[8]:
```
# Install necessary packages and upgrade them together to manage dependencies better
!pip install pandas-profiling numpy matplotlib seaborn opendatasets scikit-learn jovian --quiet --upgrade
```

[9]:
```
import opendatasets as od
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib
import jovian
import os
%matplotlib inline

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 150)
sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

DOWNLOADING THE DATA:

[10]:
```
od.download('https://www.kaggle.com/jsphyg/weather-dataset-rattle-package')
```

```
Skipping, found downloaded files in ".\weather-dataset-rattle-package" (use force=True to force download)
```

[11]:
```
os.listdir('weather-dataset-rattle-package')
```

[11]:
```
['weatherAUS.csv']
```

[12]:
```
raw_df = pd.read_csv('weather-dataset-rattle-package/weatherAUS.csv')
```

[13]:
```
raw_df
```

[13]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pr
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	20.0	24.
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	4.0	22.
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	19.0	26.
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	11.0	9.
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	7.0	20.
...	...	...	...	...	...	...	...	...	...	...	...	...	...
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	ENE	13.0	11.
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	N	13.0	9.
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	WNW	9.0	9.
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	N	13.0	7.
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	ESE	ESE	17.0	17.

145460 rows × 23 columns

[14]:
```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
```

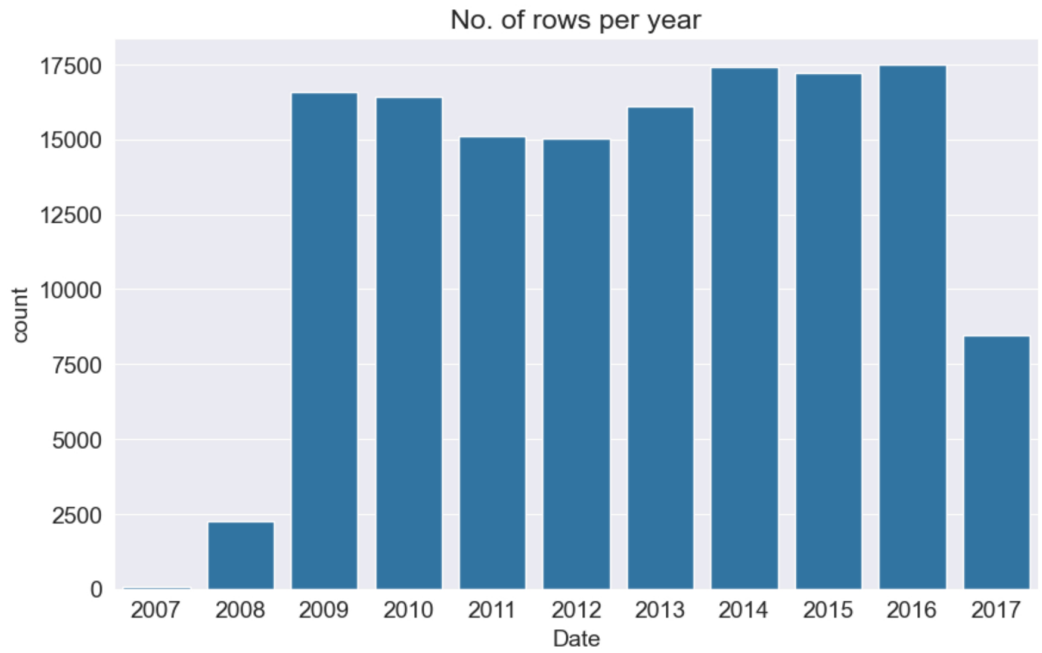
```
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

[15]:
```
raw_df.dropna(subset=['RainTomorrow'], inplace = True)
```

PREPARING DATA FOR TRAINING:

TRAINING, VALIDATION AND TEST SPLITS:

[17]:
```
plt.title('No. of rows per year')
sns.countplot(x=pd.to_datetime(raw_df.Date).dt.year);
```



[18]:
```
year = pd.to_datetime(raw_df.Date).dt.year

train_df = raw_df[year<2015]
val_df = raw_df[year==2015]
test_df = raw_df[year>2015]
```

[19]:
```
print('train_df.shape:', train_df.shape)
print('val_df.shape:', val_df.shape)
print('test_df.shape:', test_df.shape)
```

```
train_df.shape: (98988, 23)
val_df.shape: (17231, 23)
test_df.shape: (25974, 23)
```

