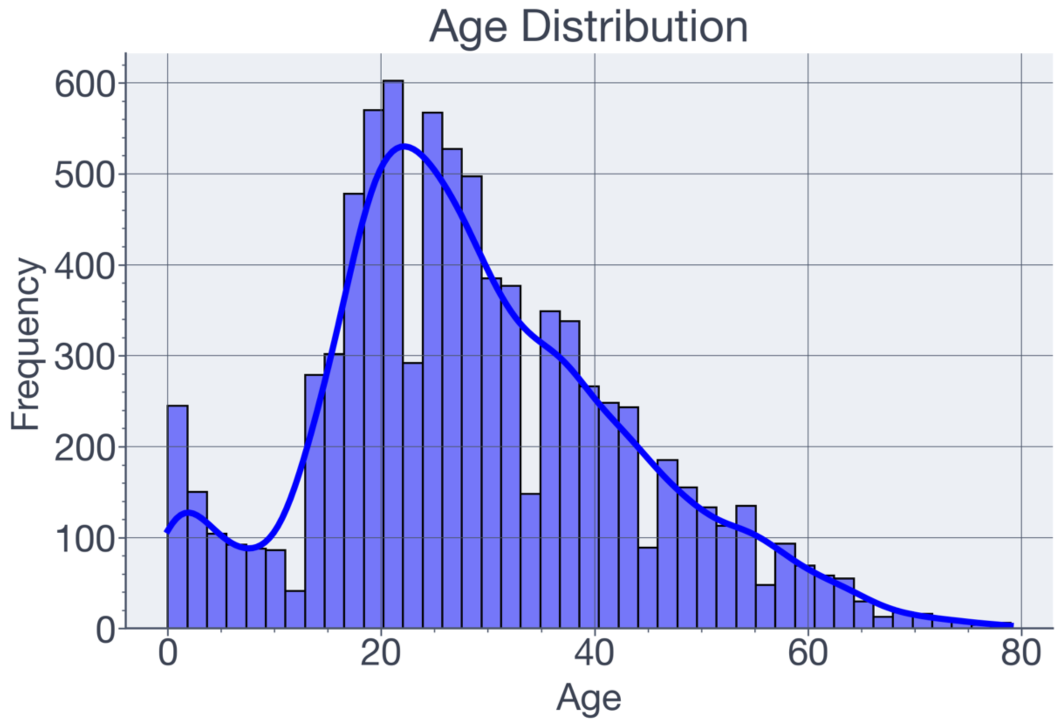




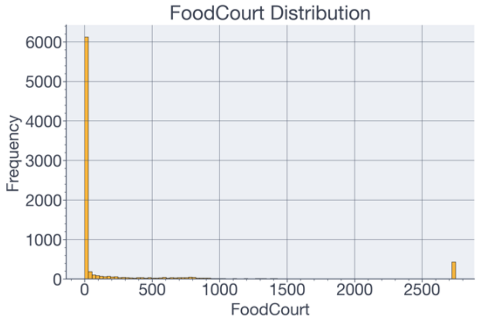


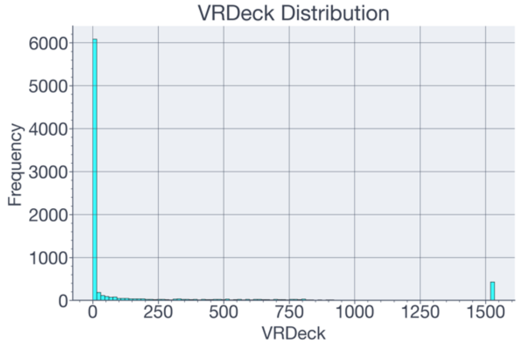
* CryoSleep indicates whether the passenger elected to be put into suspended animation for the duration of the voyage.

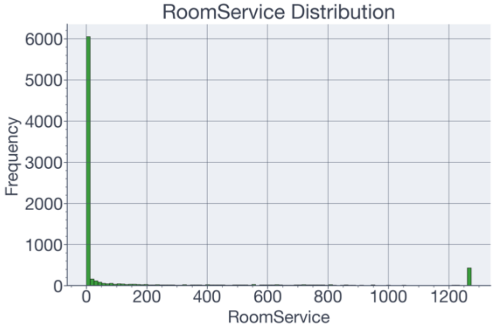
- Numerical Features

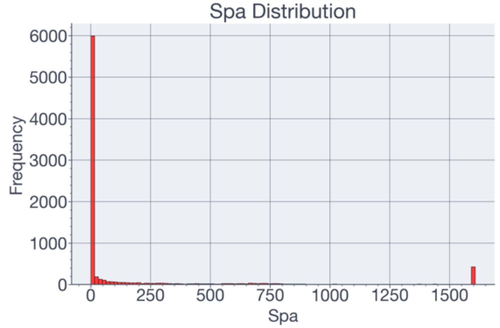


* RoomService, FoodCourt, ShoppingMall, Spa, VRDeck - Amount the passenger has billed at each of the *Spaceship Titanic*'s many luxury amenities.









