

✓ Code for CSSE415: Group 3, "Bird Species Prediction"

✓ Imports

✓ Download bird data from Google Drive

```
!pip install -U gdown
import gdown
```

```
url = "https://drive.google.com/uc?id=1hZr6QwZHTj3jYwLzgJrcMi8k11_6lLnq"
gdown.download(url, output="family_included_full_data.csv", quiet=False)
```

```
➞ Requirement already satisfied: gdown in /usr/local/lib/python3.11/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.11/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4) (2.5)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from typing_extensions) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests[socks]) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests[socks]) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests[socks]) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests[socks]) (2024.7.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.11/dist-packages (from requests[socks]) (1.7.1)
Downloading...
From: https://drive.google.com/uc?id=1hZr6QwZHTj3jYwLzgJrcMi8k11_6lLnq
To: /content/family_included_full_data.csv
100%|██████████| 16.7M/16.7M [00:00<00:00, 213MB/s]
'family_included_full_data.csv'
```

```
# Core libraries
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import time
```

```
# Scikit-learn
```

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_validate
from sklearn.preprocessing import StandardScaler, LabelEncoder, OrdinalEncoder, PolynomialFeatures
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, GradientBoostingRegressor
from sklearn.pipeline import Pipeline
```

```
df = pd.read_csv('family_included_full_data.csv', encoding='latin-1') # had to use latin-1 encoding, other
```

```
df.drop(["Unnamed: 0", "Species3_BirdTree", "Species1_BirdLife", "Species2_eBird", "eBird.species.group",
dff = df.dropna(axis=0)
print(dff.shape)
X = dff.drop(['Family3'], axis=1)
y = dff['Family3']
```

↔ (45632, 12)

```
# See how dropping NA for remaining features impacts number of remaining records
features = df.columns
before_size = [ ]
after_size = [ ]
na_feature = [ ]
for f in features:
    before_size.append(df.shape[0])
    data = df.dropna(subset = [f])
    after_size.append(data.shape[0])
    na_feature.append(f)
```

```
dropNA_results = pd.DataFrame()
dropNA_results['before_size'] = before_size
dropNA_results['after_size'] = after_size
dropNA_results['NA_feature'] = na_feature
dropNA_results['change_size'] = dropNA_results['before_size']-dropNA_results['after_size']
dropNA_results_Ordered = dropNA_results.sort_values(by='change_size',ascending=False)
dropNA_results_Ordered.head(10)
# plt.plot(dropNA_results['NA_feature'],dropNA_results['change_size'])
```



	before_size	after_size	NA_feature	change_size
2	90303	55692	Beak.Length_Nares	34611
9	90303	62645	Hand-wing.Index	27658
7	90303	62678	Kipps.Distance	27625
8	90303	64169	Secondary1	26134
3	90303	70277	Beak.Width	20026
1	90303	74172	Beak.Length_Culmen	16131
4	90303	75318	Beak.Depth	14985
5	90303	80460	Tarsus.Length	9843
10	90303	83551	Tail.Length	6752
6	90303	89219	Wing.Length	1084

✓ Preprocess Data

```
cat_cols = X.select_dtypes(include='object').columns
num_cols = X.select_dtypes(exclude='object').columns
```

```
X_cat = X[cat_cols].fillna("missing").astype(str)
encoder = OrdinalEncoder()
X_cat_encoded = encoder.fit_transform(X_cat)
```

```
X_cat_encoded_df = pd.DataFrame(X_cat_encoded, columns=cat_cols, index=X.index)
```

```
X_num = X[num_cols].fillna(0)
X_processed = pd.concat([X_num, X_cat_encoded_df], axis=1)
std = StandardScaler().set_output(transform="pandas")
X_processed = std.fit_transform(X_processed)
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y_encoded, test_size=0.20, random_state=42
)
```

```
print("Rows: ", len(X_train) + len(X_test))
print("Features: ", len(X.iloc[0]))
```

```
➡ Rows: 45632
   Features: 11
```

✓ Feature Selection

```
def SelectFeature(current_model, remaining_features, X, y):
    scores = []
    for feature in remaining_features:
        trial_features = current_model + [feature]
        r2 = cross_validate(
            LinearRegression(fit_intercept=True),
            X[trial_features],
            y,
            cv=5,
            scoring='r2'
        )['test_score'].mean()
        scores.append(r2)
    best_feature_index = np.argmax(scores)
    return remaining_features[best_feature_index], scores
```

```
model = []
candidates = X_train.columns.tolist()
selected_features = []
r2_progress = []

while len(candidates) > 0:
    next_feature, all_scores = SelectFeature(model, candidates, X_train, y_train)
    model.append(next_feature)
    selected_features.append(next_feature)
    r2_progress.append(max(all_scores))
    candidates.remove(next_feature)
    print("Added feature:", next_feature, "| Validation R^2:", round(max(all_scores), 4))
```

```
best_index = int(np.argmax(r2_progress))
model_best = model[:best_index + 2] # +2 accounts for 'bias' and best_index offset
```

```
lr_final = LinearRegression(fit_intercept=True)
lr_final.fit(X_train[model_best], y_train)
```

```
r2_test = lr_final.score(X_test[model_best], y_test)
```

```
print("Final model test R^2:", round(r2_test, 4))
print("Selected features:", model_best)
```

```
➡ Added feature: Wing.Length | Validation R^2: 0.0918
Added feature: Beak.Length_Nares | Validation R^2: 0.0982
Added feature: Beak.Width | Validation R^2: 0.1034
Added feature: Beak.Depth | Validation R^2: 0.1088
Added feature: Sex | Validation R^2: 0.1122
Added feature: Tail.Length | Validation R^2: 0.1141
Added feature: Tarsus.Length | Validation R^2: 0.1151
Added feature: Beak.Length_Culmen | Validation R^2: 0.1158
Added feature: Hand-wing.Index | Validation R^2: 0.116
Added feature: Kipps.Distance | Validation R^2: 0.1161
Added feature: Secondary1 | Validation R^2: 0.1161
Final model test R^2: 0.1223
Selected features: ['Wing.Length', 'Beak.Length_Nares', 'Beak.Width', 'Beak.Depth', 'Sex', 'Tail.Len
```

```
X_train.head()
```

```
➡
```

	Beak.Length_Culmen	Beak.Length_Nares	Beak.Width	Beak.Depth	Tarsus.Length	Wing.Length	Kipps.Distance
32795	0.093177	-0.195732	-0.176084	-0.183358	0.270495	-0.305171	
13810	6.986371	6.247147	7.402524	5.435082	0.905493	3.013555	
87598	-0.456790	-0.395666	-0.487800	-0.420201	-0.306776	-0.602451	
60353	-0.142523	-0.125755	0.330455	0.079801	0.200398	0.192098	
8136	2.379881	2.878256	3.194351	2.198229	0.456046	0.138047	

✓ Calculate the BaseLine Regressor Error Rate

```
Baseline_Error = (1 - (df['Family3'].value_counts().max())/df.shape[0])*100
print('The Baseline Error Rate is ', Baseline_Error, '%')
```

```
➡ The Baseline Error Rate is 94.72885729156285 %
```

✓ Models

The following section has a series of models which are used to determine what the best approach to classifying the family of the birds.

