5.5. Bankruptcy in Taiwan TW

```
import wqet_grader

wqet_grader.init("Project 5 Assessment")
```

```
# Import libraries here
import gzip
import json
import pickle

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.metrics import (ConfusionMatrixDisplay, classification_report, confusion_matrix)
from sklearn.model_selection import RandomizedSearchCV
#from my_predictor_assignment import make_predictions
#from joblib import dumb
from joblib import load

import wqet_grader
from sklearn.base import ClassifierMixin
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
import ipywidgets as widgets
from sklearn.ensemble import GradientBoostingClassifier
from teaching_tools.widgets import ConfusionMatrixWidget
```

Prepare Data

Import

Task 5.5.1: Load the contents of the `"data/taiwan-bankruptcy-data.json.gz"` and assign it to the variable `taiwan_data`.

Note that `taiwan_data` should be a dictionary. You'll create a DataFrame in a later task.

```
# Load data file
with gzip.open("data/taiwan-bankruptcy-data.json.gz","r") as read_file:
    taiwan_data = json.load(read_file)
print(type(taiwan_data))
```

```
<class 'dict'>
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.1", taiwan_data["metadata"])
```

Yup. You got it.
Score: 1

Task 5.5.2: Extract the key names from `taiwan_data` and assign them to the variable `taiwan_data_keys`.

> **Tip:** The data in this assignment might be organized differently than the data from the project, so be sure to inspect it first.

```
taiwan_data_keys = taiwan_data.keys()
print(taiwan_data_keys)
```

```
dict_keys(['schema', 'metadata', 'observations'])
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.2", list(taiwan_data_keys))
```

Wow, you're making great progress.
Score: 1

Task 5.5.3: Calculate how many companies are in `taiwan_data` and assign the result to `n_companies`.

Task 5.5.3: Calculate how many companies are in `taiwan_data` and assign the result to `n_companies`.

```
n_companies =  len(taiwan_data["observations"])
print(n_companies)
```

```
6137
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.3", [n_companies])
```

Excellent work.
Score: 1

Task 5.5.4: Calculate the number of features associated with each company and assign the result to `n_features`.

```
n_features =  len(taiwan_data["observations"][0])
print(n_features)
```

```
97
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.4", [n_features])
```

Yes! Great problem solving.
Score: 1

Task 5.5.5: Create a `wrangle` function that takes as input the path of a compressed JSON file and returns the file's contents as a DataFrame. Be sure that the index of the DataFrame contains the ID of the companies. When your function is complete, use it to load the data into the DataFrame `df`.

```
# Create wrangle function
def wrangle(filename):
    with gzip.open(filename, "r") as f:
        data = json.load(f)
    return pd.DataFrame().from_dict(data["observations"]).set_index("id")

def make_predictions(data_filepath, model_filepath):
    # Wrangle JSON file
    X_test = wrangle(data_filepath)
    # Load model
```

```python
def make_predictions(data_filepath, model_filepath):
    # Wrangle JSON file
    X_test = wrangle(data_filepath)
    # Load model
    with open(model_filepath,"rb") as f:
        model = pickle.load(f)
    # Generate predictions
    y_test_pred = model.predict(X_test)
    # Put predictions into Series with name "bankrupt", and same index as X_test
    y_test_pred = pd.Series(y_test_pred,index=X_test.index,name="bankrupt")
    return y_test_pred
```

```python
df = wrangle("data/taiwan-bankruptcy-data.json.gz")
print("df shape:", df.shape)
df.head()
```

```
df shape: (6137, 96)
```

id	bankrupt	feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	...	feat_86	feat_87	feat_88	feat_89	feat_
1	True	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	...	0.716845	0.009219	0.622879	0.601453	0.8278
2	True	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	...	0.795297	0.008323	0.623652	0.610237	0.8399
3	True	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	...	0.774670	0.040003	0.623841	0.601449	0.8367
4	True	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	...	0.739555	0.003252	0.622929	0.583538	0.8346
5	True	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	...	0.795016	0.003878	0.623521	0.598782	0.8399

5 rows × 96 columns

```python
wqet_grader.grade("Project 5 Assessment", "Task 5.5.5", df)
```

Excellent work.

