`Final Assignment - Team 11 - RAIN IN AUSTRALIA`

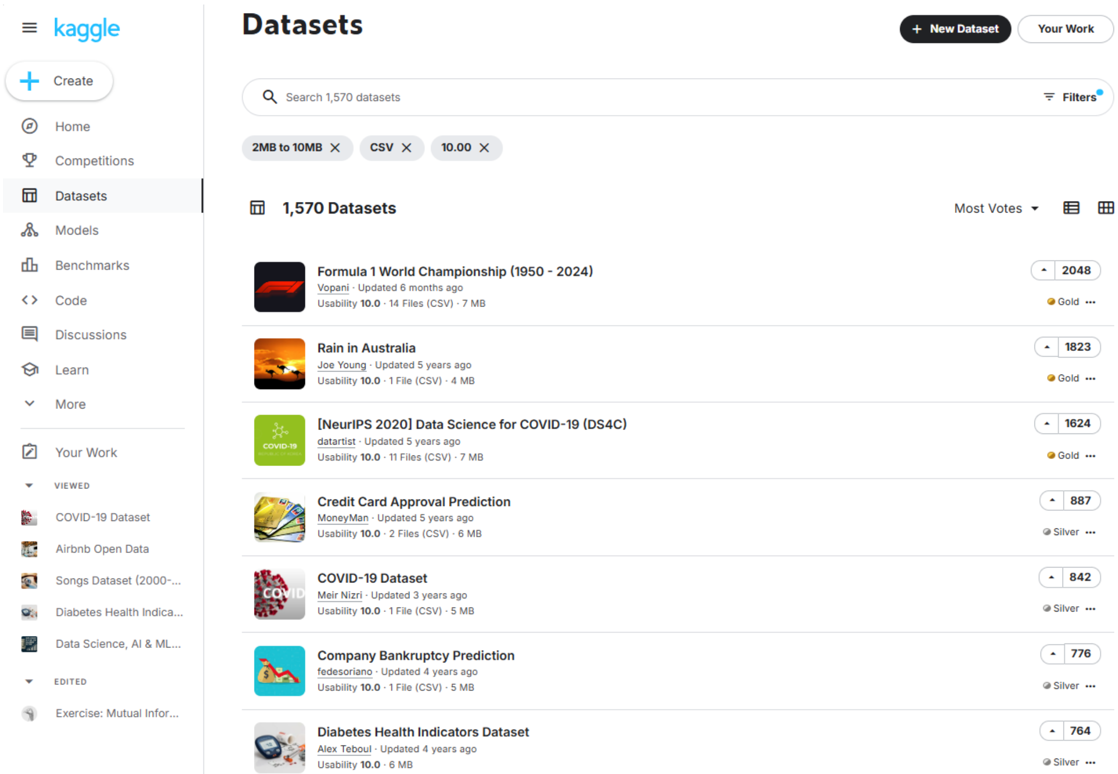
Team 11 Member

No.	Name	Student ID	Email
1	Ton That Tu Nguyen	tnguy113	tnguy113@vols.utk.edu
2	Jared Bell	jbell58	jbell58@vols.utk.edu
3	Ryan Petty	rpetty3	rpetty3@vols.utk.edu
4	Christopher Brown	cbrow308	cbrow308@vols.utk.edu

Details

Introduction about chosen dataset **Rain in Australia**

- Website: We can access Kaggle home page and choose Datasets in the left menu or use this link to go to Kaggle Datasets
- We will not use any keywords for our search. We set couple of filters for the dataset:
  - File size: Min = 2 MB; Max = 10 MB to make sure the dataset is large enough (>= 50,000 rows).
  - File type: CSV
  - Usability Rate = 10.0
  - Sort by: Most Votes



- Downloads zip file here.
- Extracts into 'data' folder and receives 'weatherAUS.csv' file.

Loading dataset

Initialize Spark Session

In [ ]:
```
# Load data into Spark data objects and explore structure, size and di
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
                    .appName("RainInAustraliaSparkApp") \
                    .getOrCreate()

# For lower memory usage
# spark = SparkSession.builder \
#                      .appName("RainInAustraliaSparkApp") \
#                      .config('spark.driver.memory', '3g') \
#                      .config('spark.executor.memory', '3g') \
#                      .getOrCreate()
```

In [ ]:
```
spark
```

Out[ ]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v4.0.0`
Master	`local[*]`
AppName	`RainInAustraliaSparkApp`

Define schemas for the dataset

In [ ]:
```
from pyspark.sql.types import *
import seaborn as sns
import matplotlib.pyplot as plt

data_path = "./data/weatherAUS.csv"

data_schema = StructType([
    StructField("Date", DateType(), True),
    StructField("Location", StringType(), True),
    StructField("MinTemp", FloatType(), True),
    StructField("MaxTemp", FloatType(), True),
    StructField("Rainfall", FloatType(), True),
    StructField("Evaporation", FloatType(), True),
    StructField("Sunshine", FloatType(), True),
    StructField("WindGustDir", StringType(), True),
    StructField("WindGustSpeed", FloatType(), True),
    StructField("WindDir9am", StringType(), True),
    StructField("WindDir3pm", StringType(), True),
    StructField("WindSpeed9am", FloatType(), True),
    StructField("WindSpeed3pm", FloatType(), True),
    StructField("Humidity9am", FloatType(), True),
    StructField("Humidity3pm", FloatType(), True),
    StructField("Pressure9am", FloatType(), True),
    StructField("Pressure3pm", FloatType(), True),
    StructField("Cloud9am", FloatType(), True),
    StructField("Cloud3pm", FloatType(), True),
    StructField("Temp9am", FloatType(), True),
    StructField("Temp3pm", FloatType(), True),
    StructField("RainToday", StringType(), True),
    StructField("RainTomorrow", StringType(), True)
])
```

