FACIAL RECOGNITION WITH SUPERVISED LEARNING

TABLE OF CONTENT

GOAL OF THIS PROJECT

- Deployed AI solution that accurately distinguish between images of notable personalities and the general populace, enhancing the personal security of such high-profile individuals.
- Focus on Arnold Schwarzenegger

PREPARE THE DATA

	0	1	2	3	4	5	6	Label
1	-2.0619872	0.5813201	-0.24911457	-0.6313401	-1.3598989	0.7516187	-0.02936425	1
2	-0.7968382	-0.66722834	-0.10788881	0.01975525	-0.68634826	0.9127795	0.46341223	1
3	5.3767786	1.1426954	2.5431106	-2.7272117	0.2727849	-0.9721871	1.1112208	1
4	7.029235	1.2428827	-2.628079	1.2244786	-1.1413696	-1.6206465	0.20589042	1
5	5.4848223	6.752706	-4.2911143	1.7404125	-1.6030868	-1.0751754	1.919936	1
6	-0.13898633	1.428951	-1.6003897	-0.06967798	2.0314894	0.20314787	1.1879293	1
7	-4.9443116	-5.249127	-0.014346995	-1.5875958	0.59518504	0.9182035	2.6764379	1
8	0.26447877	-0.6295751	2.9667895	0.20474683	0.035296015	1.3748426	-0.59298164	1
9	4.611807	-0.03589843	2.016866	-3.6675699	-1.2059602	-0.6201473	0.21297784	1
10	-0.7734096	1.0132053	3.0378122	-1.260474	-0.29910663	-2.4563105	-0.91009295	1
11	10.82687	5.471469	-1.9767497	-0.84736377	-0.29428092	1.6656278	0.16030522	1
12	-1.8959919	2.5897965	1.3965319	-3.2357833	0.7579947	-2.419011	-0.4328277	1
13	-2.7404678	1.2956209	0.7089457	-1.1112196	0.3601952	-1.0074039	0.84558564	1
14	-1.2059479	-2.3571577	4.7283483	-0.3607564	2.0740452	1.283695	-0.4369507	1
15	8.030009	1.6160346	6.4060497	-2.2581217	3.1826873	-1.8593179	3.7265582	1
16	1.6375139	5.4582734	3.1399322	-0.54387957	-0.7474723	-1.7822986	-1.1429108	1
17	-3.1716495	0.033918735	-4.2142415	1.6626971	2.7815335	0.8853233	2.0526147	1
18	3.746497	4.176198	-1.7820765	1.2646477	0.5657319	-1.3620384	1.5360591	1
19	-1.6452969	2.8804858	1.1876053	-3.8507206	1.6120536	0.60124975	-1.6142217	1
20	1.6608331	-2.902263	-5.4775248	-0.7892586	2.7927403	-2.327529	1.8659756	1
21	-3.53302	0.85868263	1.1850765	-2.3611095	-0.7843416	1.2274028	-1.9420676	1
22	-0.8072791	1.2979736	2.014958	-0.4020976	1.2676188	-1.9950204	-1.3882446	1
23	-4.8373647	0.5816817	-2.1743698	1.1480961	3.1039023	-1.9545404	-2.2339172	1
24	2.3714836	6.033799	2.0177622	-1.3971252	0.42755648	0.87853944	-1.2169197	1
25	2.5795686	4.111673	-2.5892484	-0.32674932	-1.2560158	0.95596	1.1589862	1
26	4.669867	2.6865888	-1.3185872	1.7392284	-1.2075799	1.2861456	-0.011642457	1
27	4.31588	1.6816732	1.1290458	-2.5151587	2.171758	-2.3711212	-0.9625882	1
								1

Shape: (190,150)

Principal components from PCA, capturing key image features.

PREPARE THE DATA

- Upload the dataset
- Seperate the features and target variable label

```
df = pd.read_csv("df.csv")
```

```
# Seperate the features and target variable label
X = df.drop('Label', axis=1)
y = df['Label']
```

- Split the data into training and testing sets using stratify to balance the class (proportionally equal between training data and testing data)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21, stratify=y)
```

CHOOSE THE MODEL

- Dataset: small dataset: simpler model, faster training time
- Classification Model:
  - Logistic Regression (output: probability: p < .5 => 1; p > .5 =>0)
  - KNeighborsClassifier: predict the label of data point by taking looking at the k closest labeled data point and using majority vote
  - DecisionTreeClassifier: model complex, non-linear relationships in the image data

```python
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

TRAIN THE MODEL

- Store initialized models in a dictionary
- This approach allows for easy expansion and comparison of different models

```
models = {"LogisticRegression": LogisticRegression(),
          "KNeighborsClassifier": KNeighborsClassifier(),
          "DecisionTreeClassifier": DecisionTreeClassifier()}
```

TRAIN THE MODEL

- Store the models parameters in a dictionary
- Parameters are tailored to each model to explore a range of option during Grid Search

```
param_grid = {"LogisticRegression": {"LogisticRegression__C": [0.01, 0.1, 1, 10]},
              "KNeighborsClassifier": {"KNeighborsClassifier__n_neighbors": range(1,10)},
              "DecisionTreeClassifier": {"DecisionTreeClassifier__max_depth": [2, 5, 10],
           "DecisionTreeClassifier__min_samples_split": [2, 5, 10, 20],
           "DecisionTreeClassifier__random_state": [42]}}
```

TRAIN THE MODEL

Logistic Regression:

- C is the regularization parameter for Logistic Regression, controlling the model's complexity
- Smaller values (e.g., 0.01) indicate stronger regularization, while larger values (e.g., 10) imply weaker regularization.
- The GridSearchCV process will test the model with each value of C to find the one that best balances accuracy and complexity.

```
param_grid = {"LogisticRegression": {"LogisticRegression__C": [0.01, 0.1, 1, 10]},
              "KNeighborsClassifier": {"KNeighborsClassifier__n_neighbors": range(1,10)},
              "DecisionTreeClassifier": {"DecisionTreeClassifier__max_depth": [2, 5, 10],
           "DecisionTreeClassifier__min_samples_split": [2, 5, 10, 20],
           "DecisionTreeClassifier__random_state": [42]}}
```

TRAIN THE MODEL

KNeighbors Classifier

- n_neighbors specifies the number of nearest neighbors used in K-Nearest Neighbors.

```
param_grid = {"LogisticRegression": {"LogisticRegression__C": [0.01, 0.1, 1, 10]},
              "KNeighborsClassifier": {"KNeighborsClassifier__n_neighbors": range(1,10)},
              "DecisionTreeClassifier": {"DecisionTreeClassifier__max_depth": [2, 5, 10],
           "DecisionTreeClassifier__min_samples_split": [2, 5, 10, 20],
           "DecisionTreeClassifier__random_state": [42]}}
```

TRAIN THE MODEL

Decision Tree Classifier

- max_depth: controls the maximum depth of the Decision Tree, limiting how deep the tree can go. Limiting depth can help avoid overfitting.
- min_samples_split: specifies the minimum number of samples required to split an internal node. This hyperparameter affects the tree's branching structure.
- random_state: sets a seed for reproducibility.

```
param_grid = {"LogisticRegression": {"LogisticRegression__C": [0.01, 0.1, 1, 10]},
              "KNeighborsClassifier": {"KNeighborsClassifier__n_neighbors": range(1,10)},
              "DecisionTreeClassifier": {"DecisionTreeClassifier__max_depth": [2, 5, 10],
           "DecisionTreeClassifier__min_samples_split": [2, 5, 10, 20],
           "DecisionTreeClassifier__random_state": [42]}}
```

EVALUATE THE MODEL

- Access the performance of model and optimize hyperparameters by using cross-validation
- It helps ensure that the model's performance is reliable and not dependent on just a single split of the data

```
# Define cross-validation parameters
# KFold is used here to ensure that our model generalizes well on unseen data
kf = KFold(n_splits=5, random_state=42, shuffle=True)
```

EVALUATE THE MODEL

- max_depth: controls the maximum depth of the Decision Tree, limiting how deep the tree can go. Limiting depth can help avoid overfitting.
- min_samples_split: specifies the minimum number of samples required to split an internal node. This hyperparameter affects the tree's branching structure.
- random_state: sets a seed for reproducibility.

```
param_grid = {"LogisticRegression": {"LogisticRegression__C": [0.01, 0.1, 1, 10]},
              "KNeighborsClassifier": {"KNeighborsClassifier__n_neighbors": range(1,10)},
              "DecisionTreeClassifier": {"DecisionTreeClassifier__max_depth": [2, 5, 10],
          "DecisionTreeClassifier__min_samples_split": [2, 5, 10, 20],
          "DecisionTreeClassifier__random_state": [42]}}
```

FINE TUNING THE MODEL

- Prepare to collect Grid Search CV results
- Grid Search helps find the best parameter combination for each model

```
pipe_accuracies = {}
pipe_params = {}
pipelines = {}
```

FINE TUNING THE MODEL

- Create separate pipelines for each model, loop through the models and perform GridSearchCV
- Grid Search helps find the best parameter combination for each model
- Pipelines integrate preprocessing (e.g., scaling) with the model for cleaner code and to prevent data leakage

```
for name, model in models.items():
    pipeline = Pipeline(steps=[
        ("scaler", StandardScaler()),
        (name, model)
    ])
    # Create the GridSearchCV object
    grid_search = GridSearchCV(pipeline, param_grid[name], cv=kf, scoring="accuracy")

    # Perform grid search and fit the model and store the results
    grid_search.fit(X_train, y_train)
    pipe_accuracies[name] = grid_search.best_score_
    pipe_params[name] = grid_search.best_params_
    pipelines[name] = grid_search
```

FINE TUNING THE MODEL

- Select the best model based on the best cross-validation score

```
best_model_name = max(pipe_accuracies)
best_model_cv_score = max(pipe_accuracies.values())
best_model_info = pipe_params[best_model_name]
```

FINE TUNING THE MODEL

- Print the best model name, parameters, and CV score

```
# Select the best model based on the best cross-validation score
best_model_name = max(pipe_accuracies)
best_model_cv_score = max(pipe_accuracies.values())
best_model_info = pipe_params[best_model_name]
```

```
# Print the best model name, parameters, and CV score
print(f"Best Model: {best_model_name}")
print(f"Best Model Parameters: {best_model_info}")
print(f"Best Model CV Score: {best_model_cv_score}")
```

```
Best Model: LogisticRegression
Best Model Parameters: {'LogisticRegression__C': 1}
Best Model CV Score: 0.8288172043010752
```

EVALUATE THE MODEL

- Evaluate the model using metrics: help us understand the model's effectiveness

```
y_pred = pipelines[best_model_name].predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.8158
Precision: 1.0000
Recall: 0.1250
F1 Score: 0.2222
```

THANK YOU

FOR YOUR NICE ATTENTION