

```

"""
Project 2 (Template) – Practical Machine Learning & Data Exploration
Author: Mohamed Miqdad (محمد مقداد)

IMPORTANT (Academic Integrity):
- This file is a LEARNING TEMPLATE. You should review it, run it, and
- Replace the dataset path, choose your target column, and justify you
"""

from __future__ import annotations

import argparse
import warnings
from dataclasses import dataclass
from pathlib import Path

import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV, LeaveOneOut
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

from sklearn.feature_selection import SelectKBest, mutual_info_regression

warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

@dataclass
class RunConfig:
    data_path: Path
    target: str
    random_state: int = 42
    k_best: int = 8
    out_dir: Path = Path("project2_outputs")

def _to_numeric(series: pd.Series) -> pd.Series:
    """
    Convert a column to numeric, handling percent strings like '35%' to 35.0
    """
    if series.dtype == "object":
        s = series.astype(str).str.strip()
        # Convert percentages: '35%' -> 35.0

```

```

        is_percent = s.str.endswith("%").any()
        if is_percent:
            s = s.str.replace("%", "", regex=False)
            s = s.replace({"": np.nan, "nan": np.nan, "None": np.nan})
            out = pd.to_numeric(s, errors="coerce")
            return out
    return pd.to_numeric(series, errors="coerce")

def load_and_clean(cfg: RunConfig) -> tuple[pd.DataFrame, pd.Series]:
    df = pd.read_csv(cfg.data_path)

    # Basic cleanup: strip column names
    df.columns = [c.strip() for c in df.columns]

    if cfg.target not in df.columns:
        raise ValueError(
            f"Target column '{cfg.target}' not found. Available columns: {', '.join(df.columns)}"
        )

    # Convert all columns (except non-numeric text) to numeric when possible
    for col in df.columns:
        df[col] = _to_numeric(df[col])

    y = df[cfg.target]
    X = df.drop(columns=[cfg.target])

    return X, y

def build_preprocessor(X: pd.DataFrame) -> ColumnTransformer:
    # After _to_numeric, most columns should be numeric; still handle
    # categorical features
    numeric_features = [c for c in X.columns if pd.api.types.is_numeric_dtype(X[c])]
    categorical_features = [c for c in X.columns if c not in numeric_features]

    numeric_pipe = Pipeline(
        steps=[
            ("imputer", SimpleImputer(strategy="median")), # Step 2:
            ("scaler", StandardScaler()),
        ]
    )

    cat_pipe = Pipeline(
        steps=[
            ("imputer", SimpleImputer(strategy="most_frequent")),
            ("onehot", OneHotEncoder(handle_unknown="ignore")),
        ]
    )

    return ColumnTransformer(
        transformers=[
            ("num", numeric_pipe, numeric_features),
            ("cat", cat_pipe, categorical_features),
        ],
    )

```

```

        remainder="drop",
    )

def evaluate_regression(y_true: np.ndarray, y_pred: np.ndarray) -> dict:
    return {
        "MAE": float(mean_absolute_error(y_true, y_pred)),
        "RMSE": float(mean_squared_error(y_true, y_pred, squared=False)),
        "R2": float(r2_score(y_true, y_pred)),
    }

def run_grid_search(cfg: RunConfig, X: pd.DataFrame, y: pd.Series) -> pd.DataFrame:
    pre = build_preprocessor(X)

    # Models + hyperparameter grids (Step 3)
    models = [
        ("ridge", Ridge(), {"model__alpha": [0.1, 1.0, 10.0, 100.0]}),
        ("svr", SVR(), {"model__C": [0.1, 1, 10, 100], "model__epsilon": [0.1, 0.01, 0.001]}),
        ("rf", RandomForestRegressor(random_state=cfg.random_state), {"model__n_estimators": [10, 20, 30, 40, 50]}),
    ]

    # With tiny datasets, Leave-One-Out CV is safer than train/test split
    cv = LeaveOneOut()

    rows = []
    for name, model, grid in models:
        pipe = Pipeline([("pre", pre), ("model", model)])
        gs = GridSearchCV(
            pipe,
            param_grid=grid,
            cv=cv,
            scoring="neg_mean_absolute_error",
            n_jobs=-1,
        )
        gs.fit(X, y)

        # Evaluate on the same CV style: predict via cross-val manually
        preds = np.zeros(len(y), dtype=float)
        for i, (train_idx, test_idx) in enumerate(cv.split(X)):
            best = gs.best_estimator_
            best.fit(X.iloc[train_idx], y.iloc[train_idx])
            preds[test_idx[0]] = best.predict(X.iloc[test_idx])[0]

        metrics = evaluate_regression(y.values, preds)
        rows.append(
            {
                "model": name,
                "best_params": gs.best_params_,
                **metrics,
            }
        )

    return pd.DataFrame(rows).sort_values("MAE")