Methodology for Features

- The “Name” column was split into individual words, enabling the analysis of family members.
- For missing values, the ‘mean’ strategy was used for numerical columns, and the ‘most_frequent’ strategy was applied for categorical features (Simple Imputer)
- Numerical Features → Standard Scaler
- Categorical Features → OneHotEncoder

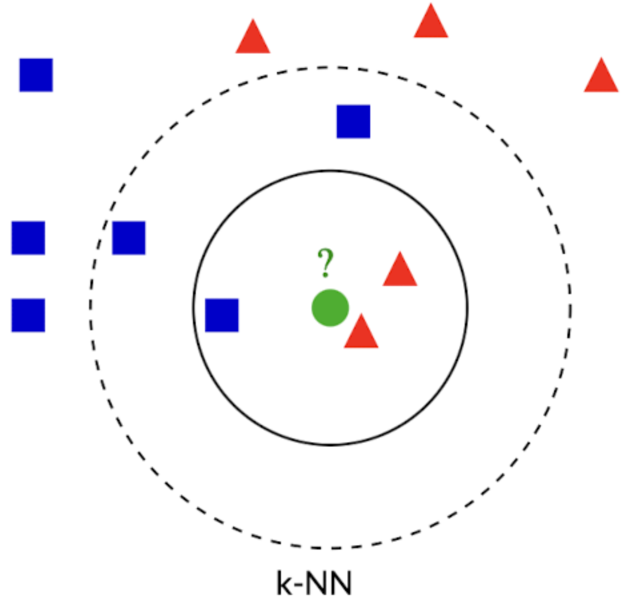
```
numerical_features = X_train.select_dtypes(include=['float64']).columns.tolist()
categorical_features = X_train.select_dtypes(include=['object']).columns.tolist()

numerical_features = [col for col in num_cols if col in X.columns]
categorical_features = [col for col in X.columns if col not in numerical_features]

numerical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_features),
    ('cat', categorical_transformer, categorical_features)
])
```

Methodology – k Nearest Neighbor Classifier

- The nearest neighbor classifier is a simple machine learning algorithm that is often used for classification tasks.
- It works by finding the "nearest" training example to a new, unlabeled example in the feature space, and assigning the label of that nearest example to the new example. Predictions are based on the similarity of a new data point to the data points in the training set.
- The algorithm is called "lazy" because it does not actually learn a model from the training data; instead, it simply memorizes the training examples and uses them to make predictions at test time.

Methodology _ Nearest Neighbor Classifier

- For a given data point (test point), KNN calculates its distance to all other points in the dataset.
- It selects the k nearest neighbors (based on the smallest distances).
- KNN assumes that similar data points are close to each other in the feature space. Therefore, the prediction for a new data point is based on its proximity to other known data points.



Oregon State University
College of Engineering

Methodology K-NN Hyperparameters

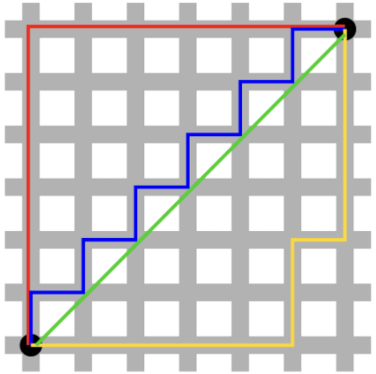
- **Number of Neighbors (k):** Determines how many closest neighbors are considered when making a prediction.
  - Small k: The model is sensitive to noise and may overfit.
  - Large k: The model becomes too generalized and may underfit.
- **Distance Metric:** The choice of distance metric determines how the algorithm measures similarity between points.
  - **Euclidean Distance:** Works well when the features have similar scales.
  - **Manhattan Distance:** More robust when dealing with grid-like data or features with varying scales

Now the Euclidean distance is simply the ℓ_2-norm:

$$\ell_2(x, y) = \sqrt{|x_1 - y_1|^2 + \ldots + |x_d - y_d|^2}$$

and the Manhattan distance is simply the ℓ_1-norm:

$$\ell_1(x, y) = |x_1 - y_1| + \ldots + |x_d - y_d|$$



Manhattan vs. Euclidean Distances

- **Weighting of Neighbors:** Neighbors can be weighted based on their distance to the test point:
  - Uniform Weighting: All neighbors have equal importance.
  - Distance Weighting: Closer neighbors have more influence.