Score: 1

Explore

Task 5.5.6: Is there any missing data in the dataset? Create a Series where the index contains the name of the columns in `df` and the values are the number of `NaN`s in each column. Assign the result to `nans_by_col`. Neither the Series itself nor its index require a name.

```
nans_by_col =  df.isnull().sum()
nans_by_col = pd.Series(nans_by_col)
print("nans_by_col shape:", nans_by_col.shape)
nans_by_col.head()
```

```
nans_by_col shape: (96,)
```

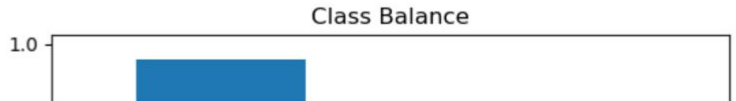
```
bankrupt    0
feat_1      0
feat_2      0
feat_3      0
feat_4      0
dtype: int64
```

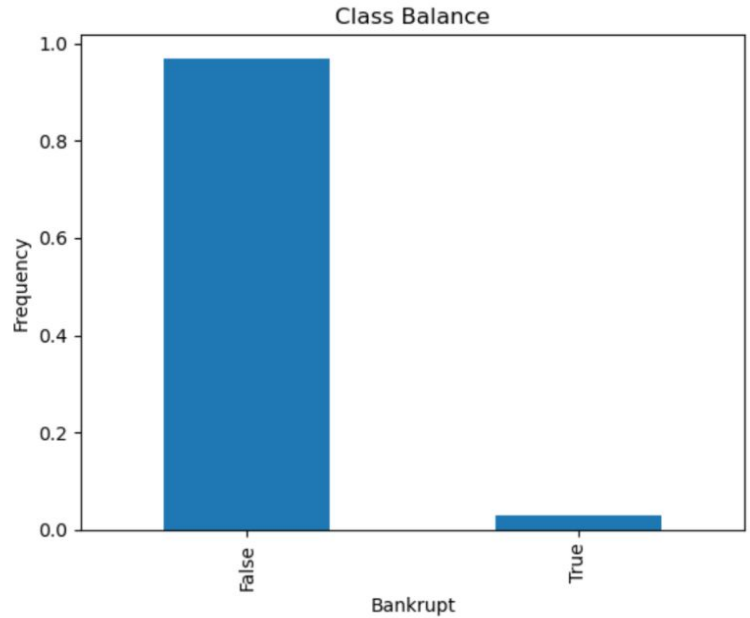
```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.6", nans_by_col)
```

Yup. You got it.
Score: 1

Task 5.5.7: Is the data imbalanced? Create a bar chart that shows the normalized value counts for the column `df["bankrupt"]`. Be sure to label your x-axis `"Bankrupt"`, your y-axis `"Frequency"`, and use the title `"Class Balance"`.

```
df["bankrupt"].value_counts(normalize=True).plot(
    kind="bar",
    xlabel="Bankrupt",
    ylabel="Frequency",
    title="Class Balance"
);
# Don't delete the code below 👇
plt.savefig("images/5-5-7.png", dpi=150)
```





]:
```
with open("images/5-5-7.png", "rb") as file:
    wqet_grader.grade("Project 5 Assessment", "Task 5.5.7", file)
```

You got it. Dance party time! 🕺 💃 🕺 💃

Score: 1

Split

Task 5.5.8: Create your feature matrix `X` and target vector `y`. Your target is `"bankrupt"`.

]:
```
target = "bankrupt"
X = df.drop(columns=target)
```

Split

Task 5.5.8: Create your feature matrix `X` and target vector `y`. Your target is `"bankrupt"`.

```
target = "bankrupt"
X = df.drop(columns=target)
y = df[target]
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (6137, 95)
y shape: (6137,)
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.8a", X)
```

Very impressive.
Score: 1

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.8b", y)
```

Yup. You got it.
Score: 1

Task 5.5.9: Divide your dataset into training and test sets using a randomized split. Your test set should be 20% of your data. Be sure to set `random_state` to `42`.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (4909, 95)
y_train shape: (4909,)
X_test shape: (1228, 95)
```

```
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (4909, 95)
y_train shape: (4909,)
X_test shape: (1228, 95)
y_test shape: (1228,)
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.9", list(X_train.shape))
```