Load dataset & create temporary view

In [ ]:
```
weather_df = spark.read.csv(path=data_path, header=True, schema=data_s
weather_df.cache()
weather_df.createOrReplaceTempView("weather_aus")
```

```
spark.sql("SELECT * FROM weather_aus LIMIT 5").show()
```

```
+----------+--------+-------+-------+--------+-----------+--------+----
-------+-------------+----------+----------+------------+------------+-
----------+-----------+-----------+-----------+--------+--------+------
-+-------+---------+------------+
|      Date|Location|MinTemp|MaxTemp|Rainfall|Evaporation|Sunshine|Wind
GustDir|WindGustSpeed|WindDir9am|WindDir3pm|WindSpeed9am|WindSpeed3pm|H
umidity9am|Humidity3pm|Pressure9am|Pressure3pm|Cloud9am|Cloud3pm|Temp9a
m|Temp3pm|RainToday|RainTomorrow|
+----------+--------+-------+-------+--------+-----------+--------+----
-------+-------------+----------+----------+------------+------------+-
----------+-----------+-----------+-----------+--------+--------+------
-+-------+---------+------------+
|2008-12-01|  Albury|   13.4|   22.9|     0.6|       NULL|    NULL|
W|         44.0|         W|       WNW|        20.0|        24.0|
71.0|       22.0|     1007.7|     1007.1|     8.0|    NULL|   16.9|   2
1.8|       No|          No|
|2008-12-02|  Albury|    7.4|   25.1|     0.0|       NULL|    NULL|
WNW|         44.0|       NNW|       WSW|         4.0|        22.0|
44.0|       25.0|     1010.6|     1007.8|    NULL|    NULL|   17.2|   2
4.3|       No|          No|
|2008-12-03|  Albury|   12.9|   25.7|     0.0|       NULL|    NULL|
WSW|         46.0|         W|       WSW|        19.0|        26.0|
38.0|       30.0|     1007.6|     1008.7|    NULL|     2.0|   21.0|   2
3.2|       No|          No|
|2008-12-04|  Albury|    9.2|   28.0|     0.0|       NULL|    NULL|
NE|          24.0|        SE|         E|        11.0|         9.0|
45.0|       16.0|     1017.6|     1012.8|    NULL|    NULL|   18.1|   2
6.5|       No|          No|
|2008-12-05|  Albury|   17.5|   32.3|     1.0|       NULL|    NULL|
W|         41.0|       ENE|        NW|         7.0|        20.0|
82.0|       33.0|     1010.8|     1006.0|     7.0|     8.0|   17.8|   2
9.7|       No|          No|
+----------+--------+-------+-------+--------+-----------+--------+----
-------+-------------+----------+----------+------------+------------+-
----------+-----------+-----------+-----------+--------+--------+------
-+-------+---------+------------+
```

Creating a few tables or charts

In [ ]:

```
# Basic dataset information
print("\nColumn information:")
weather_df.printSchema()

# Numerical features statistics
weather_df.describe().show()

# Visualize missing values
plt.figure(figsize=(15, 6))
sns.heatmap(weather_df.toPandas().isnull(), cbar=False, cmap='viridis'
```

```
plt.title('Missing Values Distribution')
plt.show()
```

```
Column information:
root
 |-- Date: date (nullable = true)
 |-- Location: string (nullable = true)
 |-- MinTemp: float (nullable = true)
 |-- MaxTemp: float (nullable = true)
 |-- Rainfall: float (nullable = true)
 |-- Evaporation: float (nullable = true)
 |-- Sunshine: float (nullable = true)
 |-- WindGustDir: string (nullable = true)
 |-- WindGustSpeed: float (nullable = true)
 |-- WindDir9am: string (nullable = true)
 |-- WindDir3pm: string (nullable = true)
 |-- WindSpeed9am: float (nullable = true)
 |-- WindSpeed3pm: float (nullable = true)
 |-- Humidity9am: float (nullable = true)
 |-- Humidity3pm: float (nullable = true)
 |-- Pressure9am: float (nullable = true)
 |-- Pressure3pm: float (nullable = true)
 |-- Cloud9am: float (nullable = true)
 |-- Cloud3pm: float (nullable = true)
 |-- Temp9am: float (nullable = true)
 |-- Temp3pm: float (nullable = true)
 |-- RainToday: string (nullable = true)
 |-- RainTomorrow: string (nullable = true)

+-------+--------+------------------+------------------+---------------
---+------------------+------------------+-----------+------------------
+----------+----------+------------------+------------------+----------
--------+------------------+------------------+------------------+-----
-------------+------------------+------------------+------------------+-
--------+------------+
|summary|Location|           MinTemp|           MaxTemp|          Rainf
all|       Evaporation|          Sunshine|WindGustDir|     WindGustSpeed
|WindDir9am|WindDir3pm|      WindSpeed9am|      WindSpeed3pm|        Hum
idity9am|       Humidity3pm|       Pressure9am|       Pressure3pm|
Cloud9am|          Cloud3pm|           Temp9am|           Temp3pm|RainTo
day|RainTomorrow|
+-------+--------+------------------+------------------+---------------
---+------------------+------------------+-----------+------------------
+----------+----------+------------------+------------------+----------
--------+------------------+------------------+------------------+-----
-------------+------------------+------------------+------------------+-
--------+------------+
|  count|  145460|            143975|            144199|            142
199|             82670|             75625|     145460|            135197
|    145460|    145460|            143693|            142398|
142806|            140953|            130395|            130432|
89572|            86102|            143693|            141851|   145460
|      145460|
```

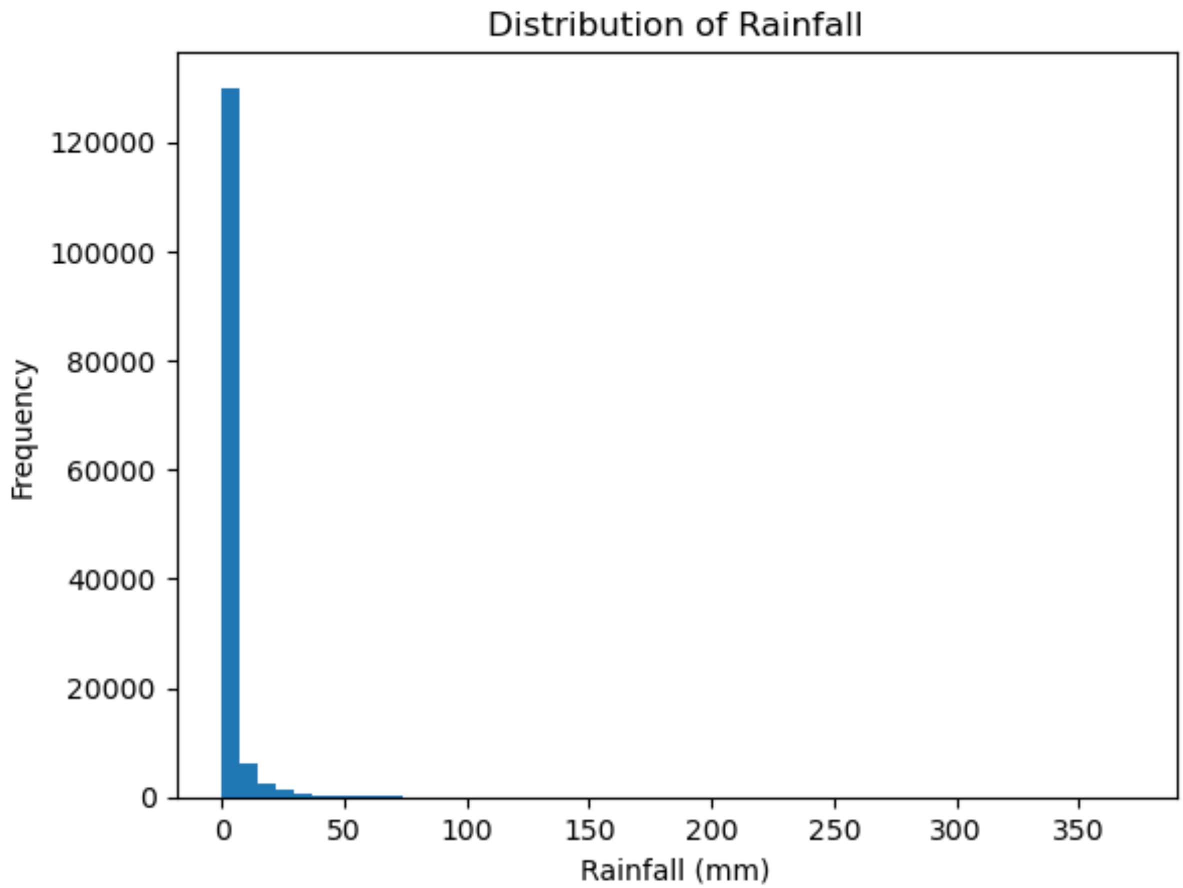
```
|   mean|    NULL|12.194034381779941|23.221348273321002|2.3609181508908
756|5.468231521887871| 7.611177522303053|       NULL| 40.03523007167319
|      NULL|      NULL|14.043425914971502|18.662656778887342| 68.880831
33761887|  51.5391158755046|1017.6499397947478|1015.2558887915008|4.447
4612602152455| 4.509930082924903|16.990631418377568|21.68339031665222|
NULL|        NULL|
| stddev|    NULL|  6.39849497591283| 7.119048841492804| 8.478059742817
674|4.193704096708969|3.7854829656916524|       NULL|13.607062267381373
|      NULL|      NULL| 8.915375322679528| 8.809800021251466|19.0291644
51844164|20.795901656021137| 7.106530159387171|7.0374136035671135|2.887
1588535172408|2.7203573103324614| 6.488753139523503|6.936650457604365|
NULL|        NULL|
|    min|Adelaide|              -8.5|              -4.8|
0.0|              0.0|               0.0|          E|              6.0
|         E|         E|               0.0|               0.0|
0.0|               0.0|             980.5|             977.1|
0.0|               0.0|              -7.2|              -5.4|       NA|
NA|
|    max| Woomera|              33.9|              48.1|             37
1.0|            145.0|              14.5|        WSW|            135.0
|       WSW|       WSW|             130.0|              87.0|
100.0|             100.0|            1041.0|            1039.6|
9.0|               9.0|              40.2|              46.7|      Yes|
Yes|
+-------+--------+------------------+------------------+---------------
---+-----------------+------------------+-----------+-----------------
+----------+----------+------------------+------------------+----------
--------+------------------+------------------+------------------+-----
-------------+------------------+------------------+------------------+-
--------+------------+
```