```

```

def run_feature_selection_and_models(cfg: RunConfig, X: pd.DataFrame,
    """
    Step 4 + Step 5:
    - Feature selection method 1: SelectKBest(mutual_info_regression)
    - Feature selection method 2: RFE with Ridge
    Then grid search models again on selected features.
    """
    # NOTE:
    # Feature selection is easiest on numeric-only data. Here, we already have numeric data.
    # If you have real categorical columns, you can one-hot encode them before selection.
    X_num = X.copy()

    # Impute missing values before selection
    imputer = SimpleImputer(strategy="median")
    X_imp = pd.DataFrame(imputer.fit_transform(X_num), columns=X_num.columns)

    # Method 1: SelectKBest with mutual information
    k = min(cfg.k_best, X_imp.shape[1])
    skb = SelectKBest(score_func=mutual_info_regression, k=k)
    X_skb = pd.DataFrame(skb.fit_transform(X_imp, y), columns=X_imp.columns)
    selected_skb = list(X_skb.columns)

    # Method 2: RFE with Ridge
    rfe_k = min(k, X_imp.shape[1])
    rfe = RFE(estimator=Ridge(alpha=1.0), n_features_to_select=rfe_k)
    X_rfe = pd.DataFrame(rfe.fit_transform(X_imp, y), columns=X_imp.columns)
    selected_rfe = list(X_rfe.columns)

    # Re-run models on selected features (simple preprocessing: impute missing values)
    def model_eval_table(X_selected: pd.DataFrame, tag: str) -> list[tuple[str, float]]:
        cv = LeaveOneOut()
        pre = Pipeline([("imputer", SimpleImputer(strategy="median")),
                        ("scaler", StandardScaler())])

        models = [
            ("ridge", Ridge(), {"model__alpha": [0.1, 1.0, 10.0, 100.0]}),
            ("svr", SVR(), {"model__C": [0.1, 1, 10, 100], "model__epsilon": [0.001, 0.01, 0.1]}),
            ("rf", RandomForestRegressor(random_state=cfg.random_state))
        ]

        rows = []
        for name, model, grid in models:
            pipe = Pipeline([("pre", pre), ("model", model)])
            gs = GridSearchCV(pipe, param_grid=grid, cv=cv, scoring="neg_mean_squared_error")
            gs.fit(X_selected, y)

            preds = np.zeros(len(y), dtype=float)
            for train_idx, test_idx in cv.split(X_selected):
                best = gs.best_estimator_
                best.fit(X_selected.iloc[train_idx], y.iloc[train_idx])
                preds[test_idx[0]] = best.predict(X_selected.iloc[test_idx])

            metrics = evaluate_regression(y.values, preds)
            rows.append({"feature_selection": tag, "model": name, "best_score": metrics})