Yup. You got it.
Score: 1

Resample

Task 5.5.10: Create a new feature matrix `X_train_over` and target vector `y_train_over` by performing random over-sampling on the training data. Be sure to set the `random_state` to `42`.

```
over_sampler = RandomOverSampler(random_state=42)
X_train_over, y_train_over = over_sampler.fit_resample(X_train, y_train)
print("X_train_over shape:", X_train_over.shape)
X_train_over.head()
```

```
X_train_over shape: (9512, 95)
```

	feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	feat_10	...	feat_86	feat_87	feat_88	feat_89	feat_9
0	0.535855	0.599160	0.594411	0.627099	0.627099	0.999220	0.797686	0.809591	0.303518	0.781865	...	0.834091	0.022025	0.624364	0.627101	0.84197
1	0.554136	0.612734	0.595000	0.607388	0.607388	0.999120	0.797614	0.809483	0.303600	0.781754	...	0.840293	0.002407	0.624548	0.607385	0.84264
2	0.549554	0.603467	0.599122	0.620166	0.620166	0.999119	0.797569	0.809470	0.303524	0.781740	...	0.840403	0.000840	0.624010	0.620163	0.84287
3	0.543801	0.603249	0.606992	0.622515	0.622515	0.999259	0.797728	0.809649	0.303510	0.781930	...	0.831514	0.006176	0.626775	0.622513	0.84298
4	0.498659	0.562364	0.546978	0.603670	0.603670	0.998904	0.797584	0.809459	0.304000	0.781713	...	0.811988	0.004256	0.623674	0.603669	0.84110

5 rows × 95 columns

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.10", list(X_train_over.shape))
```

5 rows × 95 columns

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.10", list(X_train_over.shape))
```

Very impressive.
Score: 1

Build Model

Iterate

Task 5.5.11: Create a classifier `clf` that can be trained on `(X_train_over, y_train_over)`. You can use any of the new, ensemble predictors you've learned about in this project.

```
clf = GradientBoostingClassifier(random_state=42)
#clf.fit(X_train_over, y_train_over)
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.11", clf)
```

Yup. You got it.
Score: 1

Task 5.5.12: Perform cross-validation with your classifier using the over-sampled training data, and assign your results to `cv_scores`. Be sure to set the `cv` argument to 5.

Tip: Use your CV scores to evaluate different classifiers. Choose the one that gives you the best scores.

```
cv_scores = cross_val_score(clf, X_train_over, y_train_over, cv=5, n_jobs=-1)
print(cv_scores)
```

```
[0.96952181 0.97162375 0.97003155 0.97160883 0.96845426]
```

```
cv_scores = cross_val_score(clf, X_train_over, y_train_over, cv=5, n_jobs=-1)
print(cv_scores)
```

```
[0.96952181 0.97162375 0.97003155 0.97160883 0.96845426]
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.12", list(cv_scores))
```

Python master 😁
Score: 1

Ungraded Task: Create a dictionary `params` with the range of hyperparameters that you want to evaluate for your classifier. If you're not sure which hyperparameters to tune, check the scikit-learn documentation for your predictor for ideas.

Tip: If the classifier you built is a predictor only (not a pipeline with multiple steps), you don't need to include the step name in the keys of your `params` dictionary. For example, if your classifier was only a random forest (not a pipeline containing a random forest), your would access the number of estimators using `"n_estimators"`, not `"randomforestclassifier__n_estimators"`.

```
params = {
    "max_depth": range(2,5),
    "n_estimators": range(20, 31, 5)
}
params
```

```
{'max_depth': range(2, 5), 'n_estimators': range(20, 31, 5)}
```

Task 5.5.13: Create a `GridSearchCV` named `model` that includes your classifier and hyperparameter grid. Be sure to set `cv` to 5, `n_jobs` to -1, and `verbose` to 1.

```
param_grid = {
    "max_depth": [20,10,20],
    "min_samples_split": [2,5,10]
}

model = GridSearchCV(
    clf,
    param_grid = params,
    cv=5,
```

```
model = GridSearchCV(
    clf,
    param_grid = params,
    cv=5,
    n_jobs=-1,
    verbose=1
)
model
```

```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.13", model)
```

Awesome work.
Score: 1

Ungraded Task: Fit your model to the over-sampled training data.

```
model.fit(X_train_over, y_train_over)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

Task 5.5.14: Extract the cross-validation results from your model, and load them into a DataFrame named `cv_results`. Looking at the results, which set of hyperparameters led to the best performance?