In [ ]:

```python
# Is the rainfall data normally distributed?

rainfall_df = weather_df.select('Rainfall').dropna()
plt.hist(rainfall_df.toPandas(), bins=50)
```

```
plt.title('Distribution of Rainfall')
plt.xlabel('Rainfall (mm)')
plt.ylabel('Frequency')
plt.show()
```



Testing some basic assumptions about the data. Think of this as a "quick and dirty" exploration of the dataset.

In [ ]:
```
missing = weather_df.toPandas().isnull().sum()
print(f"Missing values:\n{missing.sort_values(ascending=False)}")
```

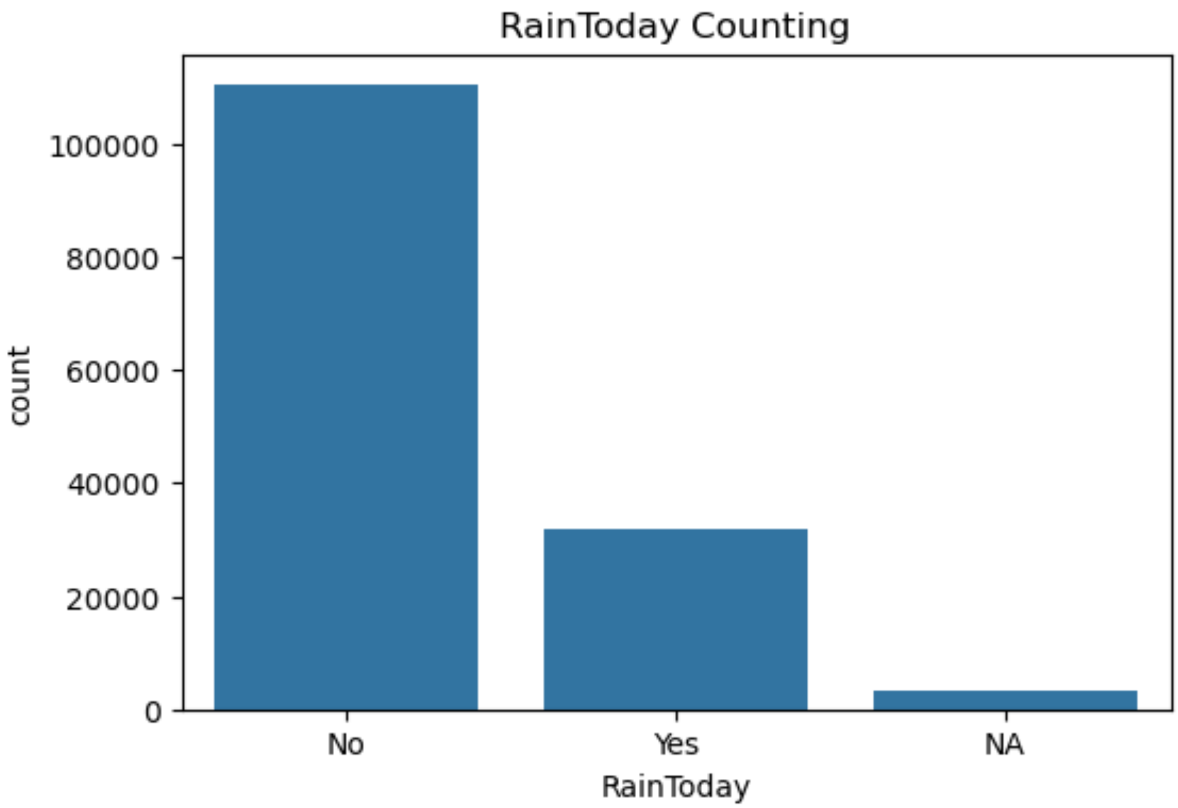
```
Missing values:
Sunshine         69835
Evaporation      62790
Cloud3pm         59358
Cloud9am         55888
Pressure9am      15065
Pressure3pm      15028
WindGustSpeed    10263
Humidity3pm       4507
Temp3pm           3609
Rainfall          3261
WindSpeed3pm      3062
Humidity9am       2654
Temp9am           1767
WindSpeed9am      1767
MinTemp           1485
MaxTemp           1261
RainToday            0
Date                 0
Location             0
WindDir3pm           0
WindDir9am           0
WindGustDir          0
RainTomorrow         0
dtype: int64
```

In [ ]:

```
# rain_tomorrow_df = weather_df.select('RainTomorrow') \
#                               .filter(weather_df.RainTomorrow != 'NA
#                               .dropna().toPandas()

rain_today_df = weather_df.select('RainToday') \
                              .dropna().toPandas()

# Target variable analysis
plt.figure(figsize=(6, 4))
sns.countplot(x='RainToday', data=rain_today_df)
plt.title('RainToday Counting')
plt.show()
```