```
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from time import time
```

```
X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)
y_train = y_train.astype(np.int32)
y_test = y_test.astype(np.int32)
```

```

start = time()
xgb_grid = {
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 200],
    'max_depth': [3],
    'tree_method': ['gpu_hist'],
    'predictor': ['gpu_predictor']
}

xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', verbosity=3)
xgbCV = GridSearchCV(xgb, param_grid=xgb_grid, cv=5, return_train_score=True, n_jobs=-1, verbose=3) # o
xgbCV.fit(X_train, y_train)

print('XGBoost Classifier:')
print(' Optimal Parameters:', xgbCV.best_params_)
print(' Optimal CV accuracy =', round(xgbCV.best_score_, 3))
print(f"Computer runtime: {time()-start:.2f} seconds")

eval_start = time()
xgb_best = xgbCV.best_estimator_
acc_train = xgb_best.score(X_train, y_train)
acc_test = xgb_best.score(X_test, y_test)

print('XGBoost Evaluation')
print(f"\tTrain accuracy = {round(acc_train,4)}, Test accuracy = {round(acc_test,4)}")
print(f"\tRuntime: {time()-eval_start:.2f} seconds")

```



```

-----
NameError                                Traceback (most recent call last)
<ipython-input-6-2b5c49431465> in <cell line: 0>()
      6 from time import time
      7
----> 8 X_train = X_train.astype(np.float32)
      9 X_test = X_test.astype(np.float32)
     10 y_train = y_train.astype(np.int32)

NameError: name 'X_train' is not defined

```

KNN

```

from cuml.neighbors import KNeighborsClassifier
from cuml.model_selection import train_test_split
from cuml.metrics import accuracy_score
import cudf
import numpy as np
import itertools

# Convert your data to cuDF format
X_train_gpu = cudf.DataFrame.from_pandas(X_train)
y_train_gpu = cudf.Series(y_train)
X_test_gpu = cudf.DataFrame.from_pandas(X_test)
y_test_gpu = cudf.Series(y_test)

# Define parameter grid

```

```

param_grid = {
    'n_neighbors': [1, 2, 3, 4, 5],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

# Manual grid search
best_score = 0
best_params = None
for n, w, m in itertools.product(param_grid['n_neighbors'],
                                  param_grid['weights'],
                                  param_grid['metric']):

    try:
        model = KNeighborsClassifier(n_neighbors=n, weights=w, metric=m)
        model.fit(X_train_gpu, y_train_gpu)
        preds = model.predict(X_train_gpu)
        acc = accuracy_score(y_train_gpu, preds)

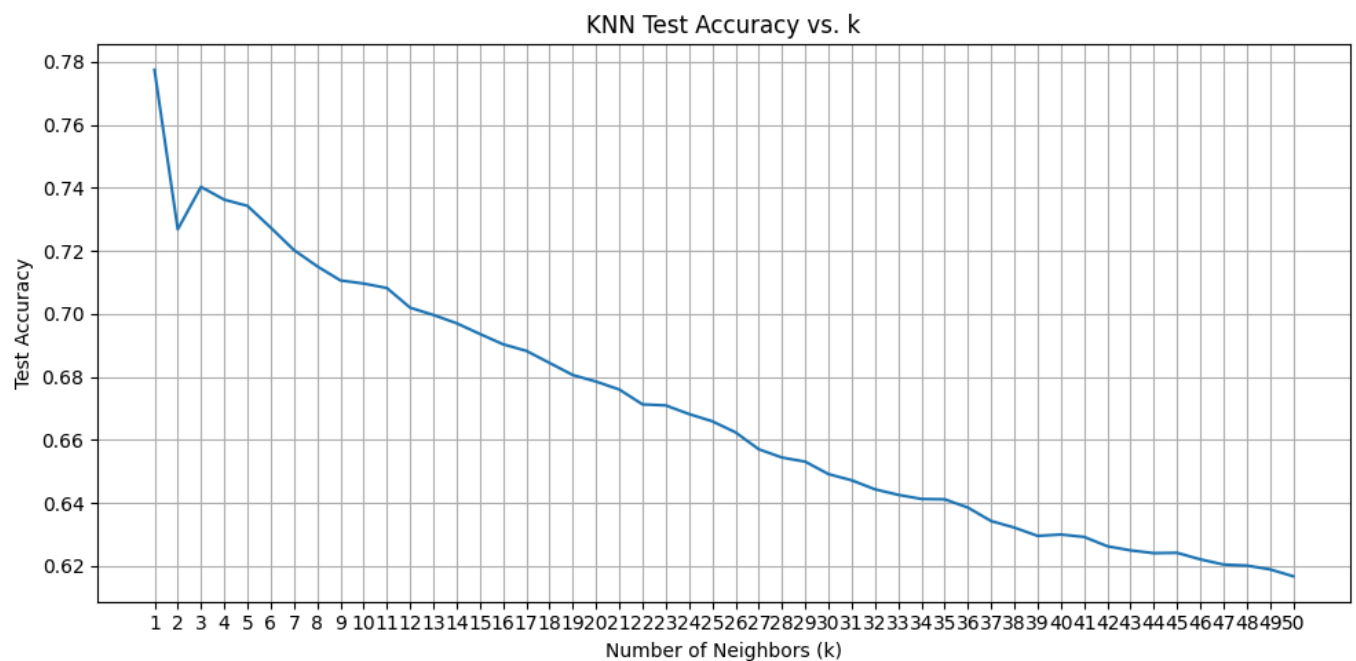
        print(f"Params: n={n}, weights={w}, metric={m} => CV Accuracy: {acc:.4f}")

        if acc > best_score:
            best_score = acc
            best_params = {'n_neighbors': n, 'weights': w, 'metric': m}
            best_model = model
    except Exception as e:
        print(f"Skipping combo n={n}, weights={w}, metric={m} due to error: {e}")

# Test set performance
y_test_pred = best_model.predict(X_test_gpu)
test_acc = accuracy_score(y_test_gpu, y_test_pred)

print("\nBest Parameters:", best_params)
print(f"Best Cross-Validation Accuracy: {best_score:.4f}")
print(f"Test Set Accuracy: {test_acc:.4f}")

```



▼ Logistic Regressor

```
from cuml.linear_model import LogisticRegression
from cuml.metrics import accuracy_score
import cudf

# Convert datasets to cuDF
X_train_gpu = cudf.DataFrame.from_pandas(X_train)
y_train_gpu = cudf.Series(y_train)
X_test_gpu = cudf.DataFrame.from_pandas(X_test)
y_test_gpu = cudf.Series(y_test)

# Initialize and fit Logistic Regression (cuML)
model1 = LogisticRegression(
    penalty='l1',
    solver='qn',
    max_iter=100,
    fit_intercept=True
)
model2 = LogisticRegression(
    penalty='l2',
    solver='qn',
    max_iter=100,
    fit_intercept=True
)