Methodology - Tuning

To optimize the performance of the KNN algorithm, we need to tune the hyperparameters:

Split the Data:

- Use the training data to tune the hyperparameters using cross-validation and evaluate the performance on a separate test set.

Hyperparameter Tuning:

- k: Test odd values such as [1, 3, 5, 7, 9, 11, ...] to avoid tie-breaking votes.
- Distance metric: Try Euclidean, Manhattan.
- Weighting: Test both uniform and distance-based weighting.
- Use cross-validation to test each combination of hyperparameters and select the combination that gives the best performance.

Optimal Hyperparameters:

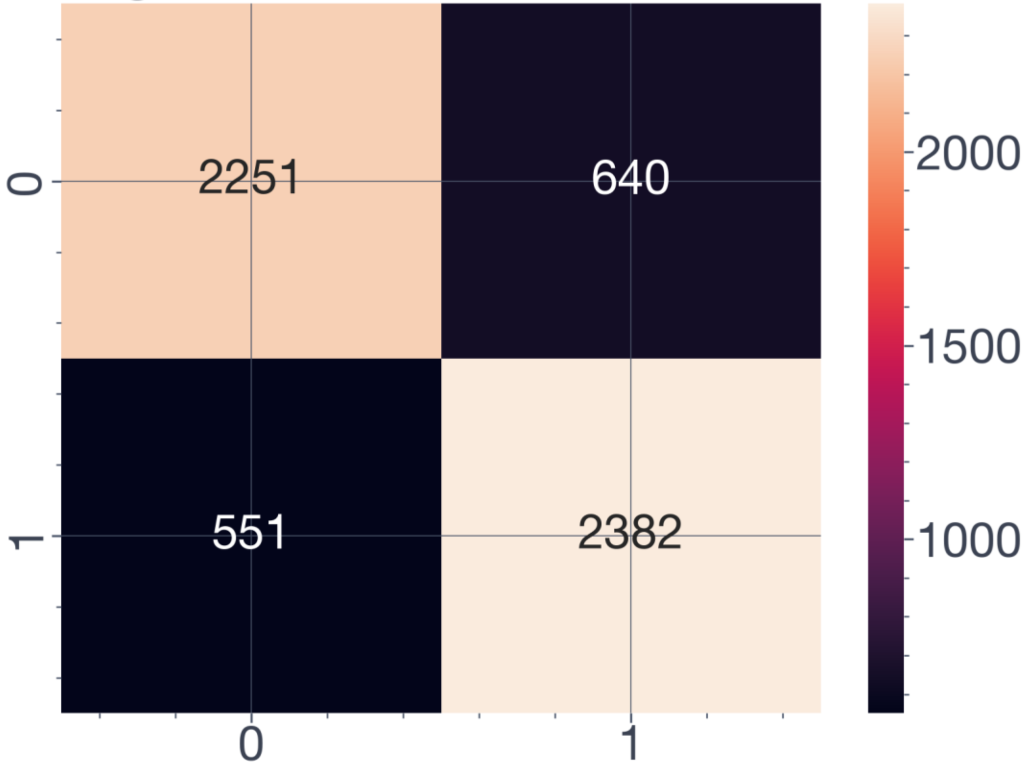
- After testing various combinations, select the hyperparameters that maximize a performance metric (e.g., accuracy, precision, recall).

Final Evaluation:

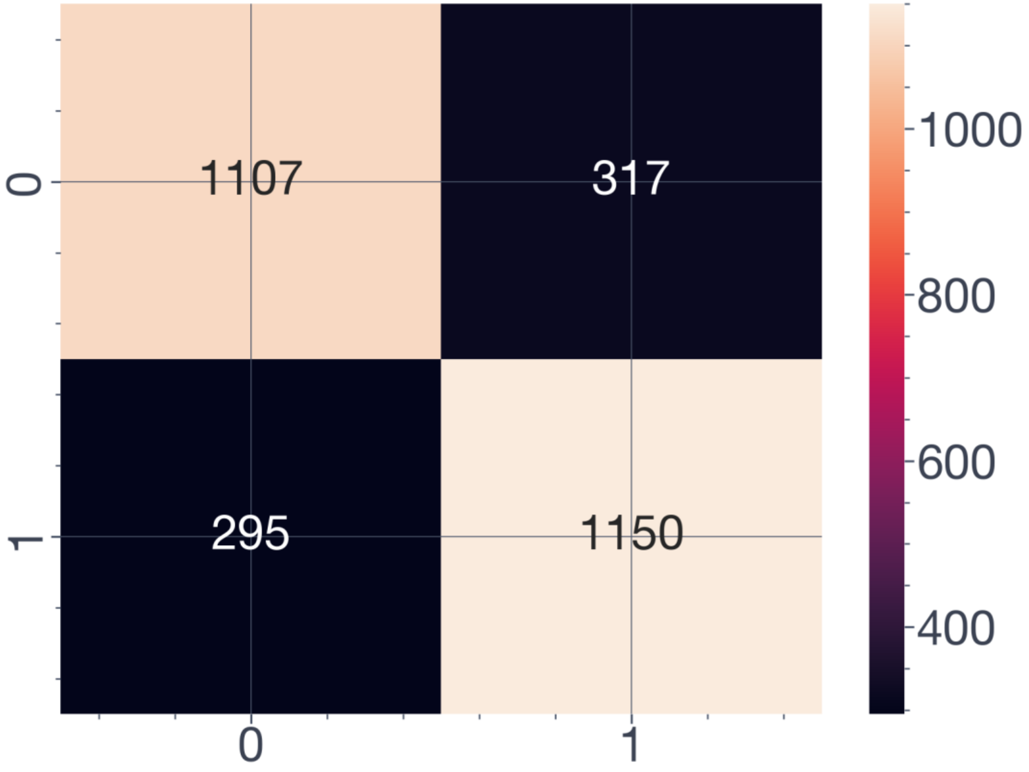
- Train the KNN model on the entire training set using the optimal hyperparameters.
- Evaluate the performance on the test set and submit the test predictions to Kaggle.