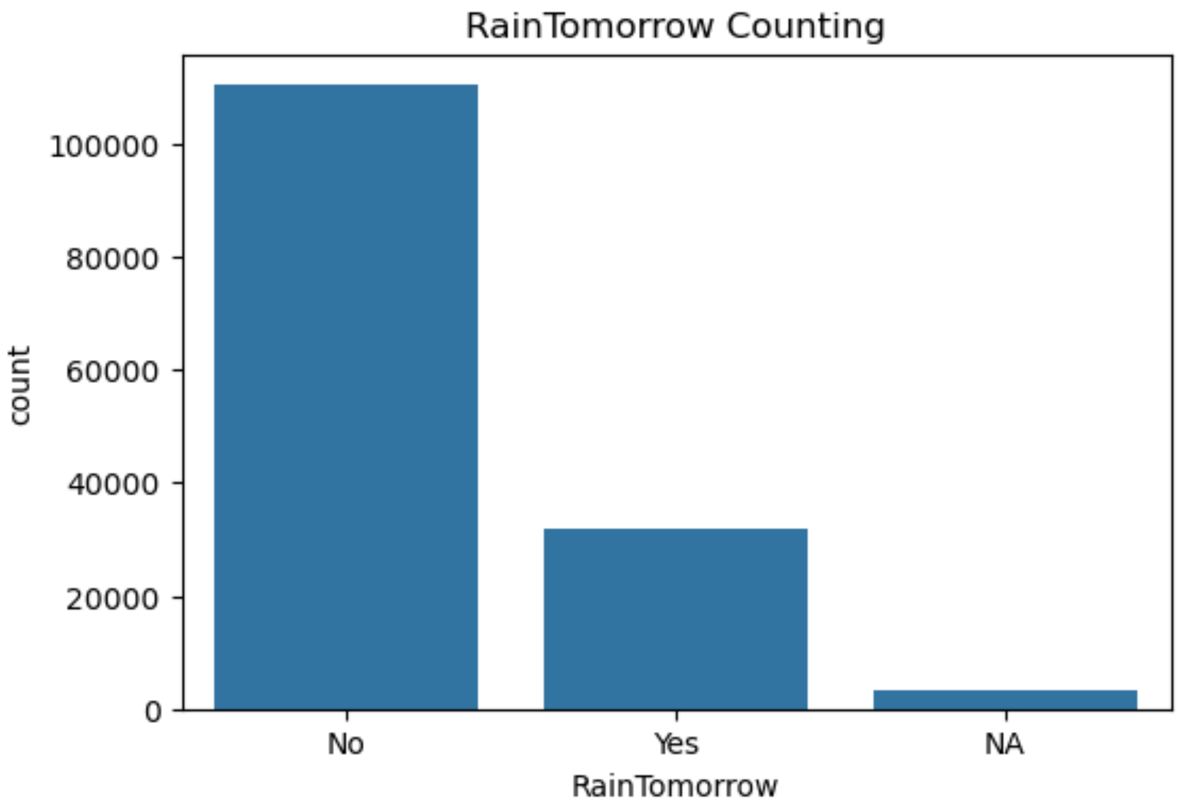


In [ ]:

```
# rain_tomorrow_df = weather_df.select('RainTomorrow') \
#                                 .filter(weather_df.RainTomorrow != 'NA
#                                 .dropna().toPandas()

rain_tomorrow_df = weather_df.select('RainTomorrow') \
                              .dropna().toPandas()

# Target variable analysis
plt.figure(figsize=(6, 4))
sns.countplot(x='RainTomorrow', data=rain_tomorrow_df)
plt.title('RainTomorrow Counting')
plt.show()
```



Let's remove the rows with RainToday or RainTomorrow are 'NA'.

In [ ]:
```
weather_df = weather_df.filter((weather_df.RainToday != 'NA') & (weath
weather_df.cache()
weather_df.createOrReplaceTempView("weather_aus")

spark.sql("SELECT COUNT(*) FROM weather_aus WHERE RainToday = 'NA' OR
```

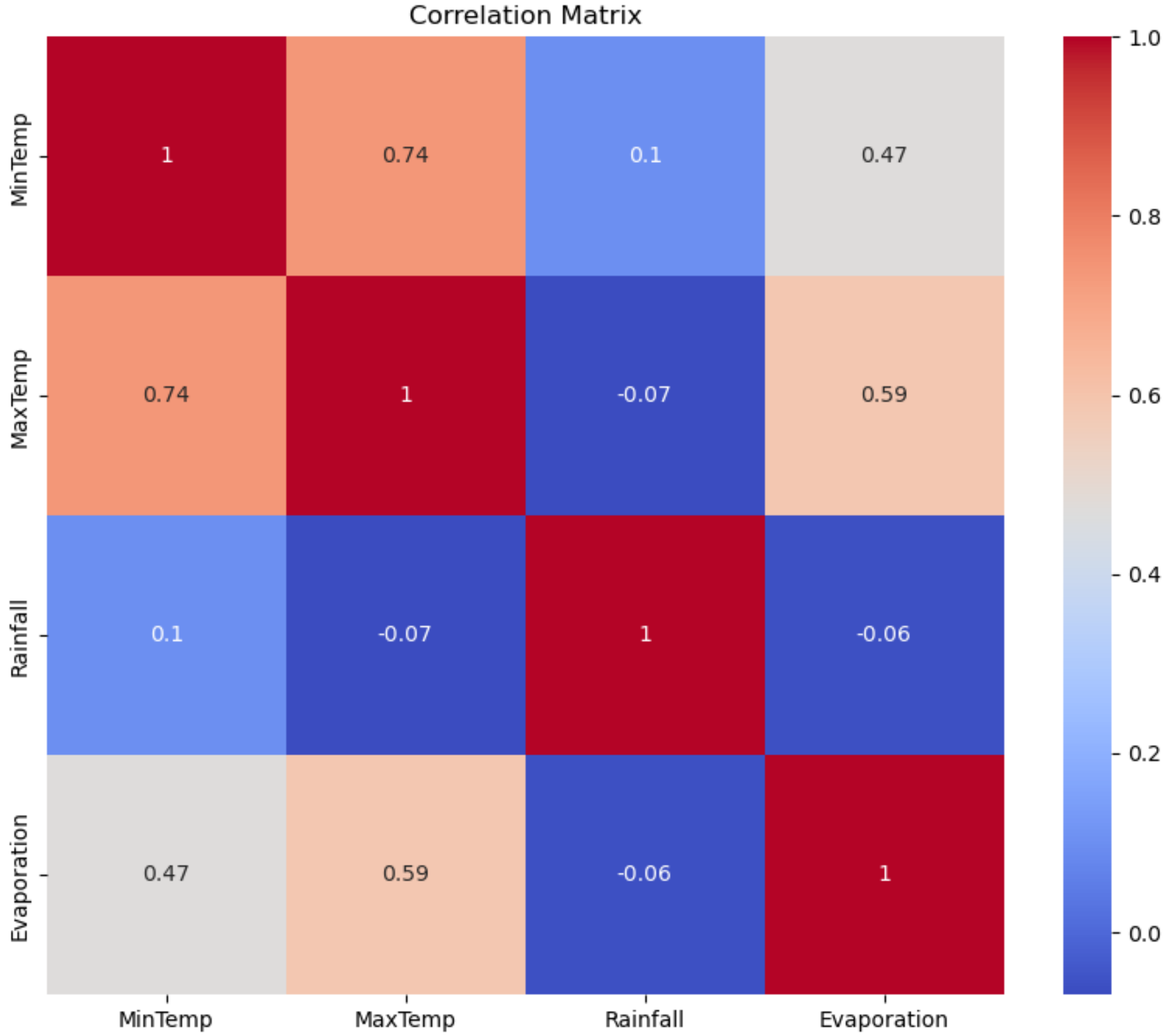
```
+--------+
|count(1)|
+--------+
|       0|
+--------+
```

Let's check the correlation between variables in the dataset.