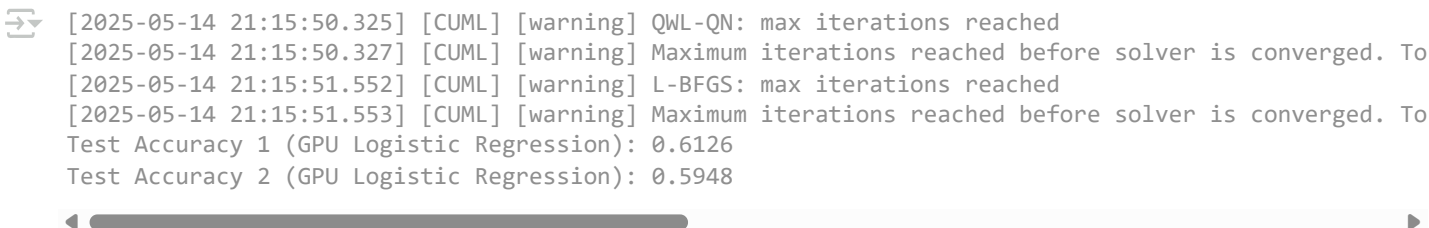
model1.fit(X_train_gpu, y_train_gpu)
model2.fit(X_train_gpu, y_train_gpu)

# Predict and evaluate
y_pred = model1.predict(X_test_gpu)
acc = accuracy_score(y_test_gpu, y_pred)

print(f"Test Accuracy 1 (GPU Logistic Regression): {acc:.4f}")

y_pred = model2.predict(X_test_gpu)
acc = accuracy_score(y_test_gpu, y_pred)

print(f"Test Accuracy 2 (GPU Logistic Regression): {acc:.4f}")
```



```
[2025-05-14 21:15:50.325] [CUML] [warning] QWL-QN: max iterations reached
[2025-05-14 21:15:50.327] [CUML] [warning] Maximum iterations reached before solver is converged. To
[2025-05-14 21:15:51.552] [CUML] [warning] L-BFGS: max iterations reached
[2025-05-14 21:15:51.553] [CUML] [warning] Maximum iterations reached before solver is converged. To
Test Accuracy 1 (GPU Logistic Regression): 0.6126
Test Accuracy 2 (GPU Logistic Regression): 0.5948
```

▼ Decision Tree

```
import xgboost as xgb
import numpy as np
import itertools
from sklearn.metrics import accuracy_score
```

```

# Prepare DMatrix format for XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define parameter grid (mimicking a decision tree)
param_grid = {
    'max_depth': [5, 10, 20, 30],
    'min_child_weight': [1, 2, 4],
    'gamma': [0, 1],
}

# Manual grid search
best_score = 0
best_params = None

for depth, child_weight, gamma in itertools.product(
    param_grid['max_depth'],
    param_grid['min_child_weight'],
    param_grid['gamma']
):
    params = {
        'tree_method': 'gpu_hist',
        'objective': 'multi:softmax',
        'num_class': len(np.unique(y_train)),
        'eval_metric': 'mlogloss',
        'max_depth': depth,
        'min_child_weight': child_weight,
        'gamma': gamma
    }

    model = xgb.train(params, dtrain, num_boost_round=1, verbose_eval=False)
    preds = model.predict(dtest)
    acc = accuracy_score(y_test, preds)

    print(f"Params: max_depth={depth}, min_child_weight={child_weight}, gamma={gamma} => Test Acc: {acc}")

    if acc > best_score:
        best_score = acc
        best_model = model
        best_params = params

# Final results
print("\nBest Parameters:", best_params)
print(f"Best Test Set Accuracy (single GPU tree): {best_score:.4f}")

```

➡ /usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:26] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)

Params: max_depth=5, min_child_weight=1, gamma=0 => Test Acc: 0.4868

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:26] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:27] WARNING: /wor


```

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:27] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
Params: max_depth=5, min_child_weight=1, gamma=1 => Test Acc: 0.4868
Params: max_depth=5, min_child_weight=2, gamma=0 => Test Acc: 0.4654
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:27] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
Params: max_depth=5, min_child_weight=2, gamma=1 => Test Acc: 0.4654
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:27] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:28] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:28] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
Params: max_depth=5, min_child_weight=4, gamma=0 => Test Acc: 0.4167
Params: max_depth=5, min_child_weight=4, gamma=1 => Test Acc: 0.4167
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:28] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)
Params: max_depth=10, min_child_weight=1, gamma=0 => Test Acc: 0.5232
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [21:12:28] WARNING: /wor

E.g. tree_method = "hist", device = "cuda"

warnings.warn(smsg, UserWarning)

```

Random Forest Classifier

```

# cross-validate to determine optimal number of estimators and depth
time_start = time()
grid = {'max_depth': [10, 15, 20], 'n_estimators': [100, 200, 300]}

rf = RandomForestClassifier(max_features = 'sqrt', n_jobs=-1) # sqrt(P) for classifier
rfCV = GridSearchCV(rf, param_grid=grid, return_train_score=True, n_jobs=-1)
rfCV.fit(X_train, y_train)

print('Random Forest Classifier:')
print(' Optimal Parameters: ', rfCV.best_params_)
print(' Optimal Valid R2 = ', (rfCV.best_score_).round(3))

time_end = time()

```

```
print(f"Computer runtime: {time_end-time_start} seconds")
```

```
time_s = time()
print('Random Forest Regressor')
rf_best = rfCV.best_estimator_
R2_train = rf_best.score(X_train,y_train)
R2_test = rf_best.score(X_test,y_test)
print(f"\ttrain R2 = {round(R2_train,4)}, test R2 = {round(R2_test,4)}")
time_e = time()
print(f"\tRuntime: {time_e-time_s} seconds")
```

```
➡ /work/cssema415/202530/03/.venv/lib/python3.11/site-packages/sklearn/model_selection/_split.py:805:
  warnings.warn(
Random Forest Classifier:
  Optimal Parameters: {'max_depth': 20, 'n_estimators': 300}
  Optimal Valid R2 = 0.809
Computer runtime: 42.81080222129822 seconds
Random Forest Regressor
  train R2 = 0.9977, test R2 = 0.822
  Runtime: 3.7895705699920654 seconds
```

```
labels = [
    'KNN', 'Logistic Regression', 'Decision Tree', 'Random Forest Classifier',
    'Gradient Boosting', 'K-means', 'SVM'
]
```

