Final Project: Building a Rainfall Prediction Classifier

Estimated time needed: **60** minutes

Objectives

After completing this lab you will be able to:

- Explore and perform feature engineering on a real-world data set
- Build a classifier pipeline and optimize it using grid search cross validation
- Evaluate your model by interpreting various performance metrics and visualizations
- Implement a different classifier by updating your pipeline
- Use an appropriate set of parameters to search over in each case

Instruction(s)

After completing the Notebook:

- Download the notebook using **File** > **Download**.
- This notebook will be then graded using **AI grader** in the subsequent section.
- Copy/Paste your markdown responses in the subsequent **AI Mark assignment**.

About The Dataset

The original source of the data is Australian Government's Bureau of Meteorology and the latest data can be gathered from http://www.bom.gov.au/climate/dwo/.

The dataset you'll use in this project was downloaded from Kaggle at
https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/
Column definitions were gathered from
http://www.bom.gov.au/climate/dwo/IDCJDW0000.shtml

The dataset contains observations of weather metrics for each day from 2008 to 2017, and includes the following fields:

Field	Description	Unit	Type
Date	Date of the Observation in YYYY-MM-DD	Date	object
Location	Location of the Observation	Location	object
MinTemp	Minimum temperature	Celsius	float
MaxTemp	Maximum temperature	Celsius	float
Rainfall	Amount of rainfall	Millimeters	float
Evaporation	Amount of evaporation	Millimeters	float
Sunshine	Amount of bright sunshine	hours	float
WindGustDir	Direction of the strongest gust	Compass Points	object
WindGustSpeed	Speed of the strongest gust	Kilometers/Hour	object
WindDir9am	Wind direction averaged over 10 minutes prior to 9am	Compass Points	object
WindDir3pm	Wind direction averaged over 10 minutes prior to 3pm	Compass Points	object
WindSpeed9am	Wind speed averaged over 10 minutes prior to 9am	Kilometers/Hour	float
WindSpeed3pm	Wind speed averaged over 10 minutes prior to 3pm	Kilometers/Hour	float
Humidity9am	Humidity at 9am	Percent	float
Humidity3pm	Humidity at 3pm	Percent	float
Pressure9am	Atmospheric pressure reduced to mean sea level at 9am	Hectopascal	float
Pressure3pm	Atmospheric pressure reduced to mean sea level at 3pm	Hectopascal	float
Cloud9am	Fraction of the sky obscured by cloud at 9am	Eights	float
Cloud3pm	Fraction of the sky obscured by cloud at 3pm	Eights	float
Temp9am	Temperature at 9am	Celsius	float
Temp3pm	Temperature at 3pm	Celsius	float
RainToday	If there was at least 1mm of rain today	Yes/No	object
RainTomorrow	If there is at least 1mm of rain tomorrow	Yes/No	object

Install and import the required libraries

Exectue the following cells to install and import the necessary libraries.

In [1]:
```
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install scikit-learn
!pip install seaborn
```

```
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages (2.
2.4)
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages (2.
2.3)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-packa
ges (from pandas) (2.2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.12/s
ite-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-packag
es (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-pack
ages (from pandas) (2025.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages
(from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.12/site-packages
(3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site-pa
ckages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-packag
es (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/site-p
ackages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/site-p
ackages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-package
s (from matplotlib) (2.2.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packages
(from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.12/site-pa
ckages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages
(from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.12/site-packag
es (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-packa
ges (from scikit-learn) (2.2.4)
Requirement already satisfied: scipy>=1.6.0 in /opt/conda/lib/python3.12/site-packag
es (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /opt/conda/lib/python3.12/site-packa
ges (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /opt/conda/lib/python3.12/sit
e-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: seaborn in /opt/conda/lib/python3.12/site-packages
(0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /opt/conda/lib/python3.12/sit
e-packages (from seaborn) (2.2.4)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-package
s (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /opt/conda/lib/python3.12/
site-packages (from seaborn) (3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site-pa
ckages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.1)
```

```
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-packag
es (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/site-p
ackages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/site-p
ackages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packages
(from matplotlib!=3.6.1,>=3.4->seaborn) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.12/site-pa
ckages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-packag
es (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-pack
ages (from pandas>=1.2->seaborn) (2025.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-packages
(from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
```

In [2]:

```python
import pandas as pd
import matplotlib.pyplot as plt
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, ConfusionMatri
import seaborn as sns
```

Load the data

Execute the following cells to load the dataset as a pandas dataframe.