In [ ]:
```
correlation_features = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation
corr_df = weather_df.toPandas()[correlation_features]
correlation_matrix = corr_df.corr().round(2)
print(correlation_matrix)

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=Tr
plt.title('Correlation Matrix')
plt.show()
```

```
             MinTemp  MaxTemp  Rainfall  Evaporation
MinTemp         1.00     0.74      0.10         0.47
MaxTemp         0.74     1.00     -0.07         0.59
Rainfall        0.10    -0.07      1.00        -0.06
Evaporation     0.47     0.59     -0.06         1.00
```



Couple of assumptions about the dataset

1. 'Rainfall' histogram shows that the dataset is not normally distributed. There is a large peak at 0, indicating that many days have no rainfall. Is Australia dry season with less rain?
2. Based on the heatmap, we can see a moderate negative correlation between Rainfall and Evaporation, as well as between Rainfall and MaxTemp.

These are just a few basic assumptions and explorations to get started with the dataset.

Machine Learning

We are attempting to predict whether it will rain today and whether it will rain tomorrow. Our first attempt is with Logistic Regression, and the results are fairly good - 83.35% accuracy for today's prediction and 84.91% accuracy for tomorrow's prediction.

In [ ]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler

# Index the labels (RainToday and RainTomorrow)
# Interestingly, using handleInvalid="keep" resulted in an additional
#   each column only contains distinct values of "Yes" and "No"
today_label = StringIndexer(inputCol="RainToday", outputCol="today_lab
tomorrow_label = StringIndexer(inputCol="RainTomorrow", outputCol="tom

# Numerical features to keep
# Omit 'Rainfall' column or we'll always perfectly predict whether it'
numerical_cols = [
    'MinTemp', 'MaxTemp', 'WindGustSpeed',
    'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
    'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
    'Temp9am', 'Temp3pm',
]

# The following columns are null in at least 1/3 of the entries. Omit
weather_df = weather_df.drop('Location', 'WindGustDir', 'WindDir9am',

# In remaining columns, drop any remaining null entries. This leaves u
weather_df = weather_df.dropna()
assembler = VectorAssembler(inputCols=numerical_cols, outputCol="featu

# Split into training and test sets
train_data, test_data = weather_df.randomSplit([0.8, 0.2], seed=42)
```

Now we have the data all ready to be plugged in to various ML models. We're going to start with LogisticRegression.

We can use the train and test data to validate the accuracy of the models, and find a good match for our weather predictions.

In [ ]:
```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Logistic Regression models
lr_today = LogisticRegression(featuresCol="features", labelCol="today_
lr_tomorrow = LogisticRegression(featuresCol="features", labelCol="tom

# Build pipelines
pipeline_today = Pipeline(stages=[today_label, assembler, lr_today])
pipeline_tomorrow = Pipeline(stages=[tomorrow_label, assembler, lr_tom
```

```
# Train LR models
model_lr_today = pipeline_today.fit(train_data)
model_lr_tomorrow = pipeline_tomorrow.fit(train_data)

# Predictions on test set
predictions_lr_today = model_lr_today.transform(test_data)
predictions_lr_tomorrow = model_lr_tomorrow.transform(test_data)

# Evaluate accuracies
evaluator_today = MulticlassClassificationEvaluator(labelCol="today_la
evaluator_tomorrow = MulticlassClassificationEvaluator(labelCol="tomor

accuracy_today = evaluator_today.evaluate(predictions_lr_today)
accuracy_tomorrow = evaluator_tomorrow.evaluate(predictions_lr_tomorro
print(f"Accuracy for today, Logistic Regression = {accuracy_today:.2%}
print(f"Accuracy for tomorrow, Logistic Regression = {accuracy_tomorro
```

```
Accuracy for today, Logistic Regression = 83.35%
Accuracy for tomorrow, Logistic Regression = 84.91%
```

Tuning with CrossValidator

Below is code we used to try tuning the LR model. It actually wound up giving slightly worse results:

- Accuracy for today, Logistic Regression = 0.8278
- Accuracy for tomorrow, Logistic Regression = 0.8462

Additionally, the time to train the models was significantly longer. Leaving the code block to reference later, but commenting it out so we don't have to wait for it.

In [ ]:
```
# from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# # Make parameter grids
# paramGrid_today = ParamGridBuilder() \
#     .addGrid(lr_today.regParam, [0.01, 0.1, 1.0]) \
#     .addGrid(lr_today.elasticNetParam, [0.0, 0.5, 1.0]) \
#     .addGrid(lr_today.maxIter, [10, 50]) \
#     .build()
# paramGrid_tomorrow = ParamGridBuilder() \
#     .addGrid(lr_tomorrow.regParam, [0.01, 0.1, 1.0]) \
#     .addGrid(lr_tomorrow.elasticNetParam, [0.0, 0.5, 1.0]) \
#     .addGrid(lr_tomorrow.maxIter, [10, 50]) \
#     .build()

# # Set up CrossValidator
# cv_today = CrossValidator(
```

```
#     estimator=lr_today,
#     estimatorParamMaps=paramGrid_today,
#     evaluator=evaluator_today,
#     numFolds=3,
#     parallelism=2
# )
# cv_tomorrow = CrossValidator(
#     estimator=lr_tomorrow,
#     estimatorParamMaps=paramGrid_tomorrow,
#     evaluator=evaluator_tomorrow,
#     numFolds=3,
#     parallelism=2
# )

# # Re-do pipelines with CV
# pipeline_today = Pipeline(stages=[today_label, assembler, cv_today])
# pipeline_tomorrow = Pipeline(stages=[tomorrow_label, assembler, cv_t

# # Fit with cross-validation
# cv_model_today = pipeline_today.fit(train_data)
# cv_model_tomorrow = pipeline_tomorrow.fit(train_data)

# # Predictions on test data
# predictions_lr_today = cv_model_today.transform(test_data)
# predictions_lr_tomorrow = cv_model_tomorrow.transform(test_data)