```
import matplotlib.pyplot as plt
import numpy as np
```

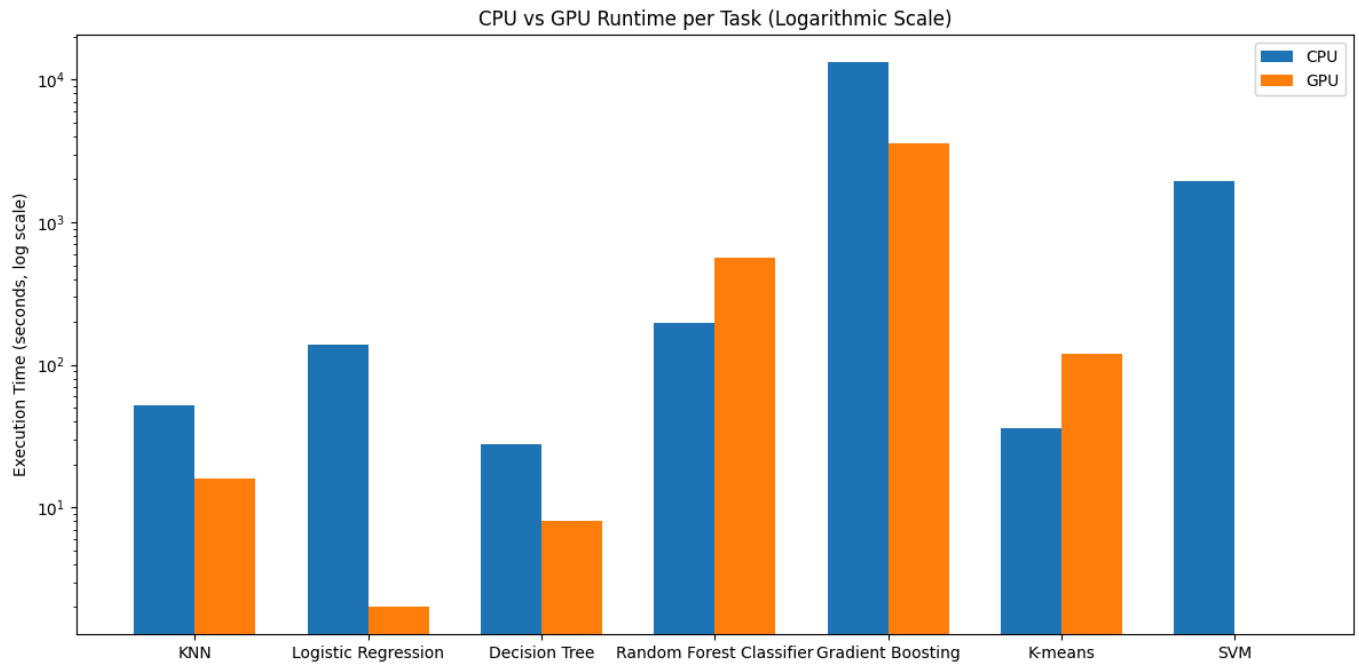
```
# Your data
cpu_performance = [52.1, 137.3, 27.8, 197.67, 13234.11, 36, 1958.73]
gpu_performance = [16, 2, 8, 560.85, 3571.21, 120, 0]
```

```
# X-axis labels
x = np.arange(len(labels))
width = 0.35 # Bar width
```

```
# Create bar chart
plt.figure(figsize=(12, 6))
plt.bar(x - width/2, cpu_performance, width, label='CPU')
plt.bar(x + width/2, gpu_performance, width, label='GPU')
```

```
plt.xticks(x, labels)
plt.yscale('log')
plt.ylabel('Execution Time (seconds, log scale)')
plt.title('CPU vs GPU Runtime per Task (Logarithmic Scale)')
plt.legend()
plt.grid(True, which='both', axis='y', linestyle='', linewidth=0.5)
```

```
plt.tight_layout()
plt.show()
```



Double-click (or enter) to edit

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import silhouette_score
import pandas as pd

pipeline = Pipeline([
    ('scale', StandardScaler()),
    ('poly', PolynomialFeatures(interaction_only=True)),
    ('kmeans', KMeans())
])

grid = {
    'poly__degree': [1],
    'kmeans__n_clusters': [2, 3, 4, 5, 6],
    'kmeans__init': ['k-means++'],
    'kmeans__n_init': [10]
}

gridCV = GridSearchCV(pipeline, grid, cv=3, n_jobs=-1, verbose=1, scoring='neg_mean_squared_error')

gridCV.fit(X_processed)

best_kmeans = gridCV.best_estimator_.named_steps['kmeans']
labels = best_kmeans.labels_
sil_score = silhouette_score(X_processed, labels)
print("Best K:", best_kmeans.n_clusters)
print("Silhouette Score:", sil_score)
```

➞ Fitting 3 folds for each of 5 candidates, totalling 15 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or
warnings.warn(
Best K: 2
Silhouette Score: 0.5753904254863558

```
from cuml.preprocessing import StandardScaler
from cuml.cluster import KMeans
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import silhouette_score
import pandas as pd
import numpy as np
import cudf

poly = PolynomialFeatures(degree=1, interaction_only=True, include_bias=False)
X_poly = poly.fit_transform(X_processed)
X_cudf = cudf.DataFrame.from_records(X_poly)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cudf)
best_score = -1
best_k = None
best_model = None

for k in [2, 3, 4, 5, 6]:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    labels_cpu = kmeans.labels_.to_numpy()
    sil_score = silhouette_score(X_poly, labels_cpu)

    print(f"K={k} => Silhouette Score: {sil_score:.4f}")
    if sil_score > best_score:
        best_score = sil_score
        best_k = k
        best_model = kmeans

print(f"\nBest number of clusters: {best_k}")
print(f"Best silhouette score: {best_score:.4f}")
```

➞ K=2 => Silhouette Score: 0.5754
K=3 => Silhouette Score: 0.4496
K=4 => Silhouette Score: 0.3327
K=5 => Silhouette Score: 0.2677
K=6 => Silhouette Score: 0.2768

