Classification Notebook

```
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    balanced_accuracy_score,
    classification_report,
    confusion_matrix,
)
```

```
# Paths to the training and test data (adjust if needed)
train_path = "Data/gsb-544-fall-2025-classification/CAH-201803-train.csv"
test_path  = "Data/gsb-544-fall-2025-classification/CAH-201803-test.csv"

train = pd.read_csv(train_path)
test  = pd.read_csv(test_path)

display(train.head())
display(test.head())
```

	id_num	Q1	Q2	political_affiliation	Q4	Q5	Q6	Q7	Q8	Q9
0	1	Male	53	Independent	Liberal	College degree	Black	No	No	No
1	5	Female	66	Independent	Conservative	Some college	White	Yes	No	Yes
2	7	Female	58	Democrat	Liberal	College degree	White	No	No	No
3	8	Male	55	Independent	Moderate	High school or less	White	Yes	Yes	Yes
4	9	Male	64	Republican	Conservative	High school or less	White	Yes	Yes	Yes

	id_num	Q1	Q2	Q4	Q5	Q6	Q7	Q8	Q9	Q10
0	2	Female	78	Conservative	College degree	White	Yes	Yes	No	Yes, very religious
1	3	Male	59	Moderate	High school or less	Black	Yes	Yes	Yes	Yes, very religious
2	4	Male	59	Moderate	High school or less	White	Yes	No	Yes	Yes, very religious
3	6	Male	52	Moderate	Graduate degree	White	Yes	Yes	Yes	Yes, somewhat reli
4	11	Female	33	Moderate	High school or less	White	No	No	Yes	Yes, somewhat reli

```
display(train.describe())
display(test.describe())
```

	id_num	Q2	Q15	Q16	Q17
count	169.000000	169.000000	169.000000	169.000000	169.000000
mean	166.786982	47.508876	4.284024	2.970414	3.497041
std	97.770210	16.057233	1.277942	1.712788	1.484540
min	1.000000	18.000000	1.000000	1.000000	1.000000
25%	82.000000	35.000000	4.000000	1.000000	2.000000
50%	162.000000	50.000000	5.000000	2.000000	4.000000
75%	248.000000	60.000000	5.000000	5.000000	5.000000
max	335.000000	79.000000	5.000000	5.000000	5.000000

	id_num	Q2	Q15	Q16	Q17
count	166.000000	166.000000	166.000000	166.000000	166.000000
mean	169.234940	48.240964	4.409639	2.993976	3.373494
std	96.184918	15.775002	1.056422	1.707371	1.649322
min	2.000000	18.000000	1.000000	1.000000	1.000000
25%	91.000000	37.000000	4.000000	1.000000	2.000000

	id_num	Q2	Q15	Q16	Q17
50%	171.500000	49.000000	5.000000	2.000000	4.000000
75%	253.750000	60.000000	5.000000	5.000000	5.000000
max	334.000000	92.000000	5.000000	5.000000	5.000000

```
display(train.nunique())
display(test.nunique())
```

```
id_num                   169
Q1                         2
Q2                        56
political_affiliation      3
Q4                         3
Q5                         4
Q6                         4
Q7                         2
Q8                         2
Q9                         2
Q10                        3
Q11                        2
Q12                        2
Q13                        2
Q14                        3
Q15                        4
Q16                        4
Q17                        4
Q18                        2
dtype: int64
```

```
id_num    166
Q1          2
Q2         58
Q4          3
Q5          4
Q6          4
Q7          2
Q8          2
Q9          2
Q10         3
Q11         2
```

```
Q12         2
Q13         2
Q14         3
Q15         4
Q16         4
Q17         4
Q18         2
dtype: int64
```

```
X = train.drop(columns=["id_num", "political_affiliation"])
y = train["political_affiliation"]
X_test_full = test.drop(columns=["id_num"])

print("Train shape:", X.shape)
print("Test shape:", X_test_full.shape)
```

```
Train shape: (169, 17)
Test shape: (166, 17)
```

```
# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Numeric columns:", numeric_features)
print("Categorical columns:", categorical_features)

# OneHotEncoder signature differs slightly across sklearn versions
try:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
except TypeError:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

# Numeric and categorical transformers
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot",  ohe),
])
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