# # Re-evaluate accuracy
# accuracy_today = evaluator_today.evaluate(predictions_lr_today)
# accuracy_tomorrow = evaluator_tomorrow.evaluate(predictions_lr_tomor
# print(f"Accuracy for today, Logistic Regression = {accuracy_today:.2
# print(f"Accuracy for tomorrow, Logistic Regression = {accuracy_tomor
```

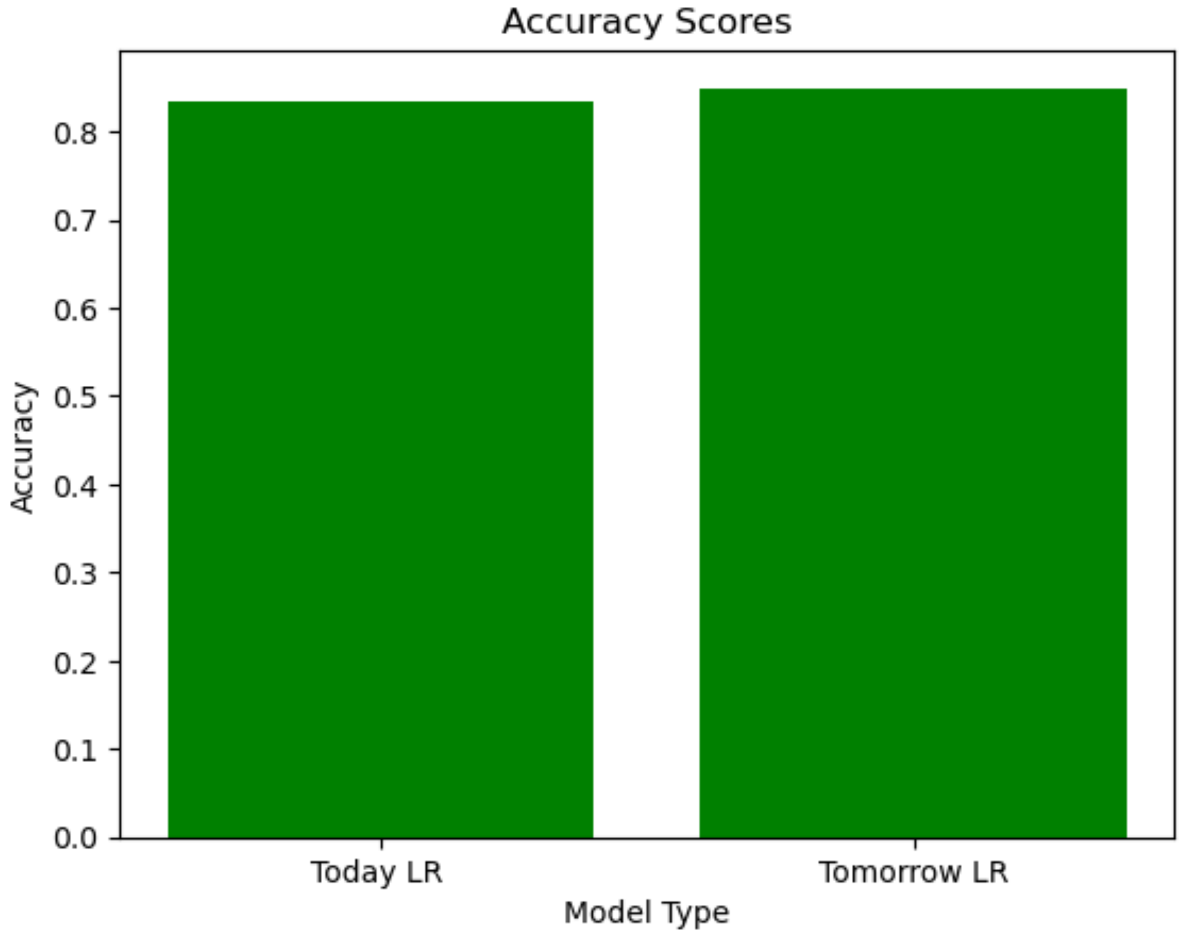
Model Performance Visualization and Evaluation

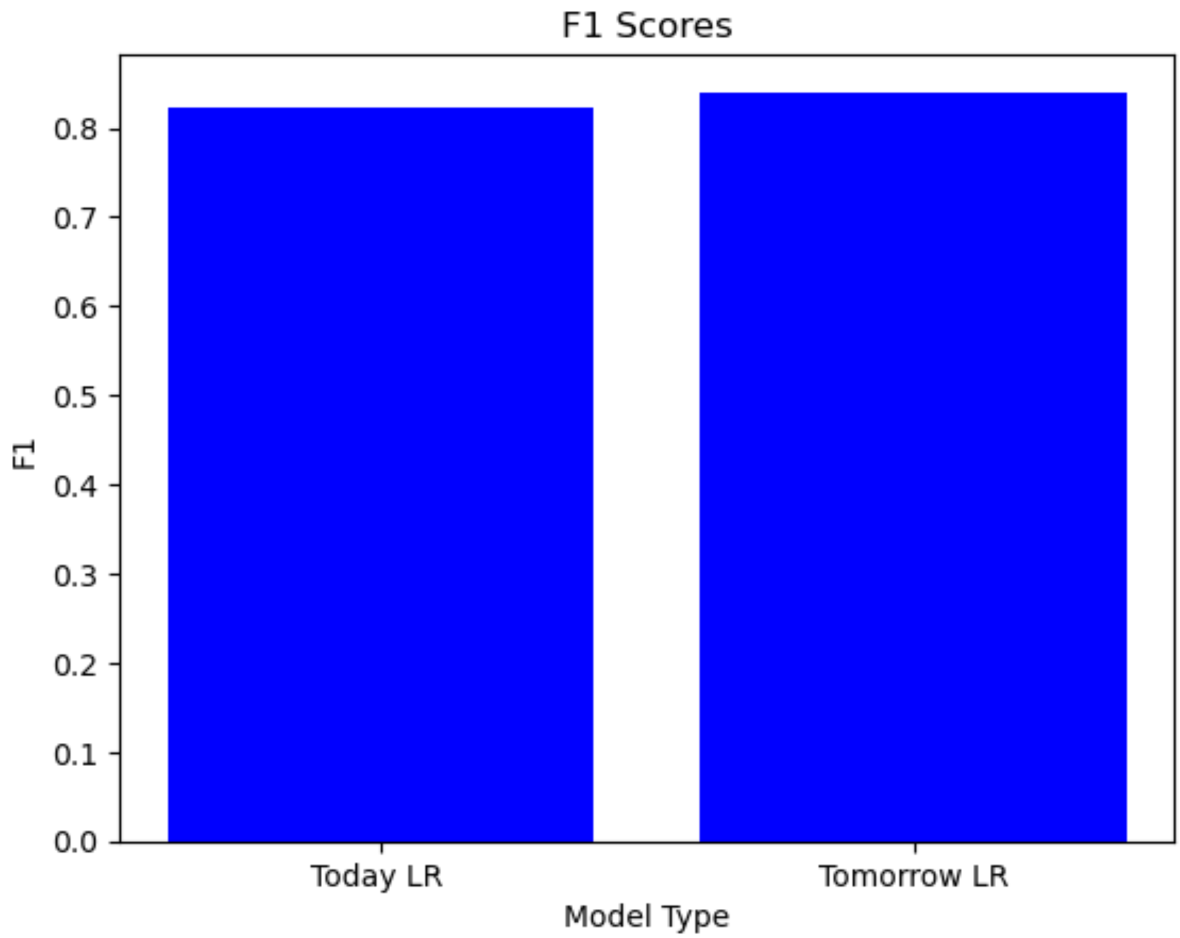
In [15]:

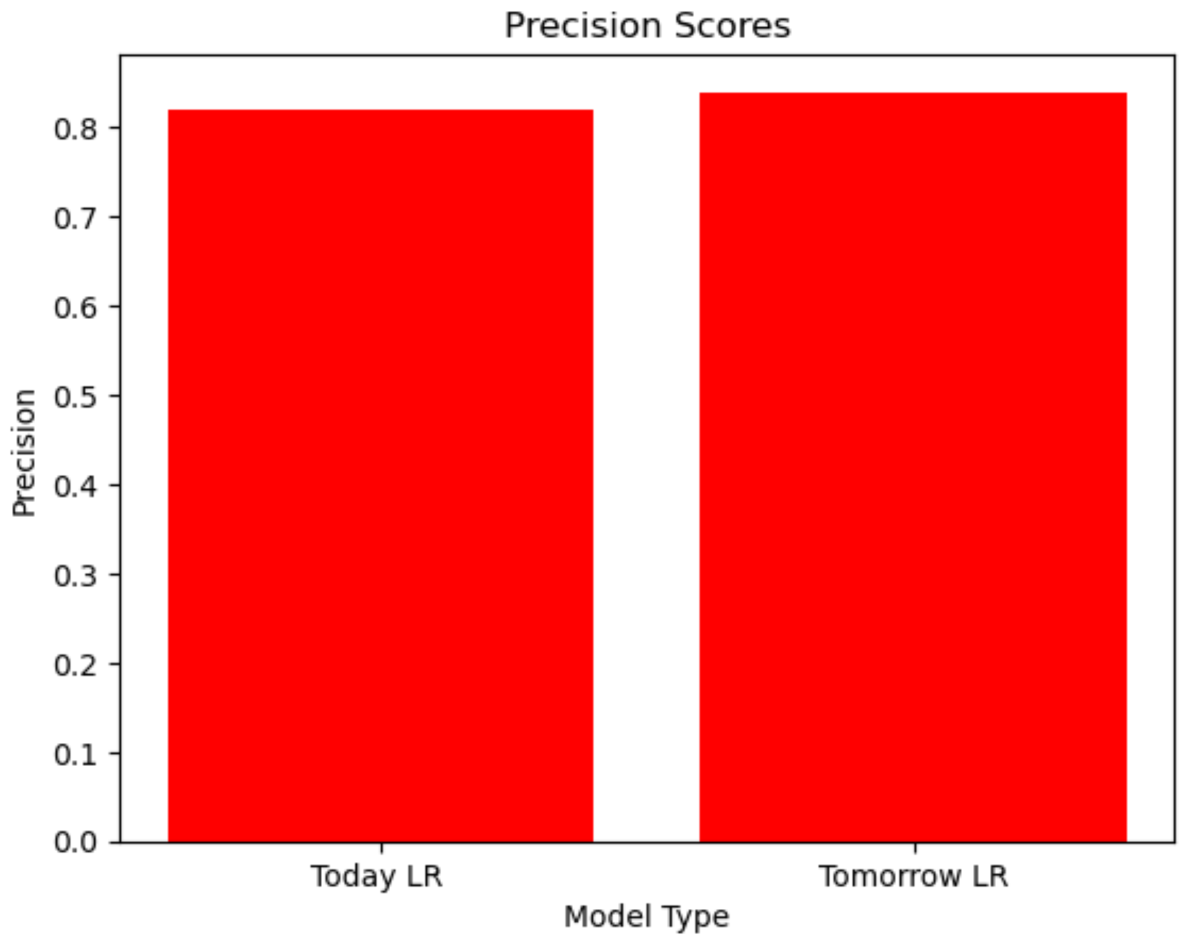
```
todayF1ScoreEvaluator = MulticlassClassificationEvaluator(labelCol="to
tomorrowF1ScoreEvaluator = MulticlassClassificationEvaluator(labelCol=
lrTodayF1Score = todayF1ScoreEvaluator.evaluate(predictions_lr_today)
lrTomorrowF1Score = tomorrowF1ScoreEvaluator.evaluate(predictions_lr_t
todayPrecisionScoreEvaluator = MulticlassClassificationEvaluator(label
tomorrowPrecisionScoreEvaluator = MulticlassClassificationEvaluator(la
lrTodayPrecisionScore = todayPrecisionScoreEvaluator.evaluate(predicti
lrTomorrowPrecisionScore = tomorrowPrecisionScoreEvaluator.evaluate(pr

