

In this final module, you will apply and demonstrate the full range of skills you have gained throughout the course. You will start with a practice project using the Titanic dataset to build and optimise classification models using pipelines, cross-validation, and hyperparameter tuning. Then, you will complete the final project by developing a rainfall prediction classifier using historical weather data. This includes data cleaning, feature engineering, model building, and evaluating performance. To conclude the course, you will take a graded final exam that tests your knowledge across all six modules. This module gives you the opportunity to showcase your learning in both practical and theoretical contexts.

## Learning Objectives

---

- Apply end-to-end machine learning techniques to develop, optimize, and evaluate a supervised classification model using real-world data.
- Practice pipeline optimization and model comparison using a real-world dataset.
- Demonstrate comprehensive understanding of machine learning concepts and techniques

## 1. Project Overview

You built and optimized two classification models (Random Forest and Logistic Regression) to predict **Titanic passenger survival**.

You used:

- **Pipelines** for clean preprocessing
- **ColumnTransformer** to handle numeric and categorical features
- **GridSearchCV** for hyperparameter tuning and cross-validation

---

## 2. Data Preparation

- Dropped irrelevant or high-missing-value features like **deck**, **embarked**, and **alive**.

Used the following relevant features:

```
['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'class', 'who', 'adult_male', 'alone']
```

- 
- Target variable: `survived`.

Class balance check:

```
y.value_counts()
# 0: 549 (62%)
# 1: 342 (38%)
```

✓ You correctly **stratified** the split to preserve class ratios:

```
train_test_split(..., stratify=y)
```

---

### 3. Preprocessing Pipeline

- **Numerical**: median imputation + scaling
  - **Categorical**: most frequent imputation + one-hot encoding
  - Combined using `ColumnTransformer`
- 

### 4. Random Forest Model

Grid search parameters:

```
{'n_estimators': [50, 100],
 'max_depth': [None, 10, 20],
 'min_samples_split': [2, 5]}
```

- 
- Used **Stratified 5-Fold Cross Validation**.
- Best accuracy on test data: **≈83%**

**Feature Importances (Top predictors likely):**

- `sex_female`
- `class_Third`
- `age`
- `fare`
- `who_man`

### Insight:

Random Forest shows **which features matter most**, but not *how* they affect survival.

For example, `sex_female` having high importance suggests being female increased survival chances.

---



## 5. Logistic Regression Model

You replaced the classifier in the pipeline:

```
pipeline.set_params(classifier=LogisticRegression(random_state=42))
```

Parameter grid:

```
{'solver': ['liblinear'],  
 'penalty': ['l1', 'l2'],  
 'class_weight': [None, 'balanced']}
```

### Results:

- Slightly improved or comparable accuracy (~82–83%)
- Slightly higher recall for survivors (class 1)
- Logistic Regression coefficients show **direction** and **strength** of influence.

✓ Unlike Random Forest, the logistic regression chart shows **positive vs negative influence** on survival probability.

---

## 6. Key Comparisons

| Model               | Accuracy | Strengths  | Weaknesses                                 |
|---------------------|----------|--|--|
| Random Forest       | ~83%     | Captures nonlinear relationships, feature importance interpretable | Harder to interpret direction of influence |
| Logistic Regression | ~82–83%  | Interpretable coefficients, faster                                 | May miss complex interactions              |

---

## 7. Visualizations

- **Confusion Matrix** — Showed balanced prediction ability across classes.
  - **Feature Importance (Random Forest)** — Highlighted which features were most predictive.
  - **Coefficient Magnitudes (Logistic Regression)** — Showed which features had the largest positive or negative effects.
- 

## 8. Conclusions

- The model correctly identifies **sex**, **class**, and **fare** as dominant predictors — consistent with historical facts.
  - Accuracy is solid for this dataset (~83%), and performance differences between models are small.
  - Pipelines + GridSearchCV make testing multiple models straightforward.
-

# Final Project Scenario

Welcome to the final project, Rainfall Prediction Classifier. In this project, you will apply the knowledge and skills learned in this course to build a machine learning classifier that predicts rainfall based on historical weather data.

You will demonstrate your data science and machine learning skills by performing the following tasks, each of which includes specific steps:

- Explore and prepare the dataset: Feature engineering and cleaning.
- Build a classifier pipeline: Model selection, training, and optimization.
- Evaluate the model's performance: Interpret metrics and visualizations.

Imagine yourself as a data scientist at WeatherTech Inc., responsible for building a model that can predict whether or not it will rain tomorrow based on historical weather data. The dataset provided includes various weather features like temperature, humidity, and wind speed.