```
Numeric columns: ['Q2', 'Q15', 'Q16', 'Q17']
Categorical columns: ['Q1', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
```

```
# Define CV and scoring
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Pipelines for each model
logreg_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LogisticRegression(max_iter=100000,
                               multi_class="multinomial",
                               solver="saga")),
])

lda_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LinearDiscriminantAnalysis()),
])

knn_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", KNeighborsClassifier()),
])

linear_svm_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LinearSVC()),
])

# Optional extra model: RBF SVM
rbf_svm_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", SVC(kernel="rbf")),
])
```

```python
tree_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", DecisionTreeClassifier(random_state=42)),
])

scoring = {
    "accuracy": "accuracy",
    "f1_macro": "f1_macro",
    "balanced_accuracy": "balanced_accuracy",
}

# Hyperparameter grids
logreg_grid = {
    "clf__penalty": ["l1", "l2"],
    "clf__C": [0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10],
    "clf__class_weight": [None, "balanced"],
}

lda_grid = {}

knn_grid = {
    "clf__n_neighbors": [3, 5, 7, 9, 11, 13, 15],
    "clf__weights": ["uniform", "distance"],
    "clf__p": [1, 2],
}

linear_svm_grid = {
    "clf__C": [0.01, 0.1, 1, 10],
    "clf__class_weight": [None, "balanced"],
}

rbf_svm_grid = {
    "clf__C": [0.1, 1, 5, 10],
    "clf__gamma": ["scale", "auto", 0.01, 0.1],
    "clf__class_weight": [None, "balanced"],
}

tree_grid = {
    "clf__max_depth": [None, 3, 5, 7],
    "clf__min_samples_leaf": [1, 2, 4],
    "clf__min_samples_split": [2, 5, 10],
}
```

```python
models = {
    "LogisticRegression": (logreg_pipe, logreg_grid),
    "LDA":                (lda_pipe, lda_grid),
    "kNN":                (knn_pipe, knn_grid),
    "LinearSVM":          (linear_svm_pipe, linear_svm_grid),
    "DecisionTree":       (tree_pipe, tree_grid),
    "RBFSVM (extra)":     (rbf_svm_pipe, rbf_svm_grid),
}
```

```python
# AI generated for better formating of outputs

best_model_name = None
best_model = None
best_cv_acc = -np.inf
summary_rows = []

for name, (pipe, param_grid) in models.items():
    print("=" * 80)
    print(f"Fitting model: {name}")

    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grid,
        cv=cv,
        scoring=scoring,
        refit="accuracy",
        n_jobs=-1,
        verbose=0,
    )

    grid.fit(X, y)

    best_idx = grid.best_index_
    results = grid.cv_results_

    best_acc  = results["mean_test_accuracy"][best_idx]
    best_f1   = results["mean_test_f1_macro"][best_idx]
    best_bacc = results["mean_test_balanced_accuracy"][best_idx]

    print(f"{name} best params: {grid.best_params_}")
    print(f"{name} CV accuracy (mean):          {best_acc:.4f}")
    print(f"{name} CV macro F1 (mean):           {best_f1:.4f}")
```

```python
    print(f"{name} CV balanced accuracy (mean): {best_bacc:.4f}")

    summary_rows.append({
        "model": name,
        "cv_accuracy": best_acc,
        "cv_f1_macro": best_f1,
        "cv_balanced_accuracy": best_bacc,
    })

    if best_acc > best_cv_acc:
        best_cv_acc = best_acc
        best_model_name = name
        best_model = grid.best_estimator_

summary_df = pd.DataFrame(summary_rows).sort_values("cv_accuracy", ascending=False)
print("=" * 80)
print(summary_df.to_string(index=False))
print("\nChosen BEST model (by accuracy):", best_model_name)
print("Best CV accuracy:", best_cv_acc)
```

```
================================================================================
Fitting model: LogisticRegression

C:\Users\navsa\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:1247: FutureWarn
  warnings.warn(

