# Rainfall Weather forecasting Project

In this Project, I will be implementing a predictive model on Rain Dataset to predict whether or not it will rain tomorrow in Australia. The Dataset contains about 10 years of daily weather observations of different locations in Australia. By the end of this article, you will be able to build a predictive model

Design a predictive model with the use of machine learning algorithms to forecast whether or not it will rain tomorrow in Australia.

## Importing Libraries

The first step in any Data Analysis step is importing necessary libraries.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

## Loading the Dataset

Dataset can be loaded using a method read_csv

In [2]:
```
df=pd.read_csv(r"https://raw.githubusercontent.com/dsrscientist/dataset3/main/weatherAUS.csv")
df.head()
```

Out[2]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSpeed | WindDir9am | ... | Humidity9am | Humidity3pm | Pressure9am | Pressure3pm | Cloud9am | Cloud3pm | Temp9am | Temp3pm | RainToday | RainTomorrow |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2008-12-01 | Albury | 13.4 | 22.9 | 0.6 | NaN | NaN | W | 44.0 | W | ... | 71.0 | 22.0 | 1007.7 | 1007.1 | 8.0 | NaN | 16.9 | 21.8 | No | No |
| 1 | 2008-12-02 | Albury | 7.4 | 25.1 | 0.0 | NaN | NaN | WNW | 44.0 | NNW | ... | 44.0 | 25.0 | 1010.6 | 1007.8 | NaN | NaN | 17.2 | 24.3 | No | No |
| 2 | 2008-12-03 | Albury | 12.9 | 25.7 | 0.0 | NaN | NaN | WSW | 46.0 | W | ... | 38.0 | 30.0 | 1007.6 | 1008.7 | NaN | 2.0 | 21.0 | 23.2 | No | No |
| 3 | 2008-12-04 | Albury | 9.2 | 28.0 | 0.0 | NaN | NaN | NE | 24.0 | SE | ... | 45.0 | 16.0 | 1017.6 | 1012.8 | NaN | NaN | 18.1 | 26.5 | No | No |
| 4 | 2008-12-05 | Albury | 17.5 | 32.3 | 1.0 | NaN | NaN | W | 41.0 | ENE | ... | 82.0 | 33.0 | 1010.8 | 1006.0 | 7.0 | 8.0 | 17.8 | 29.7 | No | No |

5 rows × 23 columns

## Checking the Dimensions of Dataset

The shape property is used to find the dimensions of the dataset

In [15]:
```
df.shape
```

Out[15]:
```
(8425, 23)
```

## Data Preprocessing

Real-world data is often messy, incomplete, unstructured, inconsistent, redundant, sprinkled with wacky values. So, without deploying any Data Preprocessing techniques, it is almost impossible to gain insights from raw data.

What exactly is Data Preprocessing?

Data preprocessing is a process of converting raw data to a suitable format to extract insights. It is the first and foremost step in the Data Science life cycle. Data Preprocessing makes sure that data is clean, organize and read-to-feed to the Machine Learning model.