In [3]:

```python
url="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/_0eYOqji3un
df = pd.read_csv(url)
df.head()
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	W
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	

5 rows × 23 columns

In [4]:

```
df.count()
```

Out[4]:

```
Date             145460
Location         145460
MinTemp          143975
MaxTemp          144199
Rainfall         142199
Evaporation       82670
Sunshine          75625
WindGustDir      135134
WindGustSpeed    135197
WindDir9am       134894
WindDir3pm       141232
WindSpeed9am     143693
WindSpeed3pm     142398
Humidity9am      142806
Humidity3pm      140953
Pressure9am      130395
Pressure3pm      130432
Cloud9am          89572
Cloud3pm          86102
Temp9am          143693
Temp3pm          141851
RainToday        142199
RainTomorrow     142193
dtype: int64
```

Sunshine and cloud cover seem like important features, but they have a lot of missing values, far too many to impute their missing values.

Drop all rows with missing values

To try to keep things simple we'll drop rows with missing values and see what's left

In [5]:
```
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 56420 entries, 6049 to 142302
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           56420 non-null  object 
 1   Location       56420 non-null  object 
 2   MinTemp        56420 non-null  float64
 3   MaxTemp        56420 non-null  float64
 4   Rainfall       56420 non-null  float64
 5   Evaporation    56420 non-null  float64
 6   Sunshine       56420 non-null  float64
 7   WindGustDir    56420 non-null  object 
 8   WindGustSpeed  56420 non-null  float64
 9   WindDir9am     56420 non-null  object 
 10  WindDir3pm     56420 non-null  object 
 11  WindSpeed9am   56420 non-null  float64
 12  WindSpeed3pm   56420 non-null  float64
 13  Humidity9am    56420 non-null  float64
 14  Humidity3pm    56420 non-null  float64
 15  Pressure9am    56420 non-null  float64
 16  Pressure3pm    56420 non-null  float64
 17  Cloud9am       56420 non-null  float64
 18  Cloud3pm       56420 non-null  float64
 19  Temp9am        56420 non-null  float64
 20  Temp3pm        56420 non-null  float64
 21  RainToday      56420 non-null  object 
 22  RainTomorrow   56420 non-null  object 
dtypes: float64(16), object(7)
memory usage: 10.3+ MB
```

Since we still have 56k observations left after dropping missing values, we may not need to impute any missing values.
Let's see how we do.

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

Data leakage considerations

Consider the descriptions above for the columns in the data set. Are there any practical limitations to being able to predict whether it will rain tomorrow given the available data?

Points to note - 1

List some of the features that would be inefficient in predicting tomorrow's rainfall. There will be a question in the quiz that follows based on this observation.

▶ Click here for Hint

If we adjust our approach and aim to predict today's rainfall using historical weather data up to and including yesterday, then we can legitimately utilize all of the available features. This shift would be particularly useful for practical applications, such as deciding whether you will bike to work today.

With this new target, we should update the names of the rain columns accordingly to avoid confusion.

In [7]:
```
df = df.rename(columns={'RainToday': 'RainYesterday',
                        'RainTomorrow': 'RainToday'
                        })
```

Data Granularity

Would the weather patterns have the same predictability in vastly different locations in Australia? I would think not.
The chance of rain in one location can be much higher than in another. Using all of the locations requires a more complex model as it needs to adapt to local weather patterns.
Let's see how many observations we have for each location, and see if we can reduce our attention to a smaller region.

Location selection

You could do some research to group cities in the `Location` column by distance, which I've done for you behind the scenes.
I found that Watsonia is only 15 km from Melbourne, and the Melbourne Airport is only 18 km from Melbourne.
Let's group these three locations together and use only their weather data to build our localized prediction model.
Because there might still be some slight variations in the weather patterns we'll keep `Location` as a categorical variable.

In [8]:
```
df = df[df.Location.isin(['Melbourne','MelbourneAirport','Watsonia',])]
df. info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7557 entries, 64191 to 80997
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           7557 non-null   object 
 1   Location       7557 non-null   object 
 2   MinTemp        7557 non-null   float64
 3   MaxTemp        7557 non-null   float64
 4   Rainfall       7557 non-null   float64
 5   Evaporation    7557 non-null   float64
 6   Sunshine       7557 non-null   float64
 7   WindGustDir    7557 non-null   object 
 8   WindGustSpeed  7557 non-null   float64
 9   WindDir9am     7557 non-null   object 
 10  WindDir3pm     7557 non-null   object 
 11  WindSpeed9am   7557 non-null   float64
 12  WindSpeed3pm   7557 non-null   float64
 13  Humidity9am    7557 non-null   float64
 14  Humidity3pm    7557 non-null   float64
 15  Pressure9am    7557 non-null   float64
 16  Pressure3pm    7557 non-null   float64
 17  Cloud9am       7557 non-null   float64
 18  Cloud3pm       7557 non-null   float64
 19  Temp9am        7557 non-null   float64
 20  Temp3pm        7557 non-null   float64
 21  RainYesterday  7557 non-null   object 
 22  RainToday      7557 non-null   object 
dtypes: float64(16), object(7)
memory usage: 1.4+ MB
```

We still have 7557 records, which should be enough to build a reasonably good model.
You could always gather more data if needed by partioning the data into similar locations or simplyby updating it from the source to include a larger time frame.

Extracting a seasonality feature

Now consider the `Date` column. We expect the weather patterns to be seasonal, having different predictablitiy levels in winter and summer for example.
There may be some variation with `Year` as well, but we'll leave that out for now. Let's engineer a `Season` feature from `Date` and drop `Date` afterward, since it is most likely less informative than season. An easy way to do this is to define a function that assigns seasons to given months, then use that function to transform the `Date` column.

Create a function to map dates to seasons

In [9]:
```
def date_to_season(date):
    month = date.month
    if (month == 12) or (month == 1) or (month == 2):
        return 'Summer'
    elif (month == 3) or (month == 4) or (month == 5):
```