LogisticRegression best params: {'clf__C': 0.5, 'clf__class_weight': 'balanced', 'clf__penal
LogisticRegression CV accuracy (mean):          0.6224
LogisticRegression CV macro F1 (mean):          0.6057
LogisticRegression CV balanced accuracy (mean): 0.6256
================================================================================
Fitting model: LDA
LDA best params: {}
LDA CV accuracy (mean):          0.5989
LDA CV macro F1 (mean):          0.5768
LDA CV balanced accuracy (mean): 0.6022
================================================================================
Fitting model: kNN

C:\Users\navsa\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1108: UserWarni
 0.50955882        nan 0.52720588 0.55110294 0.53970588        nan 0.51544118
```

```
 0.51507353 0.54448529        nan 0.55110294 0.49742647 0.50330882        nan
 0.51544118 0.55698529 0.53933824        nan 0.54595588 0.52132353 0.52095588]
  warnings.warn(
C:\Users\navsa\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1108: UserWarni
 0.50317157        nan 0.52297017 0.54101584 0.53279472        nan 0.50483037
 0.50187533 0.53617023        nan 0.54754533 0.48757363 0.49285733        nan
 0.5159955  0.55303465 0.53614774        nan 0.54395068 0.51468169 0.51534695]
  warnings.warn(
C:\Users\navsa\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1108: UserWarni
 0.51             nan 0.53111111 0.55222222 0.54333333        nan 0.51888889
 0.51444444 0.54666667        nan 0.55333333 0.49666667 0.50444444        nan
 0.51777778 0.55666667 0.54              nan 0.54888889 0.52111111 0.52111111]
  warnings.warn(

kNN best params: {'clf__n_neighbors': 13, 'clf__p': 2, 'clf__weights': 'uniform'}
kNN CV accuracy (mean):         0.5570
kNN CV macro F1 (mean):          0.5530
kNN CV balanced accuracy (mean): 0.5567
================================================================================
Fitting model: LinearSVM
LinearSVM best params: {'clf__C': 0.01, 'clf__class_weight': None}
LinearSVM CV accuracy (mean):         0.6169
LinearSVM CV macro F1 (mean):          0.6062
LinearSVM CV balanced accuracy (mean): 0.6178
================================================================================
Fitting model: DecisionTree
DecisionTree best params: {'clf__max_depth': 5, 'clf__min_samples_leaf': 4, 'clf__min_samples
DecisionTree CV accuracy (mean):         0.5511
DecisionTree CV macro F1 (mean):          0.5373
DecisionTree CV balanced accuracy (mean): 0.5511
================================================================================
Fitting model: RBFSVM (extra)
RBFSVM (extra) best params: {'clf__C': 1, 'clf__class_weight': None, 'clf__gamma': 'auto'}
RBFSVM (extra) CV accuracy (mean):         0.6287
RBFSVM (extra) CV macro F1 (mean):          0.6198
RBFSVM (extra) CV balanced accuracy (mean): 0.6267
================================================================================
             model  cv_accuracy  cv_f1_macro  cv_balanced_accuracy
    RBFSVM (extra)     0.628676     0.619811              0.626667
LogisticRegression     0.622426     0.605687              0.625556
         LinearSVM     0.616912     0.606159              0.617778
               LDA     0.598897     0.576770              0.602222
```

```
            kNN    0.556985    0.553035            0.556667
   DecisionTree    0.551103    0.537262            0.551111

Chosen BEST model (by accuracy): RBFSVM (extra)
Best CV accuracy: 0.6286764705882353
```

```
# Create submission using the best model
test_preds = best_model.predict(X_test_full)

submission = pd.DataFrame({
    "id_num": test["id_num"],
    "political_affiliation_predicted": test_preds,
})

submission_filename = f"CAH_submission_final_{best_model_name}.csv"
submission.to_csv(submission_filename, index=False)
print("Submission written to:", submission_filename)
display(submission.head())
```

```
Submission written to: CAH_submission_final_RBFSVM (extra).csv
```

	id_num	political_affiliation_predicted
0	2	Republican
1	3	Democrat
2	4	Independent
3	6	Independent
4	11	Independent