INPUT AND TARGET COLUMNS:

[20]:
```
input_cols = list(train_df.columns)[1:-1]
target_col = 'RainTomorrow'
```

[23]:
```
train_inputs = train_df[input_cols].copy()
train_targets = train_df[target_col].copy()
```

[27]:
```
train_inputs
```

[27]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Hun
0	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	20.0	24.0	
1	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	4.0	22.0	

2	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	19.0	26.0
3	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	11.0	9.0
4	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	7.0	20.0
...	...	...	...	...	...	...	...	...	...	...	...	...
144548	Uluru	16.9	33.2	0.0	NaN	NaN	SSE	43.0	ESE	SSE	24.0	26.0
144549	Uluru	15.1	36.8	0.0	NaN	NaN	NE	31.0	ENE	SW	19.0	20.0
144550	Uluru	17.3	37.8	0.0	NaN	NaN	ESE	39.0	ESE	SSE	26.0	9.0
144551	Uluru	20.1	38.5	0.0	NaN	NaN	ESE	43.0	ESE	SSW	28.0	17.0
144552	Uluru	22.5	39.6	0.0	NaN	NaN	WNW	76.0	ENE	SSW	30.0	13.0

98988 rows × 21 columns

[28]:
```
train_targets
```

[28]:
```
0         No
1         No
2         No
3         No
4         No
          ..
144548    No
144549    No
144550    No
144551    No
144552    No
Name: RainTomorrow, Length: 98988, dtype: object
```

[29]:
```
val_inputs = val_df[input_cols].copy()
val_targets = val_df[target_col].copy()
```

[30]:
```
test_inputs = test_df[input_cols].copy()
test_targets = test_df[target_col].copy()
```

[33]:
```
train_inputs.select_dtypes(include = np.number).columns
```

[33]:
```
Index(['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
       'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
       'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
       'Temp9am', 'Temp3pm'],
      dtype='object')
```

[35]:
```
train_inputs.select_dtypes(include = 'object').columns
```

[35]:
```
Index(['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday'], dtype='object')
```

[36]:
```
numeric_cols = train_inputs.select_dtypes(include = np.number).columns.tolist()
categorical_cols = train_inputs.select_dtypes(include = 'object').columns.tolist()
```

[39]:
```
print(numeric_cols)
print(categorical_cols)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

[42]:
```
train_inputs[numeric_cols].isna().sum().sort_values(ascending = False)
```

[42]:
```
Sunshine         40696
Evaporation      37110
Cloud3pm         36766
Cloud9am         35764
Pressure9am       9345
Pressure3pm       9309
WindGustSpeed     6902
Humidity9am       1265
Humidity3pm       1186
WindSpeed3pm      1140
WindSpeed9am      1133
Rainfall          1000
Temp9am            783
Temp3pm            663
MinTemp            434
MaxTemp            198
dtype: int64
```

IMPUTING MISSING NUMERICAL VALUES:

[43]:
```
from sklearn.impute import SimpleImputer
```

[44]:
```
imputer = SimpleImputer(strategy = 'mean').fit(raw_df[numeric_cols])
```

[45]:
```
train_inputs[numeric_cols] = imputer.transform(train_inputs[numeric_cols])
val_inputs[numeric_cols] = imputer.transform(val_inputs[numeric_cols])
test_inputs[numeric_cols] = imputer.transform(test_inputs[numeric_cols])
```

[46]:
```
test_inputs[numeric_cols].isna().sum()
```

[46]:
```
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustSpeed    0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
dtype: int64
```

[47]:
```
val_inputs.describe().loc[['min','max']]
```

[47]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	C
min	-8.2	-3.2	0.0	0.0	0.0	7.0	0.0	0.0	4.0	0.0	988.1	982.2	
max	31.9	45.4	247.2	70.4	14.5	135.0	87.0	74.0	100.0	100.0	1039.3	1037.3	

SCALING NUMERIC FEATURES:

[48]:
```
from sklearn.preprocessing import MinMaxScaler
```

[49]:
```
scaler = MinMaxScaler().fit(raw_df[numeric_cols])
```

[50]:
```
train_inputs[numeric_cols] = scaler.transform(train_inputs[numeric_cols])
val_inputs[numeric_cols] = scaler.transform(val_inputs[numeric_cols])
test_inputs[numeric_cols] = scaler.transform(test_inputs[numeric_cols])
```

[51]:
```
val_inputs.describe().loc[['min','max']]
```

[51]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm
min	0.007075	0.030246	0.000000	0.000000	0.0	0.007752	0.000000	0.000000	0.04	0.0	0.125620	0.0816
max	0.952830	0.948960	0.666307	0.485517	1.0	1.000000	0.669231	0.850575	1.00	1.0	0.971901	0.9632

[52]:
```
categorical_cols
```

[52]:
```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