```
        return 'Autumn'
    elif (month == 6) or (month == 7) or (month == 8):
        return 'Winter'
    elif (month == 9) or (month == 10) or (month == 11):
        return 'Spring'
```

Exercise 1: Map the dates to seasons and drop the Date column

Complete the code:

```
# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(...)

# Apply the function to the 'Date' column
df['Season'] = df['Date'].apply(date_to_season)

df=df.drop(columns=...)
df
```

In [10]:

```
# Write your response.
# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Apply the function to the 'Date' column
df['Season'] = df['Date'].apply(date_to_season)

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

Out[10]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustD
64191	MelbourneAirport	11.2	19.9	0.0	5.6	8.8	S
64192	MelbourneAirport	7.8	17.8	1.2	7.2	12.9	S
64193	MelbourneAirport	6.3	21.1	0.0	6.2	10.5	S
64194	MelbourneAirport	8.1	29.2	0.0	6.4	12.5	S
64195	MelbourneAirport	9.7	29.0	0.0	7.4	12.3	
...	...	...	...	...	...	...	
80992	Watsonia	3.6	14.5	0.0	2.4	8.8	NN
80994	Watsonia	4.8	13.3	0.4	0.6	0.0	NN
80995	Watsonia	5.6	13.1	0.0	1.6	6.0	NN
80996	Watsonia	6.9	12.1	3.2	1.8	5.6	SS
80997	Watsonia	7.9	13.0	0.0	2.8	3.8	NN

7557 rows × 23 columns

Looks like we have a good set of features to work with.

Let's go ahead and build our model.

But wait, let's take a look at how well balanced our target is.

Exercise 2. Define the feature and target dataframes

Complete the followng code:

```
X = df.drop(columns='...', axis=1)
y = df['...']
```

In [11]:

```
# Write your response.
X = df.drop(columns='Season', axis=1)
y = df['Season']
```

Exercise 3. How balanced are the classes?

Display the counts of each class.

Complete the following code:

```
... .value_counts()
```

In [12]:
```
# Write your response.
df['Season'].value_counts()
```

Out[12]:
```
Season
Autumn    1947
Winter    1918
Spring    1911
Summer    1781
Name: count, dtype: int64
```

Exercise 4. What can you conclude from these counts?

- How often does it rain annualy in the Melbourne area?
- How accurate would you be if you just assumed it won't rain every day?
- Is this a balanced dataset?
- Next steps?