## In detailed summary of a Dataset

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8425 entries, 0 to 8424
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           8425 non-null   object
 1   Location       8425 non-null   object
 2   MinTemp        8350 non-null   float64
 3   MaxTemp        8365 non-null   float64
 4   Rainfall       8185 non-null   float64
 5   Evaporation    4913 non-null   float64
 6   Sunshine       4431 non-null   float64
 7   WindGustDir    7434 non-null   object
 8   WindGustSpeed  7434 non-null   float64
 9   WindDir9am     7596 non-null   object
 10  WindDir3pm     8117 non-null   object
 11  WindSpeed9am   8349 non-null   float64
 12  WindSpeed3pm   8318 non-null   float64
 13  Humidity9am    8366 non-null   float64
 14  Humidity3pm    8323 non-null   float64
 15  Pressure9am    7116 non-null   float64
 16  Pressure3pm    7113 non-null   float64
 17  Cloud9am       6004 non-null   float64
 18  Cloud3pm       5970 non-null   float64
 19  Temp9am        8369 non-null   float64
 20  Temp3pm        8329 non-null   float64
 21  RainToday      8185 non-null   object
 22  RainTomorrow   8186 non-null   object
dtypes: float64(16), object(7)
memory usage: 1.5+ MB
```

Dataset has two data types: float64, object

Except for the Date, Location columns, every column has missing values. Let's generate descriptive statistics for the dataset using the function describe() in pandas.

Descriptive Statistics: It is used to summarize and describe the features of data in a meaningful way to extract insights. It uses two types of statistic to describe or summarize data:

In [18]:
```
df.describe()
```

Out[18]:

| | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustSpeed | WindSpeed9am | WindSpeed3pm | Humidity9am | Humidity3pm | Pressure9am | Pressure3pm | Cloud9am | Cloud3pm | Temp9am | Temp3pm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 8350.000000 | 8365.000000 | 8185.000000 | 4913.000000 | 4431.000000 | 7434.000000 | 8349.000000 | 8318.000000 | 8366.000000 | 8323.000000 | 7116.000000 | 7113.000000 | 6004.000000 | 5970.000000 | 8369.000000 | 8329.000000 |
| mean | 13.193305 | 23.859976 | 2.805913 | 5.389395 | 7.632205 | 40.174469 | 13.847646 | 18.533662 | 67.822496 | 51.249790 | 1017.640233 | 1015.236075 | 4.566622 | 4.503183 | 17.762015 | 22.442934 |
| std | 5.403596 | 6.136408 | 10.459379 | 5.044484 | 3.896235 | 14.665721 | 10.174579 | 9.766986 | 16.833283 | 18.423774 | 6.828699 | 6.766681 | 2.877658 | 2.731659 | 5.627035 | 5.980020 |
| min | -2.000000 | 8.200000 | 0.000000 | 0.000000 | 0.000000 | 7.000000 | 0.000000 | 0.000000 | 10.000000 | 6.000000 | 989.800000 | 982.900000 | 0.000000 | 0.000000 | 1.900000 | 7.300000 |
| 25% | 9.200000 | 19.300000 | 0.000000 | 2.600000 | 4.750000 | 30.000000 | 6.000000 | 11.000000 | 56.000000 | 39.000000 | 1013.000000 | 1010.400000 | 1.000000 | 2.000000 | 13.800000 | 18.000000 |
| 50% | 13.300000 | 23.300000 | 0.000000 | 4.600000 | 8.700000 | 39.000000 | 13.000000 | 19.000000 | 68.000000 | 51.000000 | 1017.700000 | 1015.300000 | 5.000000 | 5.000000 | 17.800000 | 21.900000 |
| 75% | 17.400000 | 28.000000 | 1.000000 | 7.000000 | 10.700000 | 50.000000 | 20.000000 | 24.000000 | 80.000000 | 63.000000 | 1022.300000 | 1019.800000 | 7.000000 | 7.000000 | 21.900000 | 26.400000 |
| max | 28.500000 | 45.500000 | 371.000000 | 145.000000 | 13.900000 | 107.000000 | 63.000000 | 83.000000 | 100.000000 | 99.000000 | 1039.000000 | 1036.000000 | 8.000000 | 8.000000 | 39.400000 | 44.100000 |

In [20]:
```
print(df.describe(include=[object]))
```

```
              Date   Location WindGustDir WindDir9am WindDir3pm RainToday  \
count         8425       8425        7434       7596       8117      8185
unique        3004         12          16         16         16         2
top     2011-02-01  Melbourne           N          N         SE        No
freq             5       1622         713        906        813      6195