ENCODING CATEGORICAL DATA:

[53]:
```
from sklearn.preprocessing import OneHotEncoder
```

[56]:
```
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore').fit(raw_df[categorical_cols])
```

```
C:\Users\adity\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:975: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 an
d will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[57]:
```
encoded_cols = list(encoder.get_feature_names_out(categorical_cols))
```

[58]:
```
encoded_cols
```

[58]:
```
['Location_Adelaide',
 'Location_Albany',
 'Location_Albury',
 'Location_AliceSprings',
 'Location_BadgerysCreek',
 'Location_Ballarat',
 'Location_Bendigo',
 'Location_Brisbane',
 'Location_Cairns',
 'Location_Canberra',
 'Location_Cobar',
 'Location_CoffsHarbour',
 'Location_Dartmoor',
 'Location_Darwin',
 'Location_GoldCoast',
 'Location_Hobart',
 'Location_Katherine',
 'Location_Launceston',
```

[61]:
```
train_inputs[encoded_cols] = encoder.transform(train_inputs[categorical_cols])
val_inputs[encoded_cols] = encoder.transform(val_inputs[categorical_cols])
test_inputs[encoded_cols] = encoder.transform(test_inputs[categorical_cols])
```

[62]:
```
test_inputs
```

[62]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Hu

2498	Albury	0.681604	0.801512	0.000000	0.037723	0.525852	ENE	0.372093	NaN	ESE	0.000000	0.080460
2499	Albury	0.693396	0.725898	0.001078	0.037723	0.525852	SSE	0.341085	SSE	SE	0.069231	0.195402
2500	Albury	0.634434	0.527410	0.005930	0.037723	0.525852	ENE	0.325581	ESE	ENE	0.084615	0.448276
2501	Albury	0.608491	0.538752	0.042049	0.037723	0.525852	SSE	0.255814	SE	SSE	0.069231	0.195402
2502	Albury	0.566038	0.523629	0.018329	0.037723	0.525852	ENE	0.193798	SE	SSE	0.046154	0.103448
...	...	...	...	...	...	...	...	...	...	...	...	...
145454	Uluru	0.283019	0.502836	0.000000	0.037723	0.525852	E	0.193798	ESE	E	0.115385	0.149425
145455	Uluru	0.266509	0.533081	0.000000	0.037723	0.525852	E	0.193798	SE	ENE	0.100000	0.126437
145456	Uluru	0.285377	0.568998	0.000000	0.037723	0.525852	NNW	0.124031	SE	N	0.100000	0.103448
145457	Uluru	0.327830	0.599244	0.000000	0.037723	0.525852	N	0.240310	SE	WNW	0.069231	0.103448
145458	Uluru	0.384434	0.601134	0.000000	0.037723	0.525852	SE	0.170543	SSE	N	0.100000	0.080460

25974 rows × 124 columns

[64]:
```
X_train = train_inputs[numeric_cols + encoded_cols]
X_val = val_inputs[numeric_cols + encoded_cols]
X_test = test_inputs[numeric_cols + encoded_cols]
```

[65]:
```
X_test
```

[65]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pn
2498	0.681604	0.801512	0.000000	0.037723	0.525852	0.372093	0.000000	0.080460	0.46	0.17	0.543802	0.513
2499	0.693396	0.725898	0.001078	0.037723	0.525852	0.341085	0.069231	0.195402	0.54	0.30	0.505785	0.500