In [ ]:
```
## Write your response here and convert the cell to a markdown.
```

Exercise 5. Split data into training and test sets, ensuring target stratification

Complete the followng code:

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

In [13]:
```
# Write your response.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y
```

Define preprocessing transformers for numerical and categorical features

Exercise 6. Automatically detect numerical and categorical columns and assign them to separate numeric and categorical features

Complete the followng code:

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

In [14]:
```
# Write your response.
numeric_features = X_train.select_dtypes(include=['number']).columns.tolist()
categorical_features = X_train.select_dtypes(include=['object', 'category']).column
```

Define separate transformers for both feature types and combine them into a single preprocessing transformer

In [15]:
```
# Scale the numeric features
numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])

# One-hot encode the categoricals
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle_unknown='
```

Exercise 7. Combine the transformers into a single preprocessing column transformer

Complete the followng code:

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

In [17]:
```
# Write your response.
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

Exercise 8. Create a pipeline by combining the preprocessing with a Random Forest classifier

Complete the following code:

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

In [18]:
```
# Write your response.
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

Define a parameter grid to use in a cross validation grid search model optimizer

In [19]:
```
param_grid = {
    'classifier__n_estimators': [50, 100],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5]
}
```

Pipeline usage in crossvalidation

Recall that the pipeline is repeatedly used within the crossvalidation by fitting on each internal training fold and predicting on its corresponding validation fold

Perform grid search cross-validation and fit the best model to the training data

Select a cross-validation method, ensuring target stratification during validation

In [20]:
```
cv = StratifiedKFold(n_splits=5, shuffle=True)
```

Exercise 9. Instantiate and fit GridSearchCV to the pipeline

Complete the followng code:

```
grid_search = GridSearchCV(..., param_grid, cv=..., scoring='accuracy',
verbose=2)
grid_search.fit(..., ...)
```

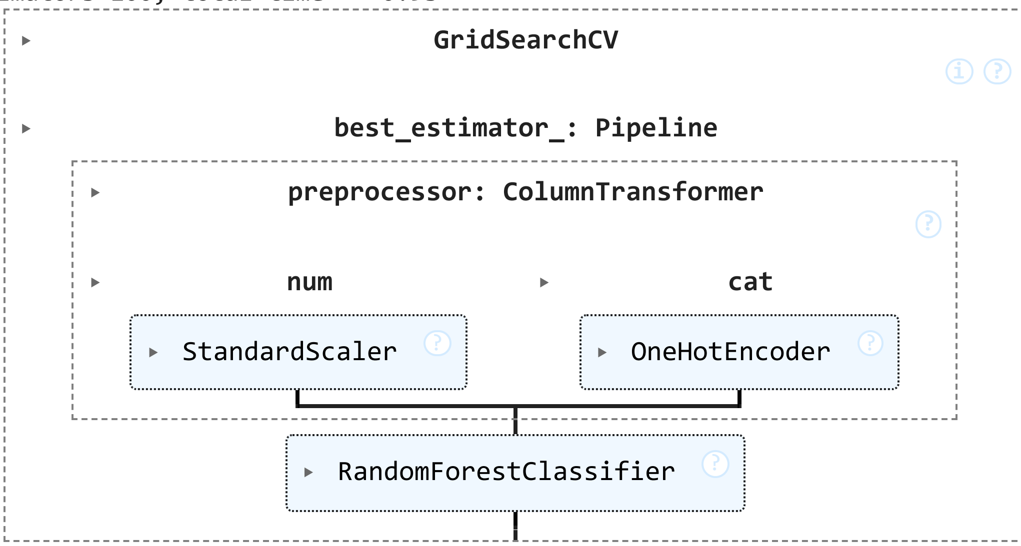
In [21]:
```
### Write your response.
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, cv=cv, scorin
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=50; total time=   1.3s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=50; total time=   1.1s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=50; total time=   0.7s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=50; total time=   0.5s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=50; total time=   0.5s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=100; total time=   1.0s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=2, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=50; total time=   0.5s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=50; total time=   0.4s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=50; total time=   0.5s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=50; total time=   0.4s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=50; total time=   0.4s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=100; total time=   0.8s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=100; total time=   0.8s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=None, classifier__min_samples_split=5, classifier__n_
estimators=100; total time=   0.9s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
```

```
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.3s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=10, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.6s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.5s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.5s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.5s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.5s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=50; total time=   0.5s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=2, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.4s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.4s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.4s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.4s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=50; total time=   0.4s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
```

```
timators=100; total time=   0.8s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.9s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.8s
[CV] END classifier__max_depth=20, classifier__min_samples_split=5, classifier__n_es
timators=100; total time=   0.9s
```

Out[21]:



Print the best parameters and best crossvalidation score

In [22]:

```python
print("\nBest parameters found: ", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters found:  {'classifier__max_depth': None, 'classifier__min_samples_spl
it': 2, 'classifier__n_estimators': 100}
Best cross-validation score: 0.76
```

Exercise 10. Display your model's estimated score

Complete the followng code:

```python
test_score = grid_search.score(..., ...)
print("Test set score: {:.2f}".format(test_score))
```

In [23]:

```python
## Write your response.
test_score = grid_search.score(X_test, y_test)
print("Test set score: {:.2f}".format(test_score))
```

```
Test set score: 0.76
```

So we have a reasonably accurate classifer, which is expected to correctly predict about 84% of the time whether it will rain today in the Melbourne area.
But careful here. Let's take a deeper look at the results.