Your submission will be auto-graded using AI Mark.

Note: After completing the final project Rainfall Prediction Classifier, download the notebook and upload it for final submission using the drag and drop option only.

# Notebook: FinalProject\_AUSWeather — Rainfall Prediction

# Final Project: Building a Rainfall Prediction Classifier

This notebook follows the project instructions and implements:

- data loading & cleaning
- feature engineering (season)
- pipeline + preprocessing
- RandomForest pipeline + GridSearchCV
- evaluation (classification report, confusion matrix)
- extracting feature importances
- switch to LogisticRegression pipeline + comparison

# Code cell: **imports**

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.compose import ColumnTransformer
```

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```
from sklearn.model_selection import train_test_split, GridSearchCV,  
StratifiedKFold
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report, confusion_matrix,  
ConfusionMatrixDisplay
```

```
# display settings
```

```
pd.options.display.max_columns = 200
```

```
sns.set_style('whitegrid')
```

## ## 2.2 Load the data

```
# Load dataset (URL from the lab)
```

```
url =
```

```
"https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/_0eYOqji3u  
nP1tDNKWZMjg/weatherAUS-2.csv"
```

```
df = pd.read_csv(url)
```

```
print("Raw rows:", len(df))
```

```
df.head()
```

### ### 2.2.1 Drop rows with missing values (simpler approach)

```
df = df.dropna()
```

```
print("Rows after dropna:", len(df))
```

```
df.info()
```

### ### 2.4 Rename rain columns for prediction of \*RainToday\* using prior-day info

```
df = df.rename(columns={'RainToday': 'RainYesterday', 'RainTomorrow':  
'RainToday'})
```

### ### 2.6 (Optional) Filter to Melbourne region (Melbourne, MelbourneAirport, Watsonia)

We follow the notebook's recommendation to focus on a local region.

```
df = df[df.Location.isin(['Melbourne', 'MelbourneAirport', 'Watsonia'])]  
print("Rows after location filter:", len(df))
```

### ### 2.7 Feature engineering: Season from Date

```
# helper
```

```
def date_to_season(date):
```

```
    month = date.month
```

```
    if month in (12, 1, 2):
```

```
        return 'Summer'
```

```
    elif month in (3, 4, 5):
```

```
        return 'Autumn'
```

```
    elif month in (6, 7, 8):
```

```
    return 'Winter'
```

```
else:
```

```
    return 'Spring'
```

```
# Map dates -> seasons
```

```
df['Date'] = pd.to_datetime(df['Date'])
```

```
df['Season'] = df['Date'].apply(date_to_season)
```

```
df = df.drop(columns=['Date'])
```

```
df.head()
```

## 2.9 Define feature (X) and target (y)

```
# choose features per the notebook; drop some columns that are not useful /  
cause leakage
```

```
# We'll use a set of columns similar to the lab. Adjust if you'd like to  
include/exclude others.
```

```
features = ['Location', 'MinTemp', 'MaxTemp', 'RainYesterday', 'Evaporation',
```

```
            'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
```

```
            'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
```

```
            'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
```

```
            'Temp9am', 'Temp3pm', 'Season', 'Rainfall', 'Sunshine', 'Year'] # Year if  
desired
```

```
# Many datasets might not have 'Year' column; create if missing
```

```

if 'Year' not in df.columns:

    df['Year'] = pd.to_datetime(df.index, errors='coerce') # placeholder

    # remove Year if invalid

    df = df.drop(columns=['Year'])

# remove columns not in df to avoid KeyError

features = [f for f in features if f in df.columns]

X = df[features].copy()

y = df['RainToday'].copy() # target is 'Yes'/'No'

# Convert target to binary if it's 'Yes'/'No'

if y.dtype == object:

    y = y.map({'Yes':1,'No':0})

print("Features used:", features)

print("X shape:", X.shape, "y shape:", y.shape)

```

## 2.10 Check class balance

```

print("Class counts for target (0 = No, 1 = Yes):")

print(y.value_counts())

print("\nPercentage distribution:")

print(y.value_counts(normalize=True).mul(100).round(2))

```

## 2.12 Split data (stratify)

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

```
print("Train/test sizes:", X_train.shape, X_test.shape)
```

## 2.13–2.14 Detect numeric & categorical features automatically

```
numeric_features = X_train.select_dtypes(include=['number']).columns.tolist()
```

```
categorical_features =  
X_train.select_dtypes(include=['object', 'category', 'bool']).columns.tolist()
```

```
print("Numeric features:", numeric_features)
```

```
print("Categorical features:", categorical_features)
```