Best number of clusters: 2
Best silhouette score: 0.5754

Start coding or [generate](#) with AI.



```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-2-01c5bfff5dc1> in <cell line: 0>()  
    13  
    14 # Convert to cuDF  
----> 15 X_train_gpu = cudf.DataFrame.from_pandas(pd.DataFrame(X_train))  
    16 y_train_gpu = cudf.Series(y_train)  
    17 X_val_gpu = cudf.DataFrame.from_pandas(pd.DataFrame(X_val))  
  
NameError: name 'X_train' is not defined
```



```
import lightgbm as lgb  
from sklearn.model_selection import GridSearchCV  
import time  
  
# Define parameter grid  
param_grid = {  
    'max_depth': [10, 15, 20],  
    'n_estimators': [100, 200, 300],  
    'learning_rate': [0.1],          # Optional, you can grid this too  
    'device_type': ['gpu'],          # Key: tell LightGBM to use GPU  
    'boosting_type': ['gbdt'],       # Standard gradient boosting  
    'tree_learner': ['serial'],      # 'serial' for single GPU  
    'verbosity': [-1],  
}  
  
# Initialize LightGBM model with GPU settings  
model = lgb.LGBMClassifier(  
    objective='multiclass',  
    num_class=len(np.unique(y_train)),  
    random_state=42  
)  
  
# Time the grid search  
time_start = time.time()  
grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy', n_jobs=1)  
grid_search.fit(X_train, y_train)  
time_end = time.time()  
  
# Report  
print("\nLightGBM (GPU) Results:")  
print("Best Params:", grid_search.best_params_)  
print("Best CV Accuracy:", round(grid_search.best_score_, 4))  
print("Training Time:", round(time_end - time_start, 2), "seconds")  
  
# Evaluate on test set  
best_model = grid_search.best_estimator_  
test_acc = best_model.score(X_test, y_test)  
print("Test Accuracy:", round(test_acc, 4))
```



```
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The leas  
warnings.warn(  
  
LightGBM (GPU) Results:  
Best Params: {'boosting_type': 'gbdt', 'device_type': 'gpu', 'learning_rate': 0.1, 'max_depth': 20,  
Best CV Accuracy: 0.1408
```

```
Training Time: 560.85 seconds
Test Accuracy: 0.1274
```

```
# cross-validate to determine optimal number of estimators and depth
time_start = time()
grid = {'max_depth': [10, 15, 20], 'n_estimators': [100, 200, 300]}

rf = RandomForestClassifier(max_features = 'sqrt', n_jobs=-1, random_state=42) # sqrt(P) for classifier
rfCV = GridSearchCV(rf, param_grid=grid, return_train_score=True, n_jobs=-1)
rfCV.fit(X_train, y_train)

print('Random Forest Classifier:')
print(' Optimal Parameters: ', rfCV.best_params_)
print(' Optimal Valid R2 = ', (rfCV.best_score_).round(3))

time_end = time()
print(f"Computer runtime: {time_end-time_start} seconds")

time_s = time()
print('Random Forest Regressor')
rf_best = rfCV.best_estimator_
R2_train = rf_best.score(X_train, y_train)
R2_test = rf_best.score(X_test, y_test)
print(f"\ttrain R2 = {round(R2_train, 4)}, test R2 = {round(R2_test, 4)}")
time_e = time()
print(f"\tRuntime: {time_e-time_s} seconds")
```

```
➦ /usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The leas
  warnings.warn(
Random Forest Classifier:
  Optimal Parameters: {'max_depth': 20, 'n_estimators': 300}
  Optimal Valid R2 = 0.809
Computer runtime: 193.14838981628418 seconds
Random Forest Regressor
  train R2 = 0.9971, test R2 = 0.8208
  Runtime: 4.518910646438599 seconds
```

```
import cudf
import numpy as np
from cuml.svm import SVC
from cuml.metrics import accuracy_score
from itertools import product
from time import time

# Convert your data to GPU-compatible format
X_train_gpu = cudf.DataFrame(X_train.astype(np.float32))
y_train_gpu = cudf.Series(y_train)
X_test_gpu = cudf.DataFrame(X_test.astype(np.float32))
y_test_gpu = cudf.Series(y_test)

# Define SVM parameter grid
svm_grid = {
    'C': [0.1, 1],
    'kernel': ['linear', 'rbf']
}
```

```

# Manual grid search
start = time()
best_acc = 0
best_params = None
best_model = None

for C, kernel in product(svm_grid['C'], svm_grid['kernel']):
    model = SVC(C=C, kernel=kernel)
    model.fit(X_train_gpu, y_train_gpu)
    acc = accuracy_score(y_train_gpu, model.predict(X_train_gpu))
    print(f"Params: C={C}, kernel={kernel} => Train Accuracy: {acc:.4f}")

    if acc > best_acc:
        best_acc = acc
        best_params = {'C': C, 'kernel': kernel}
        best_model = model

print(f"\nBest Parameters: {best_params}")
print(f"Best Train Accuracy: {best_acc:.4f}")
print(f"Grid Search Runtime: {time() - start:.2f} seconds")

# Evaluate on test set
eval_start = time()
acc_train = accuracy_score(y_train_gpu, best_model.predict(X_train_gpu))
acc_test = accuracy_score(y_test_gpu, best_model.predict(X_test_gpu))
print(f"\nSVM Evaluation:")
print(f"\tTrain Accuracy: {round(acc_train, 4)}, Test Accuracy: {round(acc_test, 4)}")
print(f"\tRuntime: {time() - eval_start:.2f} seconds")

```



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
[2025-05-14 22:07:36.011] [CUML] [warning] Warning: could not fill working set, found only 517 ele
[2025-05-14 22:07:36.012] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:36.012] [CUML] [warning] Warning: could not fill working set, found only 772 ele
[2025-05-14 22:07:36.013] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:36.076] [CUML] [warning] Warning: could not fill working set, found only 517 ele
[2025-05-14 22:07:36.077] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:36.077] [CUML] [warning] Warning: could not fill working set, found only 772 ele
[2025-05-14 22:07:36.176] [CUML] [warning] Warning: could not fill working set, found only 517 ele
[2025-05-14 22:07:36.177] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:36.336] [CUML] [warning] Warning: could not fill working set, found only 517 ele
[2025-05-14 22:07:36.337] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:36.337] [CUML] [warning] Warning: could not fill working set, found only 772 ele
[2025-05-14 22:07:37.278] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:37.279] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:37.280] [CUML] [warning] Warning: could not fill working set, found only 796 ele
[2025-05-14 22:07:37.280] [CUML] [warning] Warning: could not fill working set, found only 771 ele
[2025-05-14 22:07:37.281] [CUML] [warning] Warning: could not fill working set, found only 795 ele
[2025-05-14 22:07:37.840] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:37.841] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:37.842] [CUML] [warning] Warning: could not fill working set, found only 797 ele
[2025-05-14 22:07:37.842] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:37.843] [CUML] [warning] Warning: could not fill working set, found only 797 ele
[2025-05-14 22:07:39.189] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:39.190] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:39.191] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:39.191] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:40.572] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:40.573] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:40.574] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:40.574] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:41.574] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:41.575] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:41.575] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:41.638] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:41.639] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:41.639] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:41.736] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:41.737] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:41.737] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:41.738] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:41.899] [CUML] [warning] Warning: could not fill working set, found only 545 ele
[2025-05-14 22:07:41.900] [CUML] [warning] Warning: could not fill working set, found only 768 ele
[2025-05-14 22:07:41.901] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:41.901] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:41.902] [CUML] [warning] Warning: could not fill working set, found only 798 ele
[2025-05-14 22:07:42.823] [CUML] [warning] Warning: could not fill working set, found only 519 ele
[2025-05-14 22:07:42.824] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:42.825] [CUML] [warning] Warning: could not fill working set, found only 771 ele
[2025-05-14 22:07:42.825] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:43.430] [CUML] [warning] Warning: could not fill working set, found only 519 ele
[2025-05-14 22:07:43.431] [CUML] [warning] Warning: could not fill working set, found only 769 ele
[2025-05-14 22:07:43.432] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:43.432] [CUML] [warning] Warning: could not fill working set, found only 771 ele
[2025-05-14 22:07:44.850] [CUML] [warning] Warning: could not fill working set, found only 519 ele
[2025-05-14 22:07:44.851] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:44.852] [CUML] [warning] Warning: could not fill working set, found only 770 ele
[2025-05-14 22:07:44.852] [CUML] [warning] Warning: could not fill working set, found only 772 ele
```

MemoryError Traceback (most recent call last)