def displayBarChart(todayMetric , tomorrowMetric , name , barColor):
    plt.bar(["Today LR " , "Tomorrow LR"] , [todayMetric , tomorrowMet
    plt.xlabel("Model Type")
    plt.ylabel(name)
    plt.title(f"{name} Scores")
    plt.show()
```

```
displayBarChart(accuracy_today , accuracy_tomorrow , "Accuracy" , "gre
displayBarChart(lrTodayF1Score , lrTomorrowF1Score , "F1" , "blue")
displayBarChart(lrTodayPrecisionScore , lrTomorrowPrecisionScore , "Pr
```




F1 Scores
F1
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Today LR
Tomorrow LR
Model Type



We see from these bar charts that our linear regression model predicting rain for tomorrow is always achieving slightly higher performance.

In [16]:
```
# residual line
#doesn't show good results because our models are predicting a yes or

def showResidualLine(prediction , labelName , plotName):
    LRResidualsDF = prediction.withColumn("residual" , prediction[labe
    LRResidualsDFAsPandas = LRResidualsDF.select("prediction" , "resid
    plt.scatter(LRResidualsDFAsPandas['prediction'], LRResidualsDFAsPa
    plt.axhline(y=0, color='r', linestyle='--') # Add a horizontal lin
    plt.xlabel("Predicted Values")
    plt.ylabel("Residuals")
    plt.title(plotName)
    plt.show()