       RainTomorrow
count          8186
unique            2
top              No
freq           6195
```

In [6]:
```
df.tail(6)
```

Out[6]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSpeed | WindDir9am | ... | Humidity9am | Humidity3pm | Pressure9am | Pressure3pm | Cloud9am | Cloud3pm | Temp9am | Temp3pm | RainToday | RainTomorrow |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8419 | 2017-06-20 | Uluru | 3.5 | 21.8 | 0.0 | NaN | NaN | E | 31.0 | ESE | ... | 59.0 | 27.0 | 1024.7 | 1021.2 | NaN | NaN | 9.4 | 20.9 | No | No |
| 8420 | 2017-06-21 | Uluru | 2.8 | 23.4 | 0.0 | NaN | NaN | E | 31.0 | SE | ... | 51.0 | 24.0 | 1024.6 | 1020.3 | NaN | NaN | 10.1 | 22.4 | No | No |
| 8421 | 2017-06-22 | Uluru | 3.6 | 25.3 | 0.0 | NaN | NaN | NNW | 22.0 | SE | ... | 56.0 | 21.0 | 1023.5 | 1019.1 | NaN | NaN | 10.9 | 24.5 | No | No |
| 8422 | 2017-06-23 | Uluru | 5.4 | 26.9 | 0.0 | NaN | NaN | N | 37.0 | SE | ... | 53.0 | 24.0 | 1021.0 | 1016.8 | NaN | NaN | 12.5 | 26.1 | No | No |
| 8423 | 2017-06-24 | Uluru | 7.8 | 27.0 | 0.0 | NaN | NaN | SE | 28.0 | SSE | ... | 51.0 | 24.0 | 1019.4 | 1016.5 | 3.0 | 2.0 | 15.1 | 26.0 | No | No |
| 8424 | 2017-06-25 | Uluru | 14.9 | NaN | 0.0 | NaN | NaN | NaN | NaN | ESE | ... | 62.0 | 36.0 | 1020.2 | 1017.9 | 8.0 | 8.0 | 15.0 | 20.9 | No | NaN |

6 rows × 23 columns

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
Date                0
Location            0
MinTemp            75
MaxTemp            60
Rainfall          240
Evaporation      3512
Sunshine         3994
WindGustDir       991
WindGustSpeed     991
WindDir9am        829
WindDir3pm        308
WindSpeed9am       76
WindSpeed3pm      107
Humidity9am        59
Humidity3pm       102
Pressure9am      1309
Pressure3pm      1312
Cloud9am         2421
Cloud3pm         2455
Temp9am            56
Temp3pm            96
RainToday         240
RainTomorrow      239
dtype: int64
```

In [11]:
```
df.isnull().sum().sum()
```

Out[11]:
```
19472
```

In [14]:
```
df.columns
```

Out[14]:
```
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')
```

## Finding Categorical and Numerical Features in a Data set

Categorical features in Dataset:

In [21]:
```
categorical_features = [column_name for column_name in df.columns if df[column_name].dtype == 'O']
print("Number of Categorical Features: {}".format(len(categorical_features)))
print("Categorical Features: ",categorical_features)
```

```
Number of Categorical Features: 7
Categorical Features:  ['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
```

Numerical Features in Dataset

In [22]:
```
numerical_features = [column_name for column_name in df.columns if df[column_name].dtype != 'O']
print("Number of Numerical Features: {}".format(len(numerical_features)))
print("Numerical Features: ",numerical_features)
```

```
Number of Numerical Features: 16
Numerical Features:  ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'T
emp3pm']
```

Many Machine learning algorithms like Linear Regression, Logistic Regression, k-nearest neighbors, etc. can handle only numerical data, so encoding categorical data to numeric becomes a necessary step. But before jumping into encoding, check the cardinality of each categorical feature.

This high cardinality feature poses many serious problems like it will increase the number of dimensions of data when that feature is encoded. This is not good for the model.

In [23]:
```
for each_feature in categorical_features:
   unique_values = len(df[each_feature].unique())
   print("Cardinality(no. of unique values) of {} are: {}".format(each_feature, unique_values))
```

```
Cardinality(no. of unique values) of Date are: 3004
Cardinality(no. of unique values) of Location are: 12
Cardinality(no. of unique values) of WindGustDir are: 17
Cardinality(no. of unique values) of WindDir9am are: 17
Cardinality(no. of unique values) of WindDir3pm are: 17
Cardinality(no. of unique values) of RainToday are: 3
Cardinality(no. of unique values) of RainTomorrow are: 3
```

Date column has high cardinality which poses several problems to the model in terms of efficiency and also dimensions of data increase when encoded to numerical data.

In [38]:
```
df.head()
```

Out[38]:

| | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm | ... | Pressure3pm | Cloud9am | Cloud3pm | Temp9am | Temp3pm | RainToday | RainTomorrow | year | month | day |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Albury | 13.4 | 22.9 | 0.6 | NaN | NaN | W | 44.0 | W | WNW | ... | 1007.1 | 8.0 | NaN | 16.9 | 21.8 | No | No | 2008 | 12 | 1 |
| 1 | Albury | 7.4 | 25.1 | 0.0 | NaN | NaN | WNW | 44.0 | NNW | WSW | ... | 1007.8 | NaN | NaN | 17.2 | 24.3 | No | No | 2008 | 12 | 2 |
| 2 | Albury | 12.9 | 25.7 | 0.0 | NaN | NaN | WSW | 46.0 | W | WSW | ... | 1008.7 | NaN | 2.0 | 21.0 | 23.2 | No | No | 2008 | 12 | 3 |
| 3 | Albury | 9.2 | 28.0 | 0.0 | NaN | NaN | NE | 24.0 | SE | E | ... | 1012.8 | NaN | NaN | 18.1 | 26.5 | No | No | 2008 | 12 | 4 |
| 4 | Albury | 17.5 | 32.3 | 1.0 | NaN | NaN | W | 41.0 | ENE | NW | ... | 1006.0 | 7.0 | 8.0 | 17.8 | 29.7 | No | No | 2008 | 12 | 5 |