```
cv_results =  pd.DataFrame(model.cv_results_)
cv_results.head(5)
```

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score

Task 5.5.14: Extract the cross-validation results from your model, and load them into a DataFrame named `cv_results`. Looking at the results, which set of hyperparameters led to the best performance?

```
cv_results = pd.DataFrame(model.cv_results_)
cv_results.head(5)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score
0	5.748611	0.049510	0.004738	0.000157	2	20	{'max_depth': 2, 'n_estimators': 20}	0.909616	0.897530
1	7.218885	0.080246	0.018270	0.025792	2	25	{'max_depth': 2, 'n_estimators': 25}	0.912769	0.913820
2	8.625302	0.141315	0.017407	0.024446	2	30	{'max_depth': 2, 'n_estimators': 30}	0.923279	0.917499
3	8.374087	0.026469	0.004954	0.000624	3	20	{'max_depth': 3, 'n_estimators': 20}	0.929585	0.930636
4	11.082113	0.917117	0.017096	0.023929	3	25	{'max_depth': 3, 'n_estimators': 25}	0.935365	0.931687

```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.14", cv_results)
```

Yup. You got it.
Score: 1

Task 5.5.15: Extract the best hyperparameters from your model and assign them to `best_params`.

```
best_params = model.best_params_
print(best_params)
```

```
{'max_depth': 4, 'n_estimators': 30}
```

```
wqet_grader.grade(
    "Project 5 Assessment", "Task 5.5.15", [isinstance(best_params, dict)]
)
```

Yes! Keep on rockin'. 🎸 That's right.
Score: 1

Evaluate

Ungraded Task: Test the quality of your model by calculating accuracy scores for the training and test data.

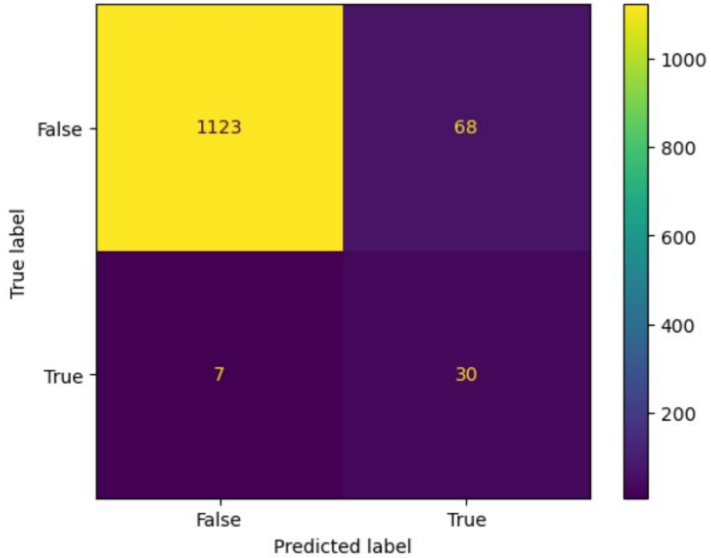
```
acc_train = model.score(X_train, y_train)
acc_test = model.score(X_test, y_test)

print("Model Training Accuracy:", round(acc_train, 4))
print("Model Test Accuracy:", round(acc_test, 4))
```

```
Model Training Accuracy: 0.9466
Model Test Accuracy: 0.9389
```

Task 5.5.16: Plot a confusion matrix that shows how your model performed on your test set.

```
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test);
# Don't delete the code below 👇
plt.savefig("images/5-5-16.png", dpi=150)
```




```
with open("images/5-5-16.png", "rb") as file:
    wqet_grader.grade("Project 5 Assessment", "Task 5.5.16", file)
```

Yup. You got it.
Score: 1

Task 5.5.17: Generate a classification report for your model's performance on the test data and assign it to `class_report`.

```
class_report = classification_report(y_test, model.predict(X_test))
print(class_report)
```

```
              precision    recall  f1-score   support

       False       0.99      0.94      0.97      1191
        True       0.31      0.81      0.44        37

    accuracy                           0.94      1228
```

```
              precision    recall  f1-score   support

       False       0.99      0.94      0.97      1191
        True       0.31      0.81      0.44        37

    accuracy                           0.94      1228
   macro avg       0.65      0.88      0.71      1228
weighted avg       0.97      0.94      0.95      1228
```