2500	0.634434	0.527410	0.005930	0.037723	0.525852	0.325581	0.084615	0.448276	0.62	0.67	0.553719	0.603
2501	0.608491	0.538752	0.042049	0.037723	0.525852	0.255814	0.069231	0.195402	0.74	0.65	0.618182	0.630
2502	0.566038	0.523629	0.018329	0.037723	0.525852	0.193798	0.046154	0.103448	0.92	0.63	0.591736	0.588
...	...	...	...	...	...	...	...	...	...	...	...	...
145454	0.283019	0.502836	0.000000	0.037723	0.525852	0.193798	0.115385	0.149425	0.59	0.27	0.730579	0.705
145455	0.266509	0.533081	0.000000	0.037723	0.525852	0.193798	0.100000	0.126437	0.51	0.24	0.728926	0.691
145456	0.285377	0.568998	0.000000	0.037723	0.525852	0.124031	0.100000	0.103448	0.56	0.21	0.710744	0.672
145457	0.327830	0.599244	0.000000	0.037723	0.525852	0.240310	0.069231	0.103448	0.53	0.24	0.669421	0.635
145458	0.384434	0.601134	0.000000	0.037723	0.525852	0.170543	0.100000	0.080460	0.51	0.24	0.642975	0.630

25974 rows × 119 columns

TRAINING DECISION TREES:

[66]:
```
from sklearn.tree import DecisionTreeClassifier
```

[68]:
```
model = DecisionTreeClassifier(random_state = 42)
```

[69]:
```
model.fit(X_train, train_targets)
```

[69]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

[70]:
```
train_preds = model.predict(X_train)
```

[71]:
```
train_preds
```

[71]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

[72]:
```
pd.value_counts(train_preds)
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\1258350197.py:1: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version.
Use pd.Series(obj).value_counts() instead.
  pd.value_counts(train_preds)
```

[72]:
```
No     76707
Yes    22281
Name: count, dtype: int64
```

EVALUATION:

[74]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

[75]:
```
accuracy_score (train_preds,train_targets)
```

[75]:
```
0.9999797955307714
```

[77]:
```
train_probs = model.predict_proba(X_train)
```

[77]:
```
train_probs = model.predict_proba(X_train)
train_probs
```

[77]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

[78]:
```
model.score(X_val, val_targets)
```

[78]:
```
0.7921188555510418
```

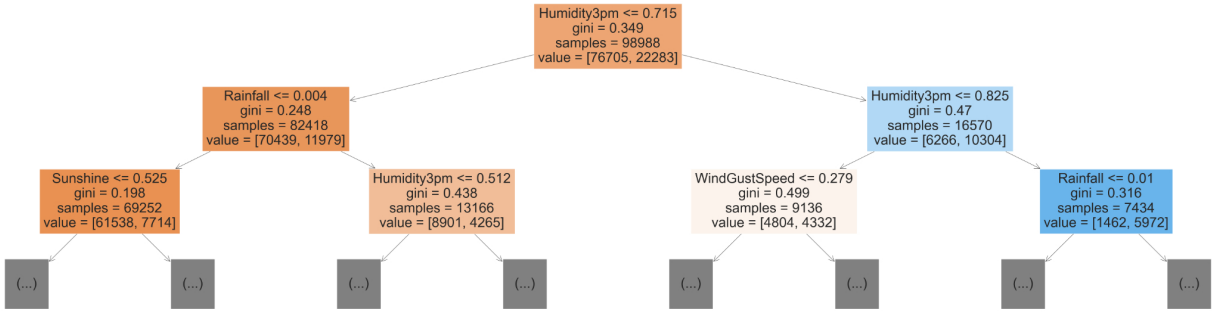
[79]:
```
val_targets.value_counts() / len(val_targets)
```

[79]:
```
RainTomorrow
No     0.788289
Yes    0.211711
Name: count, dtype: float64
```

VISUALIZATION:

[81]:
```
from sklearn.tree import plot_tree, export_text
```

[82]:
```
plt.figure(figsize=(80,20))
plot_tree(model, feature_names=X_train.columns, max_depth=2, filled=True);
```



[83]:
```
model.tree_.max_depth
```

[83]:
```
48
```

[84]:
```
tree_text = export_text(model, max_depth = 10, feature_names = list(X_train.columns))
print (tree_text[:5000])
```