5 rows × 25 columns

## Handling Missing Values

Machine learning algorithms can't handle missing values and cause problems. So they need to be addressed in the first place. There are many techniques to identify and impute missing values.

If a dataset contains missing values and loaded using pandas, then missing values get replaced with NaN(Not a Number) values. These NaN values can be identified using methods like isna() or isnull() and they can be imputed using fillna(). This process is known as Missing Data Imputation.

Handling Missing values in Categorical Features

In [39]:
```
categorical_features = [column_name for column_name in df.columns if df[column_name].dtype == 'O']
df[categorical_features].isnull().sum()
```

Out[39]:
```
Location          0
WindGustDir     991
WindDir9am      829
WindDir3pm      308
RainToday       240
RainTomorrow    239
dtype: int64
```

mputing the missing values in categorical features using the most frequent value which is mode

In [42]:
```
categorical_features_with_null = [feature for feature in categorical_features if df[feature].isnull().sum()]
for each_feature in categorical_features_with_null:
    mode_val = df[each_feature].mode()[0]
    df[each_feature].fillna(mode_val,inplace=True)
```

Handling Missing values in Numerical features

In [44]:
```
numerical_features = [column_name for column_name in df.columns if df[column_name].dtype != 'O']
df[numerical_features].isnull().sum()
```

Out[44]:
```
MinTemp            75
MaxTemp            60
Rainfall          240
Evaporation      3512
Sunshine         3994
WindGustSpeed     991
WindSpeed9am       76
WindSpeed3pm      107
Humidity9am        59
Humidity3pm       102
Pressure9am      1309
Pressure3pm      1312
Cloud9am         2421
Cloud3pm         2455
Temp9am            56
Temp3pm            96
year                0
month               0
day                 0
dtype: int64
```

Missing values in Numerical Features can be imputed using Mean and Median. Mean is sensitive to outliers and median is immune to outliers. If you want to impute the missing values with mean values, then outliers in numerical features need to be addressed properly.

## Outliers detection and treatment

What is an outlier?

An Outlier is an observation that lies an abnormal distance from other values in a given sample. They can be detected using visualization(like boxplots, scatter plots), Z-score, statistical and probabilistic algorithms

Outlier Treatment to remove outliers from Numerical Features

In [ ]:

In [ ]:
```
features_with_outliers = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'WindGustSpeed','WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm']
for feature in features_with_outliers:
    q1 = df[feature].quantile(0.25)
    q3 = df[feature].quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (IQR*1.5)
    upper_limit = q3 + (IQR*1.5)
    df.loc[df[feature]<lower_limit,feature] = lower_limit
    df.loc[df[feature]>upper_limit,feature] = upper_limit
```

Now, numerical features are free from outliers. Let's Impute missing values in numerical features using mean.

In [45]:
```
numerical_features_with_null = [feature for feature in numerical_features if df[feature].isnull().sum()]
for feature in numerical_features_with_null:
    mean_value = df[feature].mean()
    df[feature].fillna(mean_value,inplace=True)
```

It's time to do some analysis on each feature to understand about data and get some insights.

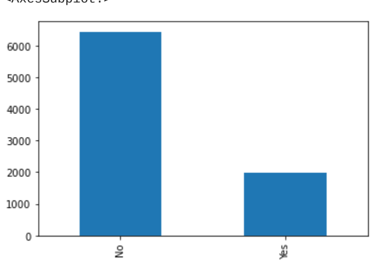
## Exploratory Data Analysis

Exploratory Data Analysis(EDA) is a technique used to analyze, visualize, investigate, interpret, discover and summarize data. It helps Data Scientists to extract trends, patterns, and relationships in data.

