# Used Imports:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

# Load the dataset:

```

file_path = "C:/Users/XPSTORE/Downloads/magic04 (1).data"
col_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',
'fM3Long', 'fM3Trans', 'fAlpha', 'fDist',
             'class']
data = pd.read_csv(file_path, names=col_names)
```

# Separate the gamma and hadron events based on the class column:

```
gamma_events = data[data['class'] == 'g']
hadron_events = data[data['class'] == 'h']
```

# Count the number of hadron events:

```
hadron_count = len(hadron_events)
```

# Sample gamma events to match the number of hadron events:

```
gamma_subset = gamma_events.sample(n=hadron_count, random_state=42)
```

# Create a balanced dataset by concatenating the gamma subset with hadron events:

```
# Create a balanced dataset by concatenating the gamma subset with
hadron events:
balanced_data = pd.concat([gamma_subset, hadron_events])
```

# Shuffle the combined dataset to ensure randomness:

```
# Shuffle the combined dataset to ensure randomness:
balanced_data = balanced_data.sample(frac=1,
random_state=42).reset_index(drop=True)
```

# Split the balanced dataset into training and the rest

```
training_set, remaining_data = train_test_split(balanced_data,
test_size=0.3, random_state=42)
```

# Split the remaining data into validation and testing sets

```
# Split the remaining data into validation and testing sets
validation_set, test_set = train_test_split(remaining_data,
test_size=0.5, random_state=42)
```

# Prepare the training data and labels

```
# Prepare the training data and labels
X_train = training_set.drop(columns=['class'])
y_train = training_set['class']
```

# Prepare the validation data and labels

```
# Prepare the validation data and labels
X_val = validation_set.drop(columns=['class'])
y_val = validation_set['class']
```

# Prepare the test data and labels

```
# Prepare the test data and labels
X_test = test_set.drop(columns=['class'])
y_test = test_set['class']
```

# Define a range of values for k

```
# Define a range of values for k
k_values = range(1, 21)
```

# Initialize lists to store accuracy scores

```
# Initialize lists to store accuracy scores
accuracy_scores = []
```

#Iterate over different values of k

```
# Iterate over different values of k
for k in k_values:
    # Initialize the k-NN classifier with the current k
    knn_classifier = KNeighborsClassifier(n_neighbors=k)

    # Train the classifier on the training data
    knn_classifier.fit(X_train, y_train)
    # Predict the classes for validation data
    y_val_pred = knn_classifier.predict(X_val)

    # Calculate accuracy score for the current k value
    accuracy = accuracy_score(y_val, y_val_pred)

    # Append accuracy score to the list
    accuracy_scores.append(accuracy)
    conf_matrix = confusion_matrix(y_val, y_val_pred)
```

# Print evaluation metrics for the current k value

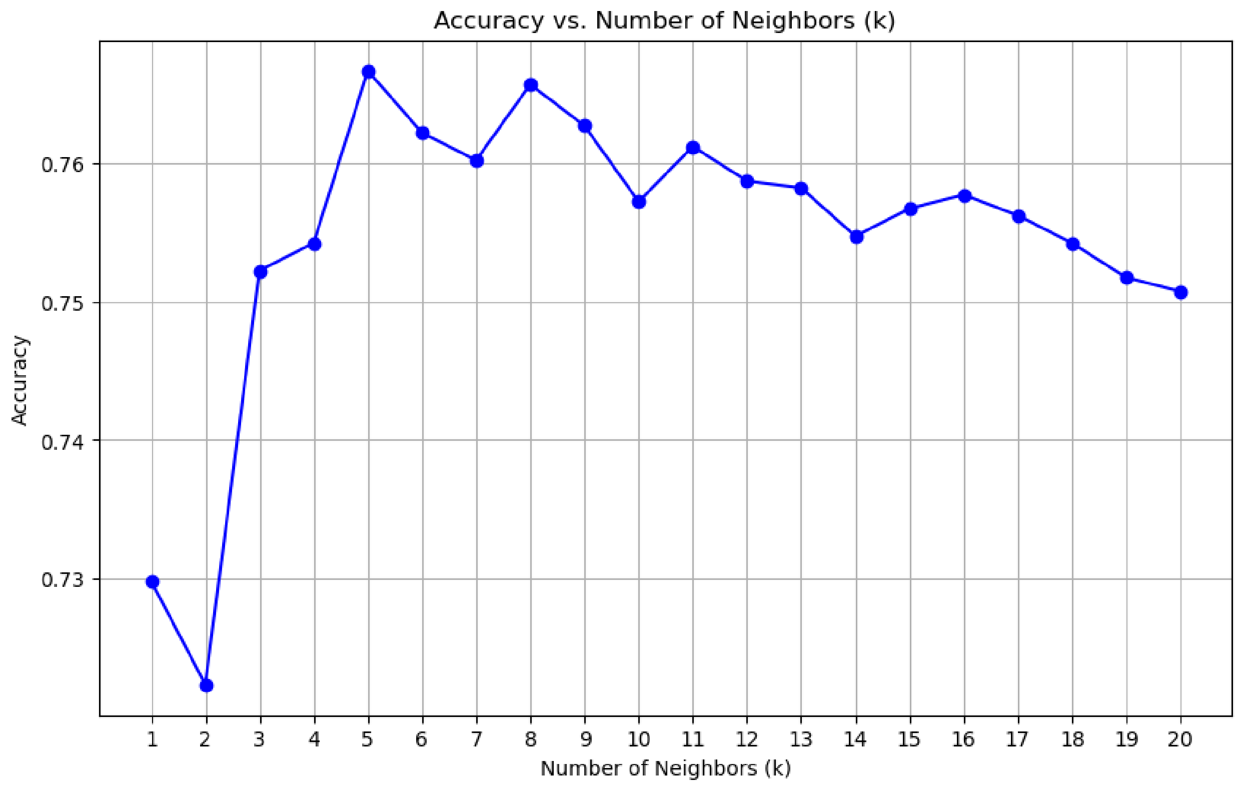
```
# Cell 16
# Print evaluation metrics for the current k value
print(f"k = {k}:")
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_val, y_val_pred))
print("Confusion Matrix:")
print(conf_matrix)
print("-------------------------------------")
```

```
k = 20:
Accuracy: 0.7507477567298105
Classification Report:
              precision    recall  f1-score   support

           g       0.70      0.87      0.78      1002
           h       0.83      0.63      0.72      1004

    accuracy                           0.75      2006
   macro avg       0.77      0.75      0.75      2006
weighted avg       0.77      0.75      0.75      2006

Confusion Matrix:
[[872 130]
 [370 634]]
-------------------------------------
```

# Plot accuracy scores for different values of k

```
# Plot accuracy scores for different values of k
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='-',
color='b')
plt.title('Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
```

```python
plt.xticks(np.arange(1, 21, step=1))
plt.grid(True)
plt.show()
```


Accuracy vs. Number of Neighbors (k)
Accuracy
Number of Neighbors (k)