```
<ipython-input-9-8ca384fe9a0a> in <cell line: 0>()
    26 for C, kernel in product(svm_grid['C'], svm_grid['kernel']):
    27     model = SVC(C=C, kernel=kernel)
----> 28     model.fit(X_train_gpu, y_train_gpu)
    29     acc = accuracy_score(y_train_gpu, model.predict(X_train_gpu))
```

```
30     print(f"Params: C={C}, kernel={kernel} => Train Accuracy: {acc:.4f}")
```

^ 12 frames

```
svc.pyx in cuml.svm.svc.SVC.fit()
```

```
svc.pyx in cuml.svm.svc.SVC._fit_multiclass()
```

```
/usr/local/lib/python3.11/dist-packages/cuml/internals/api_decorators.py in wrapper(*args,  
**kwargs)
```

```
191
```

```
192
```

```
    if process_return:
```

```
--> 193
```

```
        ret = func(*args, **kwargs)
```

```
194
```

```
    else:
```

```
195
```

```
        return func(*args, **kwargs)
```

```
svc.pyx in cuml.svm.svc.SVC.fit()
```

```
svc.pyx in cuml.svm.svc.SVC.fit()
```

MemoryError: std::bad_alloc: out_of_memory: CUDA error at: /tmp/pip-build-env-

c7imy8iv/normal/lib/python3.11/site-

packages/librmm/include/rmm/mr/device/cuda_memory_resource.hpp:62: cudaErrorMemoryAllocation out

```

from cuml.neighbors import KNeighborsClassifier
from cuml.model_selection import train_test_split
from cuml.metrics import accuracy_score
import cudf
import numpy as np
import itertools

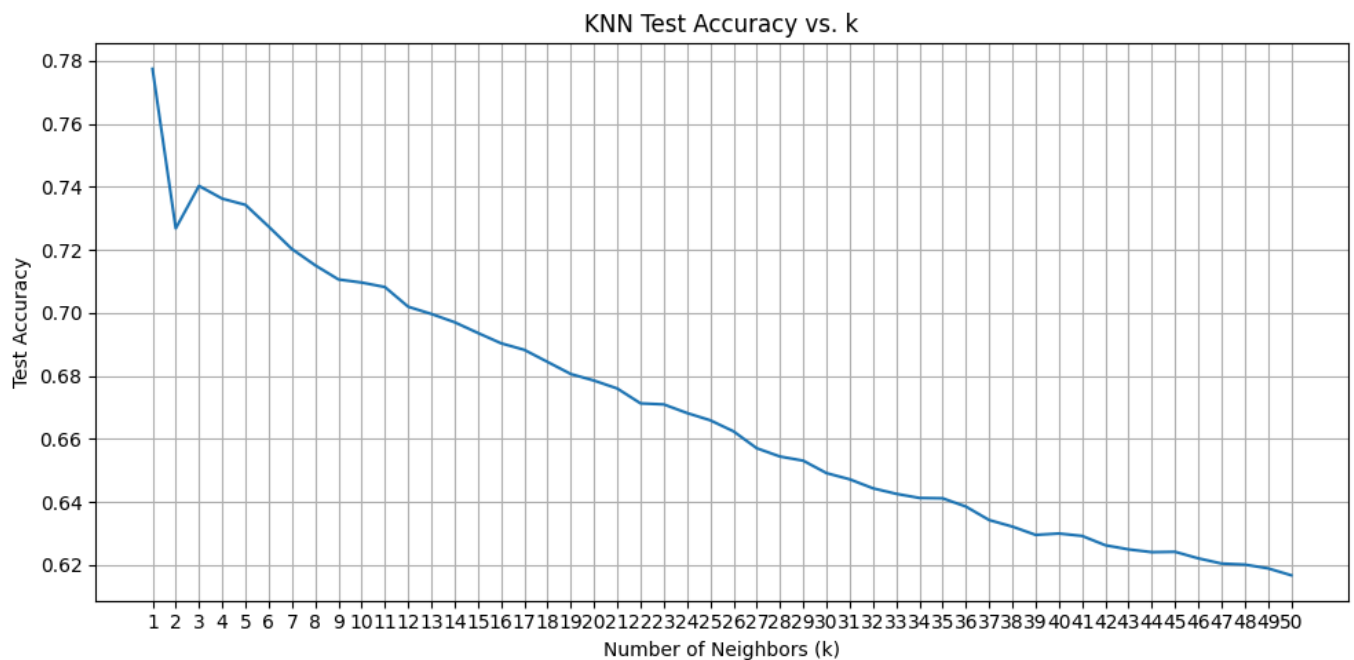
X_train_gpu = cudf.DataFrame.from_pandas(X_train)
y_train_gpu = cudf.Series(y_train)
X_test_gpu = cudf.DataFrame.from_pandas(X_test)
y_test_gpu = cudf.Series(y_test)

k_values = list(range(1, 51))
accuracies = []

for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k, weights='uniform', metric='euclidean')
    model.fit(X_train_gpu, y_train_gpu)
    preds = model.predict(X_test_gpu)
    acc = accuracy_score(y_test_gpu, preds)
    accuracies.append(acc)

plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracies, marker='')
plt.title("KNN Test Accuracy vs. k")
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Test Accuracy")
plt.grid(True)
plt.xticks(k_values)
plt.tight_layout()
plt.show()

```




```

import cudf
import numpy as np
import matplotlib.pyplot as plt
from cuml.linear_model import LogisticRegression
from cuml.metrics import accuracy_score
from itertools import product

# Convert to cuDF
X_train_gpu = cudf.DataFrame.from_pandas(X_train.astype('float32'))
y_train_gpu = cudf.Series(y_train)
X_test_gpu = cudf.DataFrame.from_pandas(X_test.astype('float32'))
y_test_gpu = cudf.Series(y_test)

# Define hyperparameter grid
penalties = ['l1', 'l2']
max_iters = [100, 200, 500, 1000, 2000]
results = []

# Manual grid search
for penalty, max_iter in product(penalties, max_iters):
    try:
        model = LogisticRegression(
            penalty=penalty,
            solver='qn', # GPU-optimized quasi-Newton solver
            max_iter=max_iter,
            fit_intercept=True
        )
        model.fit(X_train_gpu, y_train_gpu)
        y_pred = model.predict(X_test_gpu)
        acc = accuracy_score(y_test_gpu, y_pred)
        results.append({'penalty': penalty, 'max_iter': max_iter, 'accuracy': acc})
        print(f"penalty={penalty}, max_iter={max_iter} → Accuracy: {acc:.4f}")
    except Exception as e:
        print(f"Failed for penalty={penalty}, max_iter={max_iter}: {e}")

# Convert to DataFrame for plotting
import pandas as pd
results_df = pd.DataFrame(results)

# Plot
fig, ax = plt.subplots(figsize=(8, 5))
for penalty in penalties:
    subset = results_df[results_df['penalty'] == penalty]
    ax.plot(subset['max_iter'], subset['accuracy'], marker='o', label=f'penalty={penalty}')