Exploring target variable

In [46]:
```
df['RainTomorrow'].value_counts().plot(kind='bar')
```

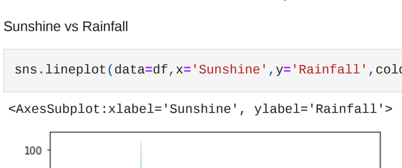
Out[46]:
```
<AxesSubplot:>
```




Sunshine vs Rainfall

In [47]:
```
sns.lineplot(data=df,x='Sunshine',y='Rainfall',color='green')
```

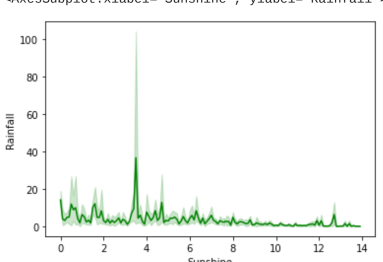
Out[47]:
```
<AxesSubplot:xlabel='Sunshine', ylabel='Rainfall'>
```




Sunshine vs Evaporation

In [48]:
```
sns.lineplot(data=df,x='Sunshine',y='Evaporation',color='blue')
```

Out[48]:
```
<AxesSubplot:xlabel='Sunshine', ylabel='Evaporation'>
```




In the above line plot, the Sunshine feature is proportional to the Evaporation feature

## Correlation

Correlation is a statistic that helps to measure the strength of the relationship between two features. It is used in bivariate analysis. Correlation can be calculated with method corr() in pandas.

In [66]:
```
df.corr()
```

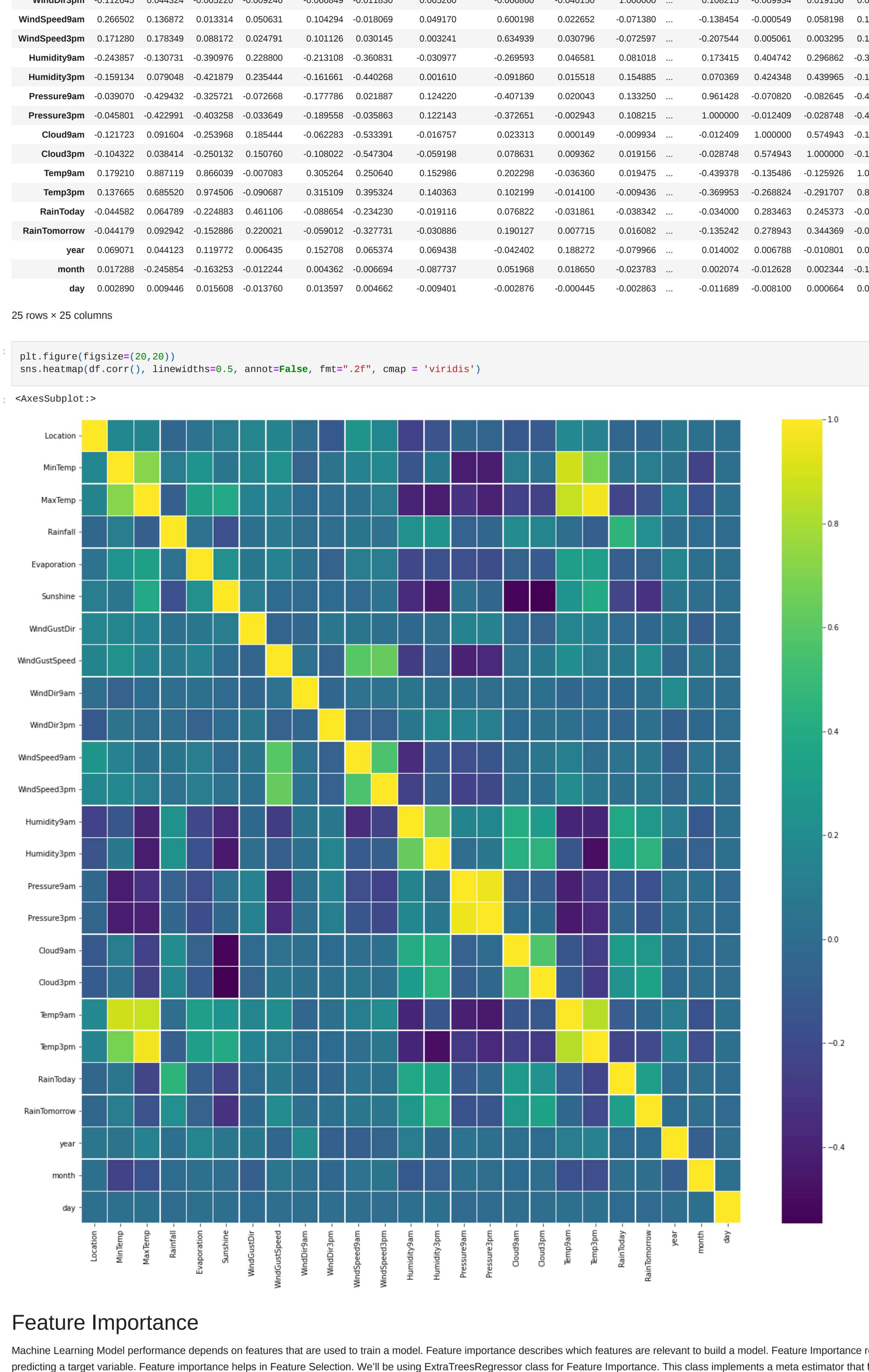
Out[66]:

| | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm | ... | Pressure3pm | Cloud9am | Cloud3pm | Temp9am | Temp3pm | RainToday | RainTomorrow | year | month | day |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Location | 1.000000 | 0.165761 | 0.142541 | -0.041452 | 0.031072 | 0.095019 | 0.149644 | 0.143151 | -0.005839 | -0.112645 | ... | -0.045801 | -0.121723 | -0.104322 | 0.179210 | 0.137665 | -0.044582 | -0.044179 | 0.069071 | 0.017288 | 0.002890 |
| MinTemp | 0.165761 | 1.000000 | 0.717522 | 0.087684 | 0.254965 | 0.056184 | 0.152939 | 0.230058 | -0.066303 | 0.044324 | ... | -0.422991 | 0.091604 | 0.038414 | 0.887119 | 0.685520 | 0.064789 | 0.092942 | 0.044123 | -0.245854 | 0.009446 |
| MaxTemp | 0.142541 | 0.717522 | 1.000000 | -0.087451 | 0.322405 | 0.378403 | 0.133165 | 0.139998 | -0.012844 | -0.005220 | ... | -0.403258 | -0.253968 | -0.250132 | 0.866039 | 0.974506 | -0.224883 | -0.152886 | 0.119772 | -0.163253 | 0.015608 |
| Rainfall | -0.041452 | 0.087684 | -0.087451 | 1.000000 | 0.023517 | -0.169432 | 0.011102 | 0.086000 | -0.002192 | -0.009246 | ... | -0.033649 | 0.185444 | 0.150760 | -0.007083 | -0.090687 | 0.461106 | 0.220021 | 0.006435 | -0.012244 | -0.013760 |
| Evaporation | 0.031072 | 0.254965 | 0.322405 | 0.023517 | 1.000000 | 0.226001 | 0.073477 | 0.136371 | 0.015346 | -0.066849 | ... | -0.189558 | -0.062283 | -0.108022 | 0.305264 | 0.315109 | -0.088654 | -0.059012 | 0.152708 | 0.004362 | 0.013597 |
| Sunshine | 0.095019 | 0.056184 | 0.378403 | -0.169432 | 0.226001 | 1.000000 | 0.096824 | -0.013778 | -0.015805 | -0.011830 | ... | -0.035863 | -0.533391 | -0.547304 | 0.250640 | 0.395324 | -0.234230 | -0.327731 | 0.065374 | -0.006694 | 0.004662 |
| WindGustDir | 0.149644 | 0.152939 | 0.133165 | 0.011102 | 0.073477 | 0.096824 | 1.000000 | -0.054493 | -0.042240 | 0.065260 | ... | 0.122143 | -0.016757 | -0.059198 | 0.152986 | 0.140363 | -0.019116 | -0.030886 | 0.069438 | -0.087737 | -0.009401 |
| WindGustSpeed | 0.143151 | 0.230058 | 0.139998 | 0.086000 | 0.136371 | -0.013778 | -0.054493 | 1.000000 | 0.026369 | -0.066860 | ... | -0.372651 | 0.023313 | 0.078631 | 0.202298 | 0.102199 | 0.076822 | 0.190127 | -0.042402 | 0.051968 | -0.002876 |
| WindDir9am | -0.005839 | -0.066303 | -0.012844 | -0.002192 | 0.015346 | -0.015805 | -0.042240 | 0.026369 | 1.000000 | -0.040150 | ... | -0.002943 | 0.000149 | 0.009362 | -0.036360 | -0.014100 | -0.031861 | 0.007715 | 0.188272 | 0.018650 | -0.000445 |
| WindDir3pm | -0.112645 | 0.044324 | -0.005220 | -0.009246 | -0.066849 | -0.011830 | 0.065260 | -0.066860 | -0.040150 | 1.000000 | ... | 0.108215 | -0.009934 | 0.019156 | 0.019475 | -0.009436 | -0.038342 | 0.016082 | -0.079966 | -0.023783 | -0.002863 |
| WindSpeed9am | 0.266502 | 0.136872 | 0.013314 | 0.050631 | 0.104294 | -0.018069 | 0.049170 | 0.600198 | 0.022652 | -0.071380 | ... | -0.138454 | -0.000549 | 0.058198 | 0.109798 | -0.002606 | 0.035531 | 0.068285 | -0.088489 | 0.038427 | -0.003067 |
| WindSpeed3pm | 0.171280 | 0.178349 | 0.088172 | 0.024791 | 0.101126 | 0.030145 | 0.003241 | 0.634939 | 0.030796 | -0.072597 | ... | -0.207544 | 0.005061 | 0.003295 | 0.184188 | 0.066087 | 0.015268 | 0.053568 | -0.051466 | 0.049483 | -0.008957 |