The best model is stored within the gridsearch object.

Exercise 11. Get the model predictions from the grid search estimator on the unseen data

Complete the followng code:

```
y_pred = grid_search.predict(...)
```

In [24]:
```
### Write your response.
y_pred = grid_search.predict(X_test)
```

Exercise 12. Print the classification report

Complete the followng code:

```
print("\nClassification Report:")
print(...(y_test, y_pred))
```

In [25]:
```
## Write your response.
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

      Autumn       0.76      0.64      0.69       390
      Spring       0.70      0.65      0.68       382
      Summer       0.76      0.88      0.81       356
      Winter       0.81      0.87      0.84       384

    accuracy                           0.76      1512
   macro avg       0.76      0.76      0.76      1512
weighted avg       0.76      0.76      0.75      1512
```

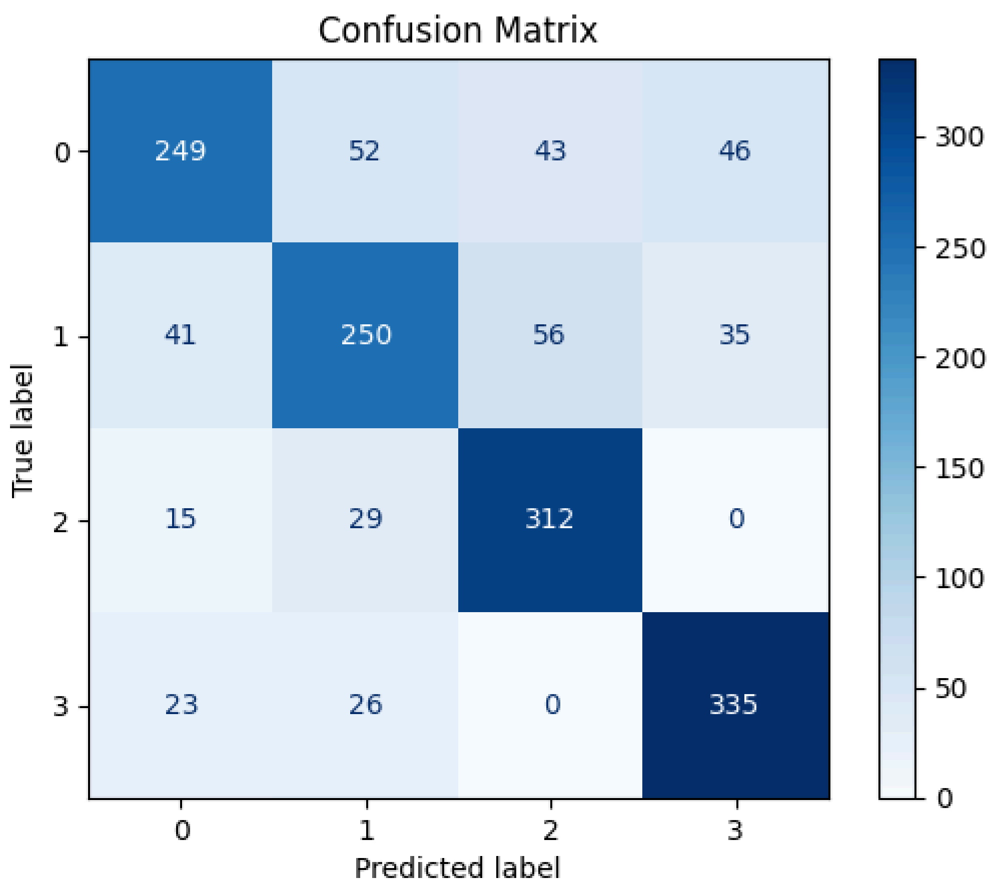
Exercise 13. Plot the confusion matrix

Complete the followng code:

```
conf_matrix = ...(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=...)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

In [27]:
```
## Write your response.
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
disp.plot(cmap='Blues')
```