### 2.14.1 Define preprocessing transformers

```
from sklearn.impute import SimpleImputer
```

```
# numeric transformer: median impute + scale
```

```
numeric_transformer = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='median')),  
    ('scaler', StandardScaler())  
])
```

# categorical transformer: most frequent impute + one-hot

```
categorical_transformer = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='most_frequent')),  
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse=False))  
])
```

## 2.15 Combine transformers into ColumnTransformer

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

## 2.16 Create pipeline with RandomForestClassifier

```
pipeline = Pipeline(steps=[  
    ('preprocessor', preprocessor),  
    ('classifier', RandomForestClassifier(random_state=42))  
])
```

### 2.16.1 Parameter grid

```
param_grid = {
```

```
'classifier__n_estimators': [50, 100],  
'classifier__max_depth': [None, 10, 20],  
'classifier__min_samples_split': [2, 5]  
}
```

## 2.17 Cross-validation selection

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

## 2.18 Instantiate and run GridSearchCV

```
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=cv,  
scoring='accuracy', verbose=2, n_jobs=-1)
```

```
grid_search.fit(X_train, y_train)
```

```
# print best params and cross-validation score
```

```
print("\nBest parameters found: ", grid_search.best_params_)
```

```
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
```

## 2.19 Show test set score

```
test_score = grid_search.score(X_test, y_test)
```

```
print("Test set score: {:.4f}".format(test_score))
```

## 2.20 Predictions on unseen data + classification report + confusion matrix

```
y_pred = grid_search.predict(X_test)

print("\nClassification Report:\n")

print(classification_report(y_test, y_pred, digits=4))

# Confusion matrix

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No','Yes'])

disp.plot(cmap='Blues')

plt.title('Confusion Matrix: RandomForest (test set)')

plt.show()
```

## 2.23–2.24 Feature importances from the trained RandomForest pipeline

We must combine numeric feature names and the OneHotEncoder output feature names for categoricals.

```
# Extract feature importances

rf = grid_search.best_estimator_['classifier']

# numeric feature names are unchanged

num_names = numeric_features

# get one-hot categorical feature names
```

```

ohe =
grid_search.best_estimator_['preprocessor'].named_transformers_['cat'].named_
steps['onehot']

cat_names = list(ohe.get_feature_names_out(categorical_features)) if
len(categorical_features)>0 else []

feature_names = num_names + cat_names

feature_importances = rf.feature_importances_

importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':
feature_importances}).sort_values(by='Importance', ascending=False)

importance_df.head(30)

# plot top N features

N = 20

top = importance_df.head(N).iloc[::-1]

plt.figure(figsize=(10,6))

plt.barh(top['Feature'], top['Importance'], color='skyblue')

plt.title(f'Top {N} Features — RandomForest')

plt.xlabel('Importance')

plt.show()

```

**2.26–2.27 Replace classifier with LogisticRegression, update grid, refit, and compare**

```
# Update pipeline classifier to LogisticRegression
```

```
pipeline.set_params(classifier=LogisticRegression(random_state=42,  
max_iter=200))
```

```
# reassign grid_search to use the same pipeline but different param grid
```

```
grid_search.estimator = pipeline
```

```
param_grid_log = {
```

```
    'classifier__solver' : ['liblinear'],
```

```
    'classifier__penalty': ['l1', 'l2'],
```

```
    'classifier__class_weight' : [None, 'balanced']
```

```
}
```

```
grid_search.param_grid = param_grid_log
```

```
# Fit grid_search for logistic
```

```
grid_search.fit(X_train, y_train)
```

```
print("Best params (logistic):", grid_search.best_params_)
```

```
print("Best CV score (logistic): {:.4f}".format(grid_search.best_score_))
```

```
# Evaluate on test set
```

```
y_pred_log = grid_search.predict(X_test)
```

```
print("\nClassification report — LogisticRegression:\n")
```

```
print(classification_report(y_test, y_pred_log, digits=4))
```

```

cm_log = confusion_matrix(y_test, y_pred_log)

disp = ConfusionMatrixDisplay(confusion_matrix=cm_log,
display_labels=['No', 'Yes'])

disp.plot(cmap='Blues')

plt.title('Confusion Matrix: LogisticRegression (test set)')

plt.show()

# compute accuracy and true positive rate (sensitivity) for comparison

acc_rf = rf.score(X_test, y_test)

acc_log =
grid_search.best_estimator_['classifier'].score(grid_search.best_estimator_['prepr
ocessor'].transform(X_test), y_test) \

    if hasattr(grid_search.best_estimator_['classifier'], 'score') else
grid_search.score(X_test, y_test)

# true positive rate for logistic

tn, fp, fn, tp = confusion_matrix(y_test, y_pred_log).ravel()

tpr_log = tp / (tp + fn) if (tp + fn) > 0 else np.nan

print(f"RandomForest test accuracy (best grid run): {test_score:.4f}")

print(f"LogisticRegression test accuracy: {grid_search.score(X_test, y_test):.4f}")

print(f"LogisticRegression true positive rate (recall for 'Yes'): {tpr_log:.4f}")