Results

Training Set Confusion Matrix for k-nn



Test Set Confusion Matrix for k-nn



Oregon State University College of Engineering

Results

```
Best hyperparameters: n_neighbors=41, metric=manhattan, weight=uniform with a balanced accuracy of 0.7828
Balanced Accuracy for k-NN on train set: 0.7953805283288025
Accuracy for k-NN on train set: 0.7955013736263736
Precision for k-NN on train set: 0.7957872343239749
Recall for k-NN on train set: 0.7953805283288025
Balanced Accuracy for k-NN on test set: 0.786617695657245
Accuracy for k-NN on test set: 0.7866852561868247
Precision for k-NN on test set: 0.7867495261905526
Recall for k-NN on test set: 0.786617695657245
```

```
Cross-validation balanced accuracy scores for each combination:
n_neighbors=1, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7109
n_neighbors=3, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7476
n_neighbors=5, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7622
n_neighbors=7, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7694
n_neighbors=9, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7689
n_neighbors=11, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7699
n_neighbors=13, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7706
n_neighbors=15, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7785
n_neighbors=17, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7784
n_neighbors=19, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7771
n_neighbors=21, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7772
n_neighbors=23, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7778
n_neighbors=25, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7783
n_neighbors=27, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7788
n_neighbors=29, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7797
n_neighbors=31, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7800
n_neighbors=33, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7805
n_neighbors=35, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7788
n_neighbors=37, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7819
n_neighbors=39, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7805
n_neighbors=41, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7793
n_neighbors=43, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7792
n_neighbors=45, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7821
n_neighbors=47, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7823
n_neighbors=49, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7816
n_neighbors=51, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7809
n_neighbors=53, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7795
n_neighbors=55, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7799
n_neighbors=57, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7811
n_neighbors=59, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7809
n_neighbors=61, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7808
n_neighbors=63, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7797
n_neighbors=65, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7806
n_neighbors=67, metric=euclidean, weight=uniform: Mean Balanced Accuracy = 0.7816
```

```
n_neighbors=41, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7813
n_neighbors=43, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7820
n_neighbors=45, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7808
n_neighbors=47, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7792
n_neighbors=49, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7796
n_neighbors=51, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7798
n_neighbors=53, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7812
n_neighbors=55, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7806
n_neighbors=57, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7807
n_neighbors=59, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7808
n_neighbors=61, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7803
n_neighbors=63, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7796
n_neighbors=65, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7807
n_neighbors=67, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7807
n_neighbors=69, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7812
n_neighbors=71, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7817
n_neighbors=73, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7814
n_neighbors=75, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7800
n_neighbors=77, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7806
n_neighbors=79, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7801
n_neighbors=81, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7808
n_neighbors=83, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7799
n_neighbors=85, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7803
n_neighbors=87, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7808
n_neighbors=89, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7796
n_neighbors=91, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7800
n_neighbors=93, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7803
n_neighbors=95, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7801
n_neighbors=97, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7803
n_neighbors=99, metric=manhattan, weight=distance: Mean Balanced Accuracy = 0.7802
```

Image created using DALL-E

Oregon State University
College of Engineering

Thank you!

Questions?

Oregon State University
College of Engineering

Kaggle Test Results

final_submission_svm.csv Complete · 7h ago	0.79331
final_submission.csv Complete · 8h ago · knn	0.75941
sample_submission.csv Complete · 5d ago	0.49310