```
|--- Humidity3pm <= 0.72
|   |--- Rainfall <= 0.00
|   |   |--- Sunshine <= 0.52
|   |   |   |--- Pressure3pm <= 0.58
|   |   |   |   |--- WindGustSpeed <= 0.36
|   |   |   |   |   |--- Humidity3pm <= 0.28
|   |   |   |   |   |   |--- WindDir9am_NE <= 0.50
|   |   |   |   |   |   |   |--- Location_Watsonia <= 0.50
|   |   |   |   |   |   |   |   |--- Cloud9am <= 0.83
|   |   |   |   |   |   |   |   |   |--- WindSpeed3pm <= 0.07
|   |   |   |   |   |   |   |   |   |   |--- Pressure3pm <= 0.46
|   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |--- Pressure3pm >  0.46
|   |   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |   |--- WindSpeed3pm >  0.07
|   |   |   |   |   |   |   |   |   |   |--- MinTemp <= 0.32
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- MinTemp >  0.32
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |   |--- Cloud9am >  0.83
|   |   |   |   |   |   |   |   |   |   |--- Cloud3pm <= 0.42
|   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |--- Cloud3pm >  0.42
|   |   |   |   |   |   |   |   |   |   |   |--- Rainfall <= 0.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |   |--- Rainfall >  0.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |--- Location_Watsonia >  0.50
|   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |--- WindDir9am_NE >  0.50
|   |   |   |   |   |   |   |   |--- WindGustSpeed <= 0.25
|   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |--- WindGustSpeed >  0.25
|   |   |   |   |   |   |   |   |   |--- Pressure9am <= 0.54
|   |   |   |   |   |   |   |   |   |   |--- Evaporation <= 0.09
|   |   |   |   |   |   |   |   |   |   |   |--- Location_AliceSprings <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |   |--- Location_AliceSprings >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |--- Evaporation >  0.09
|   |   |   |   |   |   |   |   |   |   |--- WindGustDir_ENE <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |--- WindGustDir_ENE >  0.50
```