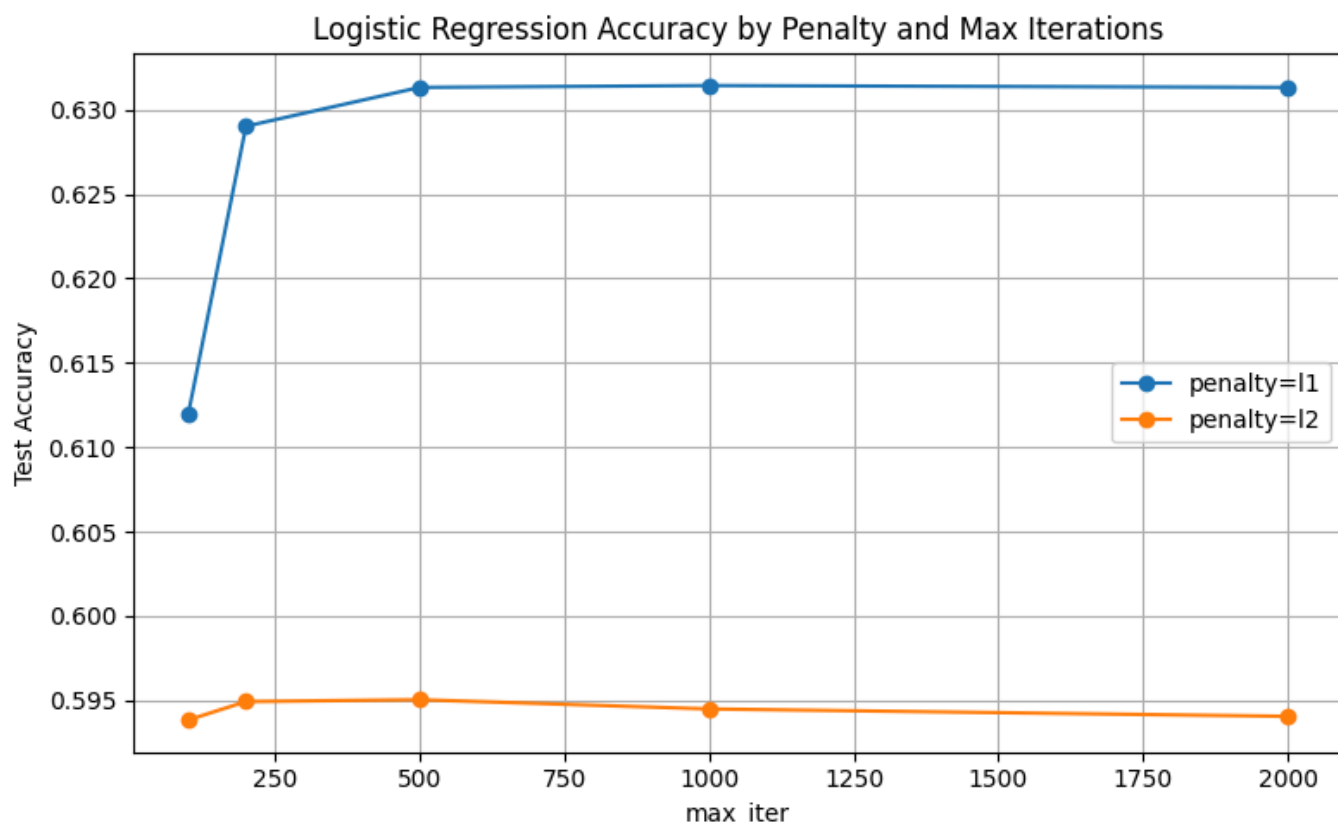
ax.set_title("Logistic Regression Accuracy by Penalty and Max Iterations")
ax.set_xlabel("max_iter")
ax.set_ylabel("Test Accuracy")
ax.legend()
ax.grid(True)
plt.tight_layout()
plt.show()

```

```

[2025-05-14 22:30:17.847] [CUMML] [warning] QWL-QN: max iterations reached
[2025-05-14 22:30:17.848] [CUMML] [warning] Maximum iterations reached before solver is converged. To
penalty=l1, max_iter=100 → Accuracy: 0.6119
[2025-05-14 22:30:18.006] [CUMML] [warning] QWL-QN: max iterations reached
[2025-05-14 22:30:18.006] [CUMML] [warning] Maximum iterations reached before solver is converged. To
penalty=l1, max_iter=200 → Accuracy: 0.6290
penalty=l1, max_iter=500 → Accuracy: 0.6313
penalty=l1, max_iter=1000 → Accuracy: 0.6314
penalty=l1, max_iter=2000 → Accuracy: 0.6313
[2025-05-14 22:30:19.130] [CUMML] [warning] L-BFGS: max iterations reached
[2025-05-14 22:30:19.131] [CUMML] [warning] Maximum iterations reached before solver is converged. To
penalty=l2, max_iter=100 → Accuracy: 0.5938
penalty=l2, max_iter=200 → Accuracy: 0.5949
penalty=l2, max_iter=500 → Accuracy: 0.5950
penalty=l2, max_iter=1000 → Accuracy: 0.5945
penalty=l2, max_iter=2000 → Accuracy: 0.5941