#showResidualLine(predictions_lr_today , "today_label" , "Today LR Res
#showResidualLine(predictions_lr_tomorrow , "tomorrow_label" , "Tomorr
```

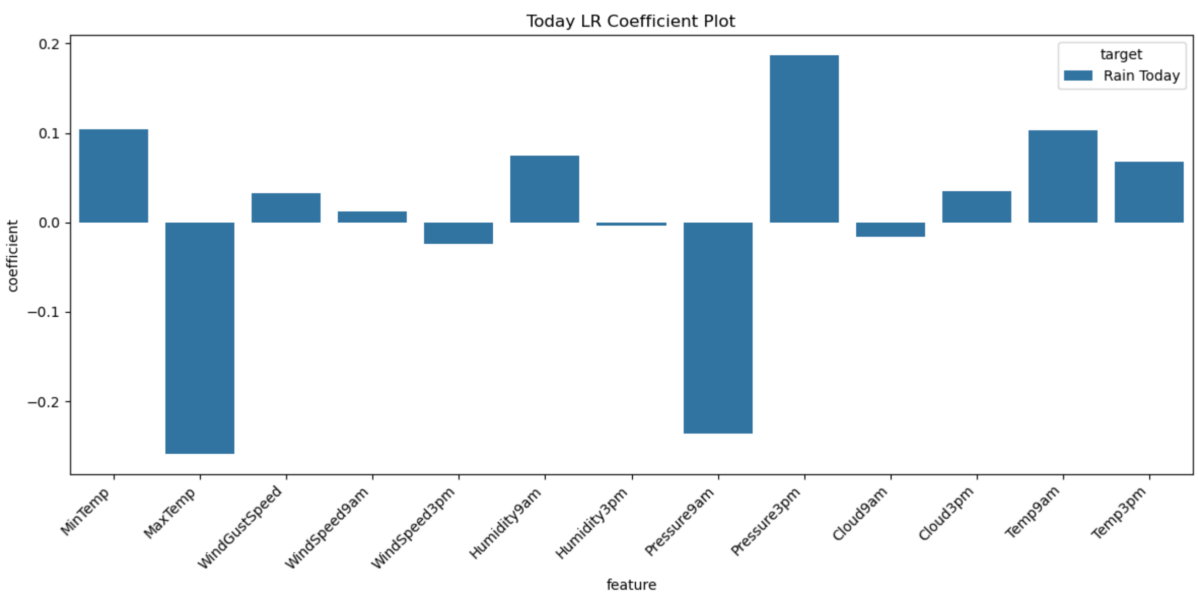
In [18]:
```
#Coefficient plots
import numpy as np
import seaborn as sns
import pandas as pd
```

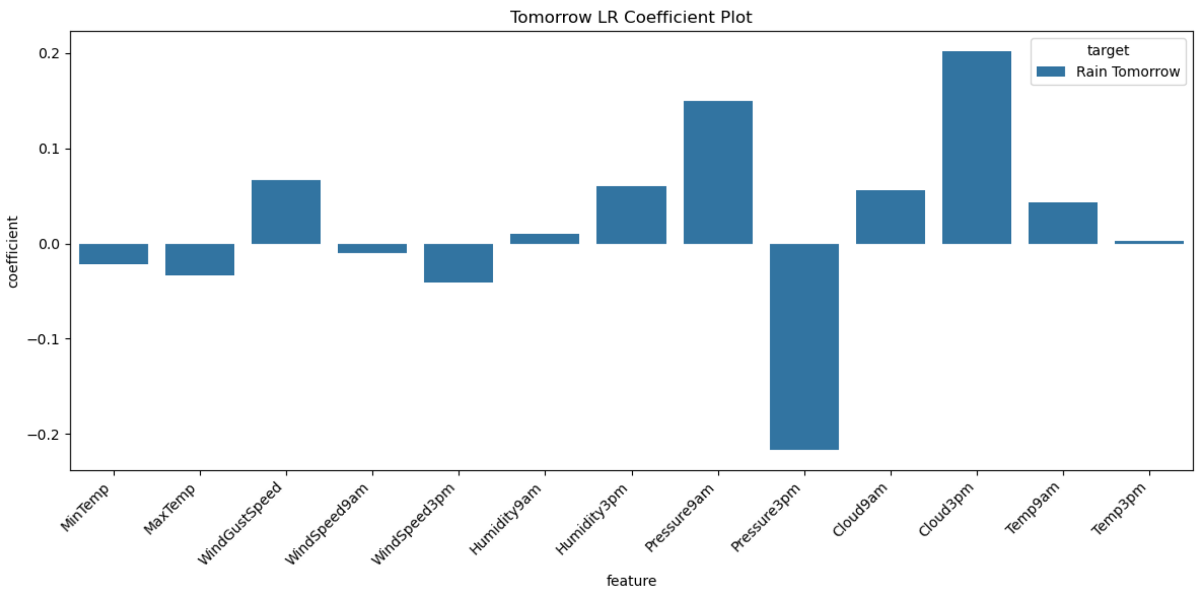
```
#implementation based on: https://chatgpt.com/share/688a7a74-2eac-8008
def showCoefficientsMatrix(lrPipeline , plotName):
    #lrPipeline is a pipeline model. We need to get the original linea
    lrModel = lrPipeline.stages[-1]
    coefficients = lrModel.coefficientMatrix.toArray()
    intercept = lrModel.interceptVector
    features = assembler.getInputCols()
    coefficientArray = np.array(coefficients)
    responseLabels = ['Rain ' + plotName.split(" ")[0]]
    plotDF = pd.DataFrame(coefficientArray.T , columns = responseLabel
    plotDF['feature'] = features
    plotDF = plotDF.melt(id_vars = 'feature' , var_name = 'target' , v

    plt.figure(figsize = (12 , 6))
    sns.barplot(data = plotDF , x = 'feature' , y = 'coefficient' , hu

    plt.xticks(rotation = 45 , ha = 'right')
    plt.title(plotName)
    plt.tight_layout()
    plt.show()


showCoefficientsMatrix(model_lr_today , "Today LR Coefficient Plot")
print()
showCoefficientsMatrix(model_lr_tomorrow , "Tomorrow LR Coefficient Pl
```





Conclusion

This project successfully developed and evaluated machine learning models to predict rainfall in Australia for both the current day and the following day using historical weather data. The primary objective was to create a reliable predictive tool to aid in planning and risk mitigation across various sectors.

Model Performance and Key Findings

The Logistic Regression model proved to be an effective and interpretable choice for this binary classification task.

- The model predicting rain for today achieved an accuracy of 83.35%. The most influential predictors for rain were higher minimum temperatures and afternoon pressure readings.

- The model predicting rain for tomorrow performed slightly better, with an accuracy of 84.91%. Key positive indicators for future rain included morning pressure and afternoon cloud cover, whereas higher afternoon pressure was a strong indicator of no rain the next day.

Interestingly, while the baseline Logistic Regression model performed well, a cross-validated version yielded poor results, suggesting that the standard model may have generalized better to the test set in this specific instance, though this warrants further investigation.

Limitations

A significant challenge in this analysis was the volume of missing data for crucial features, including Sunshine, Evaporation, and cloud cover at both 9 am and 3 pm (Cloud9am, Cloud3pm). These data gaps necessitated imputation, which may have introduced some bias or reduced the potential predictive power of the models.

Practical Implications & Future Work

The developed models demonstrate significant practical value. The predictions can be instrumental for:

- Agriculture: Optimizing irrigation schedules and planning crop planting.
- Emergency Services: Assessing daily fire danger and issuing timely warnings.
- Construction & Events: Scheduling outdoor activities and projects to avoid weather-related disruptions.

Looking ahead, several avenues could enhance this project's predictive capabilities. The logical next step is to explore more complex, non-linear models. Experimenting with a neural network or ensemble methods like Gradient Boosting could capture more intricate patterns in the data. Furthermore, acquiring a more complete dataset would be paramount to improving model accuracy and reliability.