```
|   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |   |--- Pressure9am >  0.54
|   |   |   |   |   |   |   |   |   |   |--- Humidity3pm <= 0.20
|   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |--- Humidity3pm >  0.20
|   |   |   |   |   |   |   |   |   |   |   |--- Evaporation <= 0.02
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |   |--- Evaporation >  0.02
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |--- Humidity3pm >  0.28
|   |   |   |   |   |   |   |--- Sunshine <= 0.05
|   |   |   |   |   |   |   |   |--- WindGustSpeed <= 0.25
|   |   |   |   |   |   |   |   |   |--- Evaporation <= 0.01
|   |   |   |   |   |   |   |   |   |   |--- WindGustSpeed <= 0.23
|   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |--- WindGustSpeed >  0.23
|   |   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |   |--- Evaporation >  0.01
|   |   |   |   |   |   |   |   |   |   |--- Evaporation <= 0.07
|   |   |   |   |   |   |   |   |   |   |   |--- Temp3pm <= 0.34
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |   |--- Temp3pm >  0.34
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 11
|   |   |   |   |   |   |   |   |   |   |--- Evaporation >  0.07
|   |   |   |   |   |   |   |   |   |   |   |--- WindSpeed9am <= 0.12
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |   |--- WindSpeed9am >  0.12
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |--- WindGustSpeed >  0.25
|   |   |   |   |   |   |   |   |   |--- Pressure9am <= 0.56
|   |   |   |   |   |   |   |   |   |   |--- MinTemp <= 0.40
|   |   |   |   |   |   |   |   |   |   |   |--- WindDir9am_WNW <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: Yes
|   |   |   |   |   |   |   |   |   |   |   |--- WindDir9am_WNW >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- class: No
|   |   |   |   |   |   |   |   |   |   |--- MinTemp >  0.40
|   |   |   |   |   |   |   |   |   |   |   |--- Humidity3pm <= 0.66
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |   |   |   |--- Humidity3pm >  0.66
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |--- Pressure9am >  0.56
|   |   |   |   |   |
```

FEATURE IMPORTANCE:

[85]:
```
model.feature_importances_
```

[85]:
```
array([3.48942086e-02, 3.23605486e-02, 5.91385668e-02, 2.49619907e-02,
       4.94652143e-02, 5.63334673e-02, 2.80205998e-02, 2.98128801e-02,
       4.02182908e-02, 2.61441297e-01, 3.44145027e-02, 6.20573699e-02,
       1.36406176e-02, 1.69229866e-02, 3.50001550e-02, 3.04064076e-02,
       2.24086587e-03, 2.08018104e-03, 1.27475954e-03, 7.26936324e-04,
       1.39779517e-03, 1.15264873e-03, 6.92808159e-04, 1.80675598e-03,
       1.08370901e-03, 1.19773895e-03, 8.87119103e-04, 2.15764220e-03,
       1.67094731e-03, 7.98919493e-05, 1.10558668e-03, 1.42008656e-03,
       4.10087635e-04, 1.09028115e-03, 1.44164766e-03, 9.08284767e-04,
       1.05770304e-03, 6.18133455e-04, 1.80387272e-03, 2.10403527e-03,
       2.74413333e-04, 7.31599405e-04, 1.35408990e-03, 1.54759332e-03,
       1.30917564e-03, 1.07134670e-03, 8.36408023e-04, 1.62662229e-03,