| Humidity9am | -0.243857 | -0.130731 | -0.390976 | 0.228800 | -0.213108 | -0.360831 | -0.030977 | -0.269593 | 0.046581 | 0.081018 | ... | 0.173415 | 0.404742 | 0.296862 | -0.378344 | -0.382401 | 0.374007 | 0.271222 | 0.099717 | -0.115530 | 0.003799 |
| Humidity3pm | -0.159134 | 0.079048 | -0.421879 | 0.235444 | -0.161661 | -0.440268 | 0.001610 | -0.091860 | 0.015518 | 0.154885 | ... | 0.070369 | 0.424348 | 0.439965 | -0.130414 | -0.491485 | 0.359219 | 0.444808 | -0.028920 | -0.074256 | 0.012512 |
| Pressure9am | -0.039070 | -0.429432 | -0.325721 | -0.072668 | -0.177786 | 0.021887 | 0.124220 | -0.407139 | 0.020043 | 0.133250 | ... | 0.961428 | -0.070820 | -0.082645 | -0.410849 | -0.283412 | -0.104411 | -0.163985 | 0.030558 | 0.007156 | -0.015974 |
| Pressure3pm | -0.045801 | -0.422991 | -0.403258 | -0.033649 | -0.189558 | -0.035863 | 0.122143 | -0.372651 | -0.002943 | 0.108215 | ... | 1.000000 | -0.012409 | -0.028748 | -0.439378 | -0.369953 | -0.034000 | -0.135242 | 0.014002 | 0.002074 | -0.011689 |
| Cloud9am | -0.121723 | 0.091604 | -0.253968 | 0.185444 | -0.062283 | -0.533391 | -0.016757 | 0.023313 | 0.000149 | -0.009934 | ... | -0.012409 | 1.000000 | 0.574943 | -0.135486 | -0.268824 | 0.283463 | 0.278943 | 0.006788 | -0.012628 | -0.008100 |
| Cloud3pm | -0.104322 | 0.038414 | -0.250132 | 0.150760 | -0.108022 | -0.547304 | -0.059198 | 0.078631 | 0.009362 | 0.019156 | ... | -0.028748 | 0.574943 | 1.000000 | -0.125926 | -0.291707 | 0.245373 | 0.344369 | -0.010801 | 0.002344 | 0.000664 |
| Temp9am | 0.179210 | 0.887119 | 0.866039 | -0.007083 | 0.305264 | 0.250640 | 0.152986 | 0.202298 | -0.036360 | 0.019475 | ... | -0.439378 | -0.135486 | -0.125926 | 1.000000 | 0.834165 | -0.097332 | -0.023211 | 0.088735 | -0.165236 | 0.010978 |
| Temp3pm | 0.137665 | 0.685520 | 0.974506 | -0.090687 | 0.315109 | 0.395324 | 0.140363 | 0.102199 | -0.014100 | -0.009436 | ... | -0.369953 | -0.268824 | -0.291707 | 0.834165 | 1.000000 | -0.234693 | -0.193805 | 0.132554 | -0.176218 | 0.012951 |
| RainToday | -0.044582 | 0.064789 | -0.224883 | 0.461106 | -0.088654 | -0.234230 | -0.019116 | 0.076822 | -0.031861 | -0.038342 | ... | -0.034000 | 0.283463 | 0.245373 | -0.097332 | -0.234693 | 1.000000 | 0.312929 | -0.011282 | -0.000801 | -0.015281 |
| RainTomorrow | -0.044179 | 0.092942 | -0.152886 | 0.220021 | -0.059012 | -0.327731 | -0.030886 | 0.190127 | 0.007715 | 0.016082 | ... | -0.135242 | 0.278943 | 0.344369 | -0.023211 | -0.193805 | 0.312929 | 1.000000 | -0.011066 | -0.001243 | -0.015461 |
| year | 0.069071 | 0.044123 | 0.119772 | 0.006435 | 0.152708 | 0.065374 | 0.069438 | -0.042402 | 0.188272 | -0.079966 | ... | 0.014002 | 0.006788 | -0.010801 | 0.088735 | 0.132554 | -0.011282 | -0.011066 | 1.000000 | -0.090394 | -0.003836 |
| month | 0.017288 | -0.245854 | -0.163253 | -0.012244 | 0.004362 | -0.006694 | -0.087737 | 0.051968 | 0.018650 | -0.023783 | ... | 0.002074 | -0.012628 | 0.002344 | -0.165236 | -0.176218 | -0.000801 | -0.001243 | -0.090394 | 1.000000 | 0.004386 |
| day | 0.002890 | 0.009446 | 0.015608 | -0.013760 | 0.013597 | 0.004662 | -0.009401 | -0.002876 | -0.000445 | -0.002863 | ... | -0.011689 | -0.008100 | 0.000664 | 0.010978 | 0.012951 | -0.015281 | -0.015461 | -0.003836 | 0.004386 | 1.000000 |