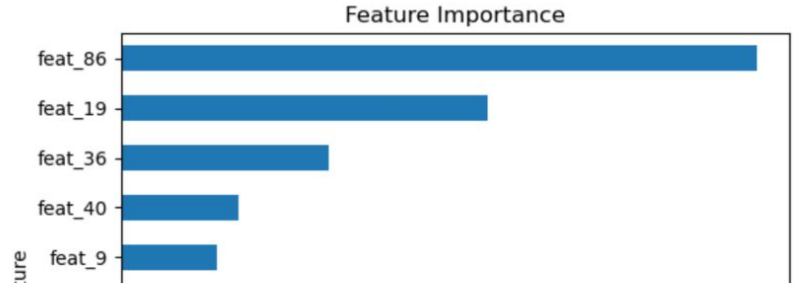
```
wqet_grader.grade("Project 5 Assessment", "Task 5.5.17", class_report)
```

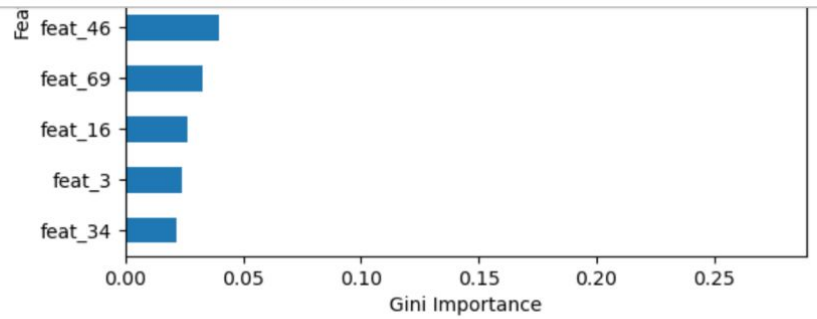
Score: 1

Communicate

Task 5.5.18: Create a horizontal bar chart with the 10 most important features for your model. Be sure to label the x-axis `"Gini Importance"`, the y-axis `"Feature"`, and use the title `"Feature Importance"`.

```
features = X_train_over.columns
importances = model.best_estimator_.feature_importances_
feat_imp = pd.Series(importances, index=features).sort_values()
feat_imp.tail(10).plot(kind="barh")
plt.xlabel("Gini Importance")
plt.ylabel("Feature")
plt.title("Feature Importance");
# Don't delete the code below 👇
plt.savefig("images/5-5-17.png", dpi=150)
```






```
with open("images/5-5-17.png", "rb") as file:
    wqet_grader.grade("Project 5 Assessment", "Task 5.5.18", file)
```

You got it. Dance party time! 🕺 💃 🕺 💃

Score: 1

Task 5.5.19: Save your best-performing model to a a file named `"model-5-5.pkl"`.

```
# Save model
with open("model-5-5.pkl", "wb") as f:
    pickle.dump(model, f)
```

```
with open("model-5-5.pkl", "rb") as f:
    wqet_grader.grade("Project 5 Assessment", "Task 5.5.19", pickle.load(f))
```

Excellent! Keep going.

Score: 1

Task 5.5.20: Open the file `my_predictor_assignment.py`. Add your `wrangle` function, and then create a `make_predictions` function that takes two arguments: `data_filepath` and `model_filepath`. Use the cell below to test your module. When you're satisfied with the result, submit it to the grader.

```
%%bash
```

```bash
%%bash

cat my_predictor_assignment.py
```

```
# Create your masterpiece :)
```

```
y_test_pred = make_predictions(
    data_filepath="data/taiwan-bankruptcy-data-test-features.json.gz",
    model_filepath="model-5-5.pkl",
)

print("predictions shape:", y_test_pred.shape)
y_test_pred.head()
```

```
predictions shape: (682,)
```

```
id
18    False
20    False
24     True
32     True
38    False
Name: bankrupt, dtype: bool
```

Tip: If you get an `ImportError` when you try to import `make_predictions` from `my_predictor_assignment`, try restarting your kernel. Go to the **Kernel** menu and click on **Restart Kernel and Clear All Outputs**. Then rerun just the cell above. ☝

```
wqet_grader.grade(
    "Project 5 Assessment",
    "Task 5.5.20",
    make_predictions(
        data_filepath="data/taiwan-bankruptcy-data-test-features.json.gz",
        model_filepath="model-5-5.pkl",
    ),
)
```