```

```

        return rows

rows = []
rows += model_eval_table(X_skb, f"SelectKBest_MI (k={k})")
rows += model_eval_table(X_rfe, f"RFE_Ridge (k={rfe_k})")

results = pd.DataFrame(rows).sort_values(["feature_selection", "M2"])
return results, selected_skb, selected_rfe

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, required=True, help="Path to data")
    parser.add_argument("--target", type=str, required=True, help="Target variable")
    parser.add_argument("--k_best", type=int, default=8, help="Number of features to select")
    args = parser.parse_args()

    cfg = RunConfig(data_path=Path(args.data), target=args.target, k_best=args.k_best)
    cfg.out_dir.mkdir(parents=True, exist_ok=True)

    X, y = load_and_clean(cfg)

    # Step 2 + Step 3
    base_results = run_grid_search(cfg, X, y)
    base_results.to_csv(cfg.out_dir / "01_base_models_gridsearch.csv", index=False)

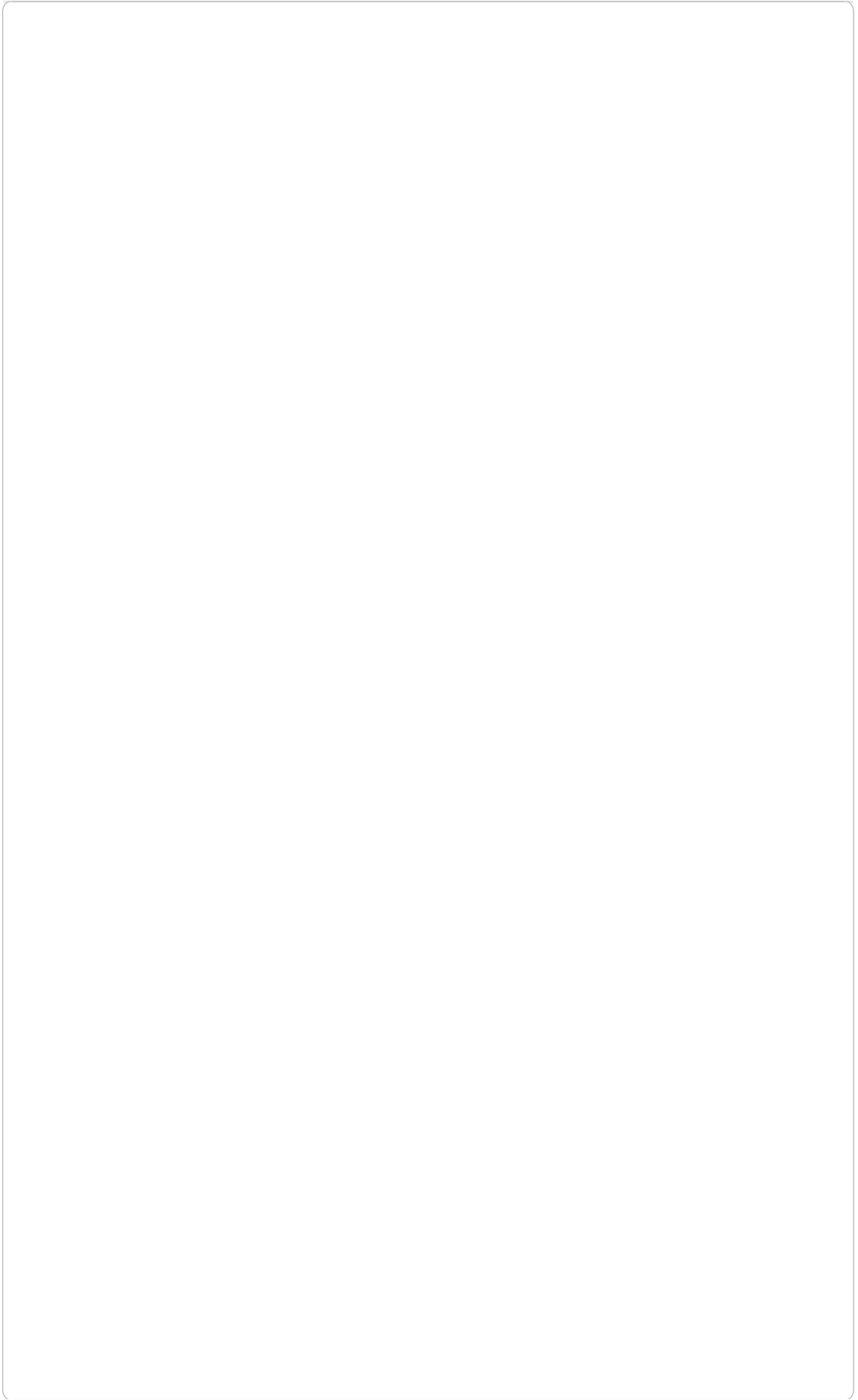
    # Step 4 + Step 5
    fs_results, selected_skb, selected_rfe = run_feature_selection_and_grid_search(cfg, X, y)
    fs_results.to_csv(cfg.out_dir / "02_feature_selected_models.csv", index=False)

    # Save selected feature lists
    (cfg.out_dir / "03_selected_features.txt").write_text(
        "SelectKBest (mutual information):\n"
        + "\n".join(selected_skb)
        + "\n\nRFE (Ridge):\n"
        + "\n".join(selected_rfe)
        + "\n"
    )

    print("\n✅ Done. Files saved to:", cfg.out_dir.resolve())
    print("\nBase (all features) results:\n", base_results.to_string(index=False))
    print("\nFeature-selected results:\n", fs_results.to_string(index=False))
    print("\nSelected features saved in 03_selected_features.txt")

if __name__ == "__main__":
    main()