25 rows × 25 columns

In [53]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), linewidths=0.5, annot=False, fmt=".2f", cmap = 'viridis')
```

Out[53]:
```
<AxesSubplot:>
```




## Feature Importance

Machine Learning Model performance depends on features that are used to train a model. Feature importance describes which features are relevant to build a model. Feature Importance refers to the techniques that assign a score to input/label features based on how useful they are at predicting a target variable. Feature importance helps in Feature Selection. We'll be using ExtraTreesRegressor class for Feature Importance. This class implements a meta estimator that fits a number of randomized decision trees on various samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

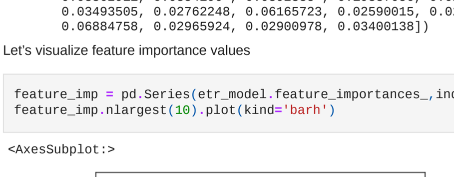
In [63]:
```
from sklearn.ensemble import ExtraTreesRegressor
etr_model = ExtraTreesRegressor()
etr_model.fit(X,y)
etr_model.feature_importances_
```

Out[63]:
```
array([0.02528765, 0.0280224 , 0.02732338, 0.03363168, 0.0235105 ,
       0.03821707, 0.02873733, 0.05853214, 0.03223321, 0.03343378,
       0.03362911, 0.0354106 , 0.0392335 , 0.19337056, 0.02885953,
       0.03493505, 0.02762248, 0.06165723, 0.02590015, 0.02893469,
       0.06884758, 0.02965924, 0.02900978, 0.03400138])
```

Let's visualize feature importance values

In [64]:
```
feature_imp = pd.Series(etr_model.feature_importances_,index=X.columns)
feature_imp.nlargest(10).plot(kind='barh')
```

Out[64]:
```
<AxesSubplot:>
```




## conclusion

These methods are extremely easy to adopt as they don't require any specific computational power like Deep Learning methods

Nonetheless, predictions perfectly fit in the error range designed by the dataset itself. It is important to consider that we only have examined monthly average values while it may be interesting to consider daily values too and have daily predictions.

I am using several technique to predict whether it will rain tomorrow or not.

## THANK YOU