```
plt.title('Confusion Matrix')
plt.show()
```



Let's consider wether the results indicate a good predictor of rainfall.

Points to note - 2

What is the true positive rate? There will be a question on this in the assignment that follows.

▶ Click here for Hints

Feature importances

Recall that to obtain the categorical feature importances, we have to work our way backward through the modelling pipeline to associate the feature importances with their original input variables, not the one-hot encoded ones. We don't need to do this for the numeric variables because we didn't modify their names in any way.
Remember we went from categorical features to one-hot encoded features, using the 'cat' column transformer.

Let's get all of the feature importances and associate them with their transformed features

Exercise 14. Extract the feature importances

Complete the followng code:

```
feature_importances = grid_search.best_estimator_['classifier']. ...
```

In [32]:
```
## Write your response.
feature_importances = grid_search.best_estimator_['classifier'].feature_importances
```

Now let's extract the feature importances and plot them as a bar graph.

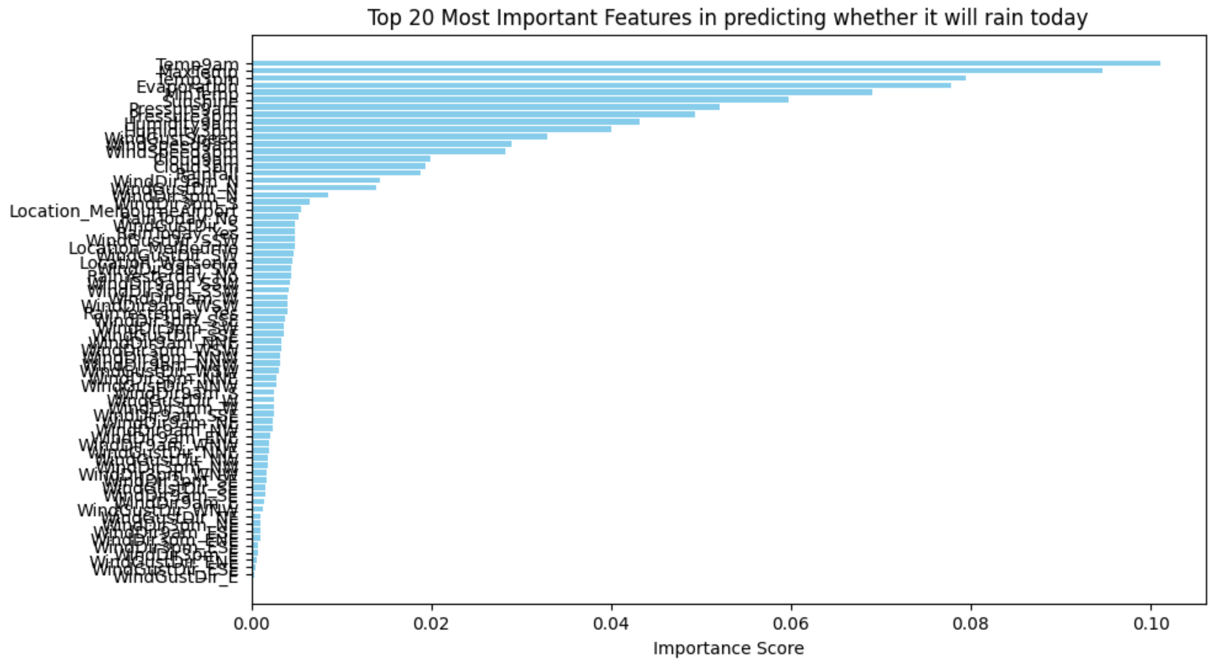
In [36]:
```
# Combine numeric and categorical feature names
feature_names = numeric_features + list(grid_search.best_estimator_['preprocessor']
                                        .named_transformers_['cat']
                                        .named_steps['onehot']
                                        .get_feature_names_out(categorical_features

feature_importances = grid_search.best_estimator_['classifier'].feature_importances