```



```
usage: colab_kernel_launcher.py [-h] --data DATA --target TARGET
                                [--k_best K_BEST]
colab_kernel_launcher.py: error: the following arguments are required
ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.
```

Traceback (most recent call last):

```
File "/usr/lib/python3.12/argparse.py", line 1943, in _parse_known_
    namespace, args = self._parse_known_args(args, namespace, intermi
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/lib/python3.12/argparse.py", line 2230, in _parse_known_
    raise ArgumentError(None, _('the following arguments are required
argparse.ArgumentError: the following arguments are required: --data,
```

During handling of the above exception, another exception occurred:

Traceback (most recent call last):

```
File "/usr/local/lib/python3.12/dist-packages/IPython/core/interact
    exec(code_obj, self.user_global_ns, self.user_ns)
File "/tmp/ipython-input-1017825567.py", line 265, in <cell line: 0
    main()
File "/tmp/ipython-input-1017825567.py", line 234, in main
    args = parser.parse_args()
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/lib/python3.12/argparse.py", line 1904, in parse_args
    args, argv = self.parse_known_args(args, namespace)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/lib/python3.12/argparse.py", line 1914, in parse_known_a
    return self._parse_known_args2(args, namespace, intermixed=False)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/lib/python3.12/argparse.py", line 1945, in _parse_known_
    self.error(str(err))
File "/usr/lib/python3.12/argparse.py", line 2650, in error
    self.exit(2, _('%(prog)s: error: %(message)s\n') % args)
File "/usr/lib/python3.12/argparse.py", line 2637, in exit
    _sys.exit(status)
SystemExit: 2
```

During handling of the above exception, another exception occurred:

Next steps: [Explain error](#)

Traceback (most recent call last):

```
File "/usr/local/lib/python3.12/dist-packages/IPython/core/ultratb.
    return _fixed_getinnerframes(etb, number_of_lines_of_context, tb_
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/local/lib/python3.12/dist-packages/IPython/core/ultratb.
```

```
# ===== Project 2 (WineQT) - Run in Google Colab =====
```

```
!pip -q install scikit-learn pandas numpy
```

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, classification_

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.feature_selection import mutual_info_classif, SelectKBest

# 1) Load data (Colab uploads go to /content/)
data_path = "/content/WineQT (1).csv"
df = pd.read_csv(data_path)

# 2) Drop Id (not useful for ML)
if "Id" in df.columns:
    df = df.drop(columns=["Id"])

# 3) Define target
target = "quality"
if target not in df.columns:
    raise ValueError(f"Target '{target}' not found. Available columns:

X = df.drop(columns=[target])
y = df[target]

# (Optional) Make it classification-friendly: keep as multi-class
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Preprocess (all numeric)
numeric_features = X.columns.tolist()
preprocess = ColumnTransformer(
    transformers=[
        ("num", Pipeline(steps=[
            ("imputer", SimpleImputer(strategy="median")),
            ("scaler", StandardScaler())
        ]), numeric_features)
    ],
    remainder="drop"
)

models = {
    "LogReg": (LogisticRegression(max_iter=5000, multi_class="auto"),
               {"model__C": [0.1, 1, 10]}),
    "SVM": (SVC(), {
        "model__C": [0.5, 1, 5],
        "model__gamma": ["scale", "auto"]
    }),
    "RF": (RandomForestClassifier(random_state=42),
           {"model__max_depth": [None, 10, 2]}_
```

```
File "/tmp/ipython-input-862618059.py", line 62
  "RF": (RandomForestClassifier(random_state=42),
        ^
SyntaxError: '(' was never closed
```

Next steps: [Explain error](#)

```
===
```

```
cv, GridSearchCV
```

```
cv, classification_report
```

```
lassif, SelectKBest, RFE
```

```
Available columns: {list(df.columns)}")
```

```
as multi-class
```

```
(
?
```

```
lan")) ,
```

```
lti_class="auto"), {
```

```
{
```

```
"): :
```

```
')
```

```
l:.4f}")
```

```
unique()))
```

```
model", model)])
```

```
_weighted", n_jobs=-1)
```

```
ns_)
```

```
le=f"[Before Feature Selection] {name}")
```

```
l Information)
```

```
{_train),
```

```
om_state=42)
```

```
st_values(ascending=False)
```

```
ectKBest-like):")
```

```
lan")),
```

```
("model", model)])
```

```
_weighted", n_jobs=-1)
```

```
title=f"[After FS1: MutualInfo Top-{k}] {name}")
```

```
regression as estimator)
```

```
simplicity
```

```
transform(X_train)
```

```
ransform(X_test)
```

```
lect=8)
```

```
olist()
```

```
lan")),
```

```
("model", model)])
_weighted", n_jobs=-1)

title=f"[After FS2: RFE (8 feats)] {name}")

s + GridSearch + 2 Feature Selection methods + Evaluation.")
```

- ✓ Data loaded: (1143, 12)
- ✓ Target: quality | Classes: [np.int64(3), np.int64(4), np.int64(5),
- ✓ LogReg best params: {'model__C': 1}

```
=====
[Before Feature Selection] LogReg
Accuracy: 0.6245 | F1 (weighted): 0.6037
-----
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	7
5	0.70	0.78	0.74	97
6	0.59	0.63	0.61	92
7	0.43	0.31	0.36	29
8	0.00	0.00	0.00	3
accuracy			0.62	229
macro avg	0.29	0.29	0.28	229
weighted avg	0.59	0.62	0.60	229

- ✓ SVM best params: {'model__C': 5, 'model__gamma': 'scale'}

```
=====
[Before Feature Selection] SVM
Accuracy: 0.6594 | F1 (weighted): 0.6411
-----
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	7
5	0.71	0.75	0.73	97
6	0.62	0.70	0.66	92
7	0.61	0.48	0.54	29
8	0.00	0.00	0.00	3
accuracy			0.66	229
macro avg	0.32	0.32	0.32	229
weighted avg	0.63	0.66	0.64	229

- ✓ RF best params: {'model__max_depth': 20, 'model__n_estimators': 40}

```
=====
[Before Feature Selection] RF
Accuracy: 0.7118 | F1 (weighted): 0.6893
-----
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	7
5	0.73	0.86	0.79	97
6	0.69	0.72	0.70	92
7	0.70	0.48	0.57	29
8	0.00	0.00	0.00	0