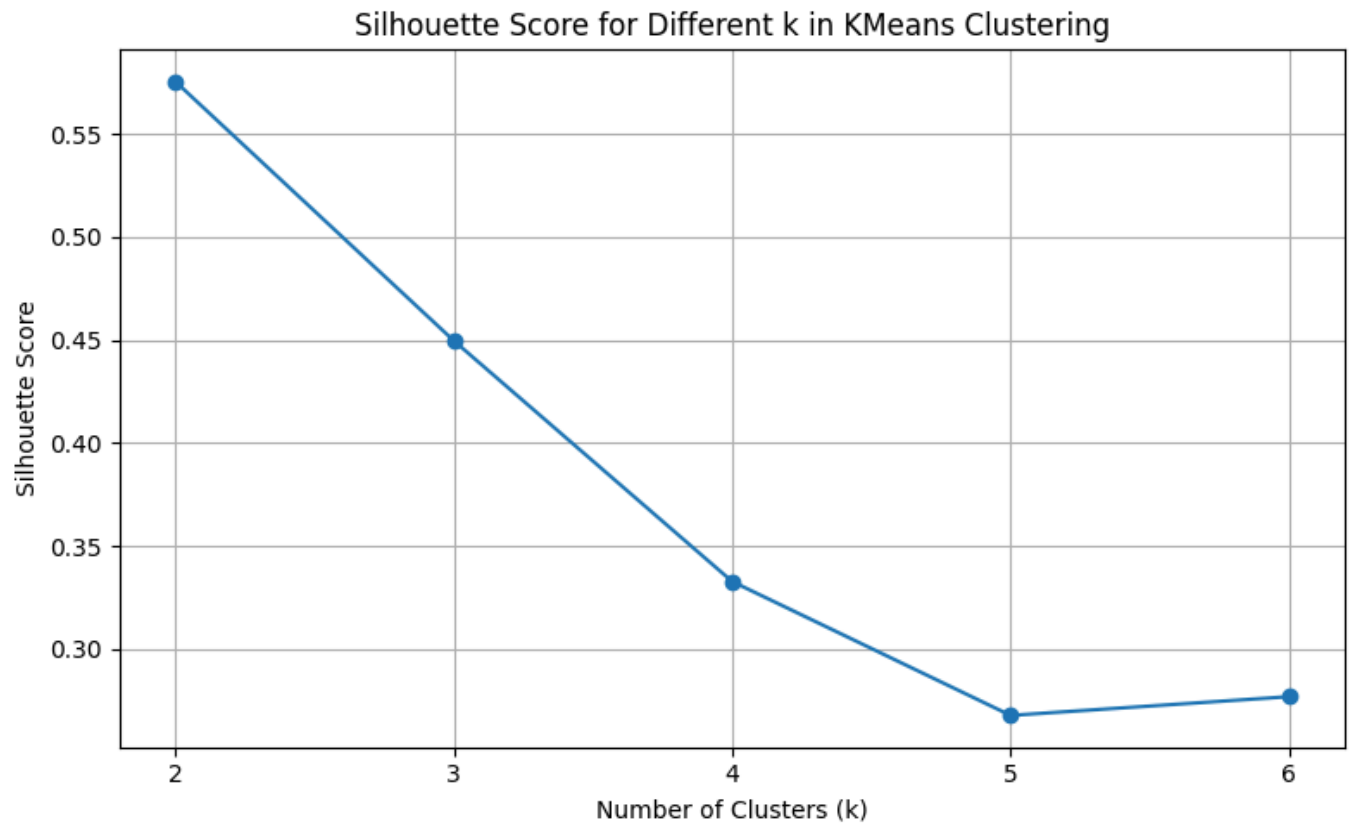
```



```

k_values = [2, 3, 4, 5, 6]
silhouette_scores = [0.5754, 0.4496, 0.3327, 0.2677, 0.2768]
plt.figure(figsize=(8, 5))
plt.plot(k_values, silhouette_scores, marker='o')
plt.xticks(k_values)
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score for Different k in KMeans Clustering")
plt.grid(True)
plt.tight_layout()
plt.show()

```

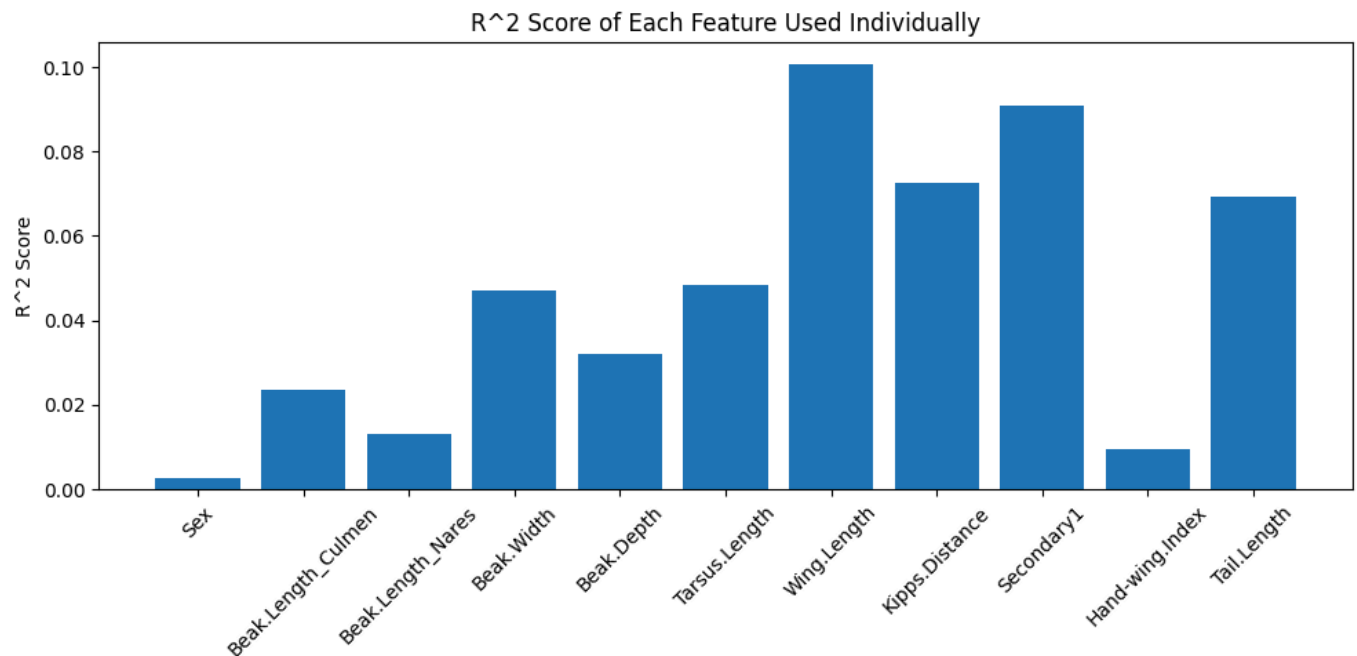


```
feature_r2_scores = {}
```

```
for feature in X.columns:
    model = LinearRegression()
    model.fit(X_train[[feature]], y_train)
    y_pred = model.predict(X_test[[feature]])
    r2 = r2_score(y_test, y_pred)
    feature_r2_scores[feature] = r2
    print(f"{feature}: R^2 = {r2:.4f}")
```

```
plt.figure(figsize=(10, 5))
plt.bar(feature_r2_scores.keys(), feature_r2_scores.values())
plt.ylabel("R^2 Score")
plt.title("R^2 Score of Each Feature Used Individually")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Sex: $R^2 = 0.0027$
Beak.Length_Culmen: $R^2 = 0.0237$
Beak.Length_Nares: $R^2 = 0.0132$
Beak.Width: $R^2 = 0.0471$
Beak.Depth: $R^2 = 0.0322$
Tarsus.Length: $R^2 = 0.0483$
Wing.Length: $R^2 = 0.1007$
Kipps.Distance: $R^2 = 0.0727$
Secondary1: $R^2 = 0.0909$
Hand-wing.Index: $R^2 = 0.0096$
Tail.Length: $R^2 = 0.0691$



```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

coefs = []
ci_lowers = []
ci_uppers = []

for feature in X.columns:
    X_feature_train = sm.add_constant(X_train[[feature]])
    model = sm.OLS(y_train, X_feature_train).fit()

    coef = model.params[feature]
    ci = model.conf_int(alpha=0.05).loc[feature].tolist()

    coefs.append(coef)
    ci_lowers.append(ci[0])
    ci_uppers.append(ci[1])
```

```
results_df = pd.DataFrame({  
    'Feature': X.columns,  
    'Coef': coef
```