```

## 2.28 Feature coefficients for LogisticRegression (magnitude) — optional

```
# If logistic solver used supports coef_  
clf_log = grid_search.best_estimator_['classifier']  
if hasattr(clf_log, 'coef_'):  
    coefs = clf_log.coef_[0]  
    coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefs})  
    coef_df['AbsCoef'] = coef_df['Coefficient'].abs()  
    coef_df = coef_df.sort_values(by='AbsCoef', ascending=False)  
    plt.figure(figsize=(10,6))  
    plt.barh(coef_df['Feature'].head(20).iloc[::-1],  
coef_df['AbsCoef'].head(20).iloc[::-1], color='skyblue')  
    plt.title('Top 20 feature coefficient magnitudes — Logistic Regression')  
    plt.xlabel('Absolute coefficient')  
    plt.show()  
else:  
    print("No coef_ available for this classifier.")
```

---

---

# AI-Grader Answer Sheet

## 1. Which features did you use?

I used the following features (local Melbourne subset):

Location, MinTemp, MaxTemp, RainYesterday, Evaporation, WindGustDir, WindGustSpeed, WindDir9am, WindDir3pm, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm, Temp9am, Temp3pm, Season, Rainfall, Sunshine (only features available after dropping NaNs).

(You can view exact list in the notebook cell that prints `features`.)

---

## 2. How balanced are the classes?

Paste the output of `y.value_counts()` here after running the notebook. Example format:

0 (No rain): <count> — e.g. 1378

1 (Rain): <count> — e.g. 179

Percent distribution:

No: XX.XX% Yes: YY.YY%

(From the dataset we expect a **class imbalance** with "No rain" being the majority.)

---

## 3. Train/test split

I used `train_test_split(..., test_size=0.2, stratify=y, random_state=42)` so the class balance is preserved in train and test.

---

## 4. Preprocessing

- Numeric features: median imputation followed by `StandardScaler`.
  - Categorical features: most-frequent imputation followed by `OneHotEncoder(handle_unknown='ignore')`.
  - Combined with `ColumnTransformer`.
- 

## 5. GridSearchCV for RandomForest

- Parameter grid searched:
  - `classifier__n_estimators`: [50, 100]
  - `classifier__max_depth`: [None, 10, 20]
  - `classifier__min_samples_split`: [2, 5]
- Cross-validation: `StratifiedKFold(n_splits=5, shuffle=True, random_state=42)`

**Best parameters found:** (paste output of `grid_search.best_params_` here after running.)

**Best cross-validation score:** (paste `grid_search.best_score_`: e.g. 0.84)

**Test set accuracy (RF):** (paste `grid_search.score(X_test, y_test)`)

---

## 6. Classification report & confusion matrix (RandomForest)

Paste the printed classification report from the notebook (the `classification_report` output). Then paste the confusion matrix printed or shown by the notebook — for example:

Confusion matrix (rows = actual, cols = predicted):

```
[[ TN, FP],  
 [ FN, TP ]]
```

From the confusion matrix you can compute metrics; the notebook also prints the classification report (precision, recall, f1-score).

---

## 7. Feature importances (RandomForest)

The notebook computes feature importances and plots the top N features. Paste the top 5 features and their importance scores (from the `importance_df.head()` printout). Example format:

1. FeatureName1 — 0.123
  2. FeatureName2 — 0.091
- 

## 8. LogisticRegression: grid and eval

- Grid searched:
  - `classifier__solver`: ['liblinear']
  - `classifier__penalty`: ['l1','l2']
  - `classifier__class_weight`: [None, 'balanced']

**Best parameters found (Logistic):** (paste `grid_search.best_params_`)

**Best CV score (Logistic):** (paste `grid_search.best_score_`)

**Test set accuracy (Logistic):** (paste `grid_search.score(X_test, y_test)`)

**Classification report & confusion matrix (Logistic):** paste outputs from the notebook.

---

## 9. Comparison: RandomForest vs LogisticRegression (accuracy & TPR)

Provide these items as bullets:

1. **RandomForest test accuracy:** . . . (paste)
2. **LogisticRegression test accuracy:** . . . (paste)
3. **True Positive Rate (TPR / Recall for 'Yes') – RandomForest:** . . . (calculate from RF confusion matrix:  $TP/(TP+FN)$ )
4. **True Positive Rate (TPR) – LogisticRegression:** . . . (calculate from logistic confusion matrix:  $TP/(TP+FN)$ )
5. **Conclusion:** Summarize which model has higher accuracy and which has better TPR, and note any trade-offs (e.g., one model may have marginally higher accuracy while the other has better recall for the "Rain" class; class imbalance may require class-weighting or alternative metrics like ROC-AUC).

---

## 10. Final notes & suggestions

- Because "No rain" is the majority class, accuracy alone is not sufficient; TPR for the "Rain" class and precision/recall/f1 are important.
  - Consider additional work: calibrate threshold, try class-weight or resampling (SMOTE/undersampling), try more features or better imputation, include more locations or use location-specific models, add lagged features (e.g., previous 3 days rainfall), or try time-series models.
  - L1-regularized logistic regression coefficients give a different perspective on "importance" than tree-based importances; examine both and consider correlation among features before making causal claims.
-

# Summary — Machine Learning with Python

This course introduced both the **theoretical foundations** and **practical applications** of machine learning (ML), focusing on how ML models use **data and algorithms** to learn, improve accuracy, and support decision-making across industries.

## 1. What is Machine Learning?

- **ML is a subset of Artificial Intelligence (AI)** that enables computers to learn from data and make predictions or decisions without explicit programming.
- It is widely applied in:
  - **Healthcare** – predicting treatments and diagnoses
  - **Finance** – credit scoring and loan approval
  - **E-commerce** – personalized product recommendations

## 2. Types of Machine Learning

- **Supervised Learning:** Uses labeled data for regression and classification tasks.
- **Unsupervised Learning:** Finds patterns and groups in unlabeled data using clustering and dimensionality reduction.
- **Semi-supervised Learning:** Combines small labeled data with large unlabeled data.
- **Reinforcement Learning:** Models learn by interacting with environments and receiving rewards or penalties.

---

## 3. Key Machine Learning Models

- **Regression:**  
Predicts continuous outcomes.
  - *Simple Regression* → one independent variable
  - *Multiple Regression* → two or more independent variables

- *Logistic Regression* → predicts categorical outcomes (e.g., yes/no)
  - **Classification:**  
Predicts discrete labels. Examples: email filtering, handwriting recognition, speech-to-text.  
Algorithms include:
    - **K-Nearest Neighbors (KNN)** – classifies based on similarity to nearby data points
    - **Support Vector Machines (SVM)** – separates classes using hyperplanes
    - **Decision Trees** – use conditional rules for classification
    - **Regression Trees** – similar to decision trees but predict continuous values
- 

#### 4. Unsupervised Learning Techniques

- **Clustering:** Groups data by similarity (used in customer segmentation, market analysis).
  - **Dimensionality Reduction:** Simplifies large datasets while preserving critical information; improves visualization and reduces noise.
  - **Feature Engineering:** Creates meaningful input variables that improve model performance.
- 

#### 5. Model Evaluation & Validation

- **Classification Metrics:** Accuracy, Precision, Recall, and Confusion Matrix.
- **Regression Metrics:** Evaluate prediction errors for continuous targets.
- **Validation:** Ensures models generalize to unseen data and prevents **overfitting**.
  - *Important:* Avoid “data snooping” — don’t test before model tuning is complete.
  - Use **cross-validation** and **hyperparameter tuning** properly.

---

## 6. Tools and ML Pipelines

- ML pipelines include **data preprocessing, model training, evaluation, optimization, and deployment.**
- Tools simplify handling **big data**, performing **statistical analyses**, and making **accurate predictions.**

---

## 7. Learning Approach

- Reinforce your understanding by:
  - Completing **practice assessments and graded evaluations.**
  - Engaging with **hands-on labs** and a **final project** for real-world experience.
  - Referring to **summaries and glossaries** in each module for quick revision.

---

## Key Takeaways

- Machine Learning empowers systems to **analyze data, recognize patterns, and make data-driven decisions.**
  - Core skills: **Statistics, programming (Python), data handling, and model evaluation.**
  - Avoid data leakage, ensure fair evaluation, and focus on **continuous learning and ethical AI use.**
  - You are now equipped to apply ML techniques in **real-world projects** across industries.
-