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



# Plotting
plt.figure(figsize=(10, 6))
plt.barh(top_features['Feature'], top_features['Importance'], color='skyblue')
plt.gca().invert_yaxis()  # Invert y-axis to show the most important feature on top
plt.title(f'Top {N} Most Important Features in predicting whether it will rain toda
plt.xlabel('Importance Score')
plt.show()
```



Point to note - 3

Identify the most important feature for predicting whether it will rain based on the feature importance bar graph. There will be a question on this in the assignment that follows.

Try another model

Some thoughts.

In practice you would want to try out different models and even revisit the data analysis to improve your model's performance. Maybe you can engineer better features, drop irrelevant or redundant ones, project your data onto a dimensional feature space, or impute missing values to be able to use more data. You can also try a larger set of parameters to define you search grid, or even engineer new features using cluster analysis. You can even include the clustering algorithm's hyperparameters in your search grid!

With Scikit-learn's powerful pipeline and GridSearchCV classes, this is easy to do in a few steps.

Exercise 15. Update the pipeline and the parameter grid

Let's update the pipeline and the parameter grid and train a Logistic Regression model and compare the performance of the two models. You'll need to replace the clasifier with LogisticRegression. We have supplied the parameter grid for you.

Complete the following code:

```
# Replace RandomForestClassifier with LogisticRegression
pipeline.set_params(...=LogisticRegression(random_state=42))

# update the model's estimator to use the new pipeline
grid_search.estimator = ...

# Define a new grid with Logistic Regression parameters
param_grid = {
    # 'classifier__n_estimators': [50, 100],
    # 'classifier__max_depth': [None, 10, 20],
    # 'classifier__min_samples_split': [2, 5],
    'classifier__solver' : ['liblinear'],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__class_weight' : [None, 'balanced']
}

grid_search.param_grid = ...

# Fit the updated pipeline with LogisticRegression
model.fit(..., ...)

# Make predictions
y_pred = model.predict(X_test)
```

In [41]:

```
## Write your response

# Replace RandomForestClassifier with LogisticRegression
pipeline.set_params(classifier=LogisticRegression(random_state=42))

# update the model's estimator to use the new pipeline
grid_search.estimator = pipeline

# Define a new grid with Logistic Regression parameters
param_grid = {
    # 'classifier__n_estimators': [50, 100],
    # 'classifier__max_depth': [None, 10, 20],
    # 'classifier__min_samples_split': [2, 5],
    'classifier__solver' : ['liblinear'],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__class_weight' : [None, 'balanced']
}

grid_search.param_grid = param_grid

# Fit the updated pipeline with LogisticRegression
grid_search.fit(X_train, y_train)

# Make predictions
y_pred = grid_search.best_estimator_.predict(X_test)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] END classifier__class_weight=None, classifier__penalty=l1, classifier__solver=l
iblinear; total time=   0.9s
[CV] END classifier__class_weight=None, classifier__penalty=l1, classifier__solver=l
iblinear; total time=   0.8s
[CV] END classifier__class_weight=None, classifier__penalty=l1, classifier__solver=l
iblinear; total time=   0.8s
[CV] END classifier__class_weight=None, classifier__penalty=l1, classifier__solver=l
iblinear; total time=   0.7s
[CV] END classifier__class_weight=None, classifier__penalty=l1, classifier__solver=l
iblinear; total time=   0.8s
[CV] END classifier__class_weight=None, classifier__penalty=l2, classifier__solver=l
iblinear; total time=   0.2s
[CV] END classifier__class_weight=None, classifier__penalty=l2, classifier__solver=l
iblinear; total time=   0.2s
[CV] END classifier__class_weight=None, classifier__penalty=l2, classifier__solver=l
iblinear; total time=   0.1s
[CV] END classifier__class_weight=None, classifier__penalty=l2, classifier__solver=l
iblinear; total time=   0.2s
[CV] END classifier__class_weight=None, classifier__penalty=l2, classifier__solver=l
iblinear; total time=   0.1s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1, classifier__solv
er=liblinear; total time=   0.8s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1, classifier__solv
er=liblinear; total time=   0.9s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1, classifier__solv
er=liblinear; total time=   0.8s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1, classifier__solv
er=liblinear; total time=   0.8s
[CV] END classifier__class_weight=balanced, classifier__penalty=l1, classifier__solv
er=liblinear; total time=   0.7s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2, classifier__solv
er=liblinear; total time=   0.1s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2, classifier__solv
er=liblinear; total time=   0.2s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2, classifier__solv
er=liblinear; total time=   0.2s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2, classifier__solv
er=liblinear; total time=   0.2s
[CV] END classifier__class_weight=balanced, classifier__penalty=l2, classifier__solv
er=liblinear; total time=   0.2s
```

Compare the results to your previous model.

Display the clasification report and the confusion matrix for the new model and compare your results with the previous model.

In [42]:
```python
print(classification_report(y_test, y_pred))

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
```

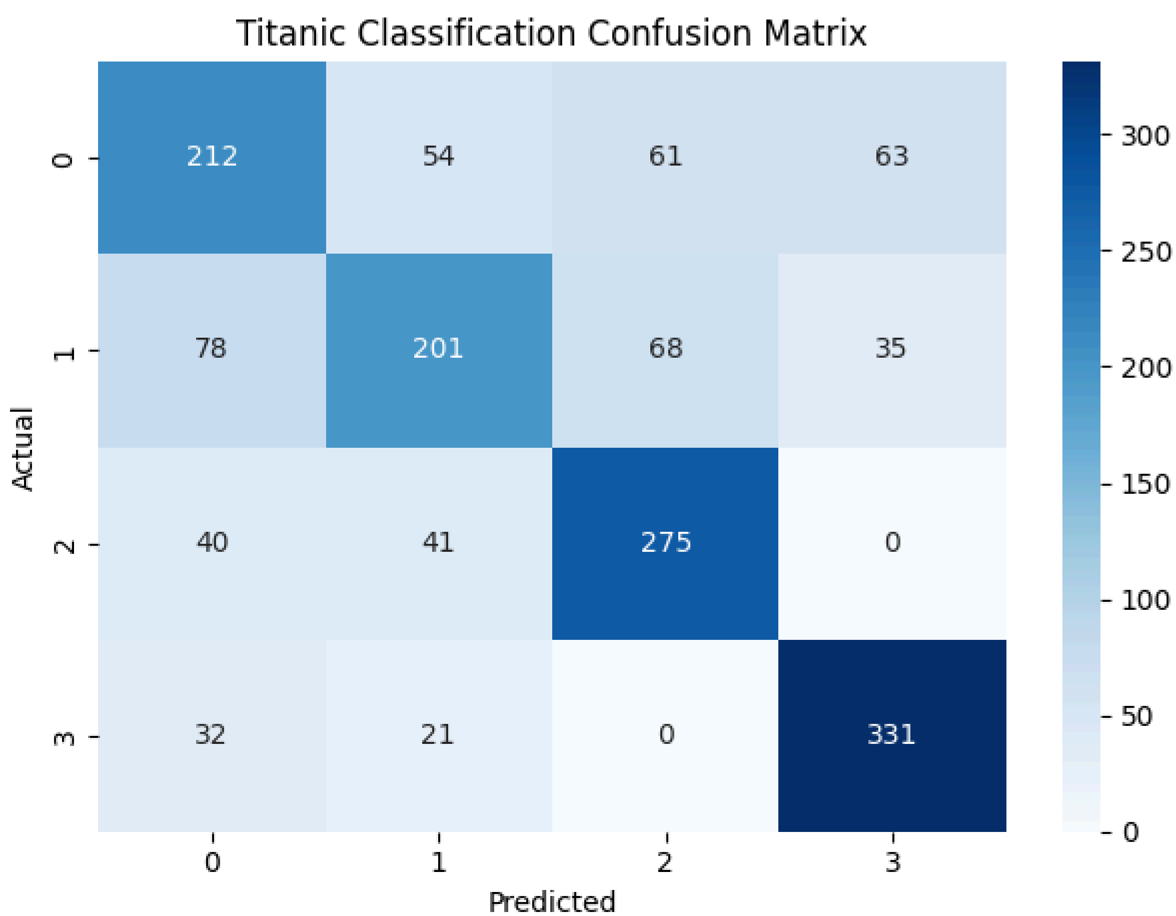
```python
# Set the title and labels
plt.title('Titanic Classification Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Show the plot
plt.tight_layout()
plt.show()
```

```
              precision    recall  f1-score   support

      Autumn       0.59      0.54      0.56       390
      Spring       0.63      0.53      0.58       382
      Summer       0.68      0.77      0.72       356
      Winter       0.77      0.86      0.81       384

    accuracy                           0.67      1512
   macro avg       0.67      0.68      0.67      1512
weighted avg       0.67      0.67      0.67      1512
```



What can you conclude about the model performances?

Points to note - 4

Compare the accuracy and true positive rate of rainfall predictions between the LogisticRegression model and the RandomForestClassifier model.

Note: Make sure to provide the answer in the form of a list using either bullets or numbers.

There will be a question on this in the assignment that follows.

▶ Click here for Hints

Congratulations! You've made it the end of your final project!

Well done! You now have some great tools to use for tackling complex real-world problems with machine learning.

Author

Jeff Grossman

Other Contributor(s)

Abhishek Gagneja