       1.00326116e-03, 2.16053455e-03, 8.46802258e-04, 1.88919081e-03,
       9.29325203e-04, 1.29545157e-03, 1.27604831e-03, 5.12736888e-04,
       1.38458902e-03, 3.97103931e-04, 1.03734689e-03, 1.44437047e-03,
       1.75870184e-03, 1.42487857e-03, 2.78109569e-03, 2.00782698e-03,
       2.80617652e-04, 1.61509734e-03, 1.64361598e-03, 2.36124112e-03,
       3.05457932e-03, 2.33239534e-03, 2.78643875e-03, 2.16695261e-03,
       3.41491352e-03, 2.30573542e-03, 2.28270604e-03, 2.34408118e-03,
       2.26557332e-03, 2.54592702e-03, 2.75264499e-03, 2.83905192e-03,
       2.49480561e-03, 1.54840338e-03, 2.50305095e-03, 2.53945388e-03,
       2.28130055e-03, 3.80572180e-03, 2.58535069e-03, 3.10172224e-03,
       2.54236791e-03, 2.50297796e-03, 2.06400988e-03, 2.52931192e-03,
       2.07840517e-03, 1.77387278e-03, 1.78920555e-03, 2.77709687e-03,
       2.42564566e-03, 2.26471887e-03, 1.73346117e-03, 2.23926957e-03,
       2.47865244e-03, 2.31917387e-03, 3.21211861e-03, 2.92382975e-03,
       2.24399274e-03, 3.68774754e-03, 3.87595982e-03, 3.20326068e-03,
       2.53323550e-03, 2.40444844e-03, 2.26790411e-03, 2.19744009e-03,
       2.28064147e-03, 2.88545323e-03, 2.05278867e-03, 1.12604304e-03,
       2.86325849e-04, 1.32322128e-03, 1.72690480e-03])
```

[88]:
```
importance_df = pd.DataFrame({
    'feature':X_train.columns,
    'importance':model.feature_importances_
    }).sort_values('importance', ascending = False)
```

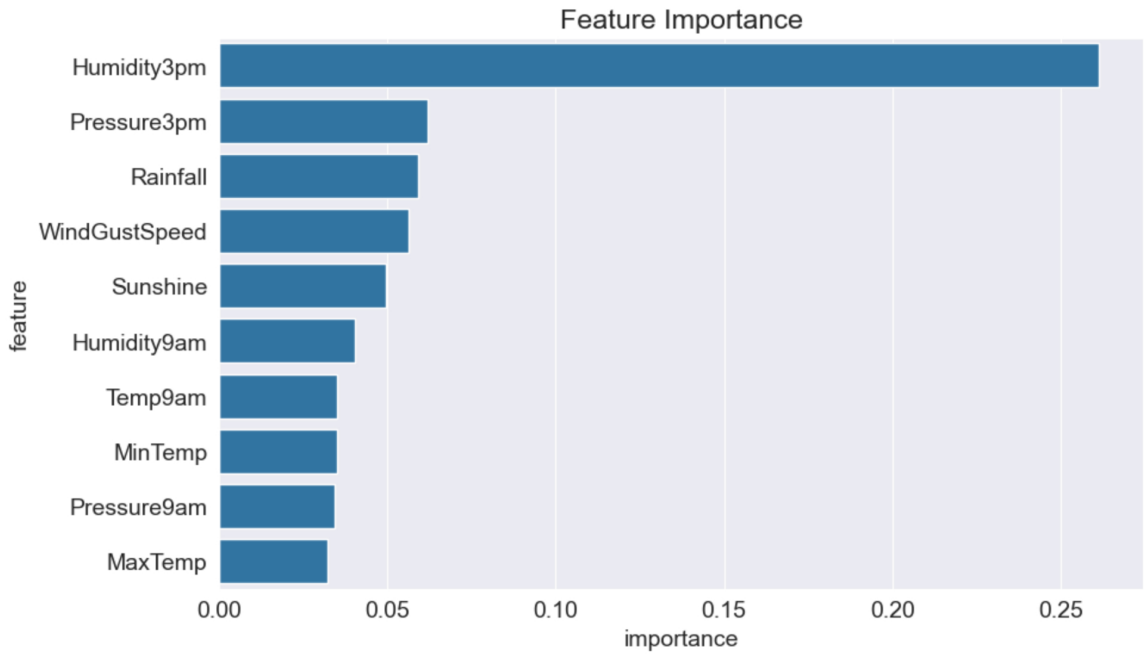
[89]:
```
importance_df.head(10)
```

[89]:

	feature	importance
9	Humidity3pm	0.261441
11	Pressure3pm	0.062057
2	Rainfall	0.059139
5	WindGustSpeed	0.056333
4	Sunshine	0.049465
8	Humidity9am	0.040218

14	Temp9am	0.035000
0	MinTemp	0.034894
10	Pressure9am	0.034415
1	MaxTemp	0.032361

[90]:
```
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(10), x='importance', y='feature');
```



HYPERPARAMETER TUNING AND OVERFITTING:

MAX_DEPTH:

[91]:
```
model = DecisionTreeClassifier(max_depth = 3, random_state = 42)
model.fit(X_train, train_targets)
```

[91]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

[92]:
```
model.score(X_train,train_targets)
```

[92]:
```
0.8291308037337859
```

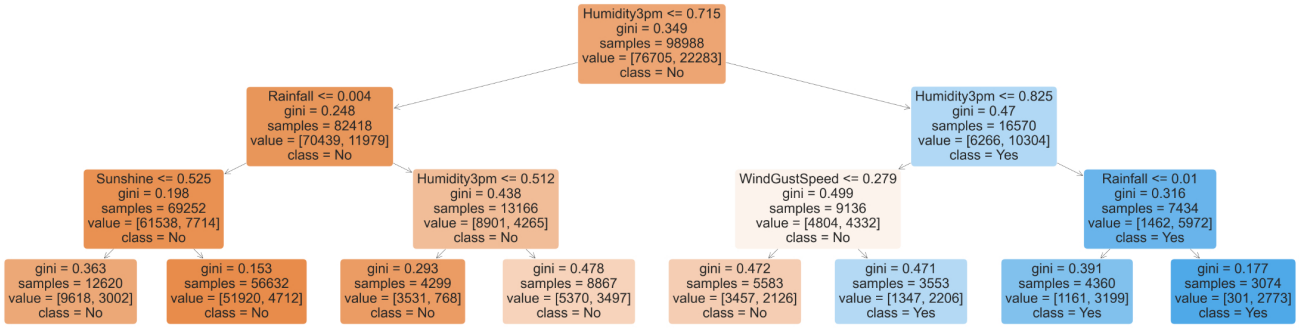
[93]:
```
model.score(X_val,val_targets)
```

[93]:
```
0.8334397307178921
```

[94]:
```
model.classes_
```

[94]:
```
array(['No', 'Yes'], dtype=object)
```

[95]:
```
plt.figure(figsize=(80,20))
plot_tree(model, feature_names=X_train.columns, filled=True, rounded=True, class_names=model.classes_);
```



[107]:
```
def max_depth_error(md):

    model = DecisionTreeClassifier(max_depth = md, random_state = 42)
    model.fit(X_train, train_targets)
```

```
    train_error = 1 - model.score(X_train, train_targets)
    val_error = 1 - model.score(X_val,val_targets)
    return {
    'Max Depth':md,
    'Training Error':train_error,
    'Validation Error':val_error
    }
```

[108]:
```
%%time
errors_df = pd.DataFrame([max_depth_error(md) for md in range(1, 21)])
```

```
CPU times: total: 26.6 s
Wall time: 29.9 s
```

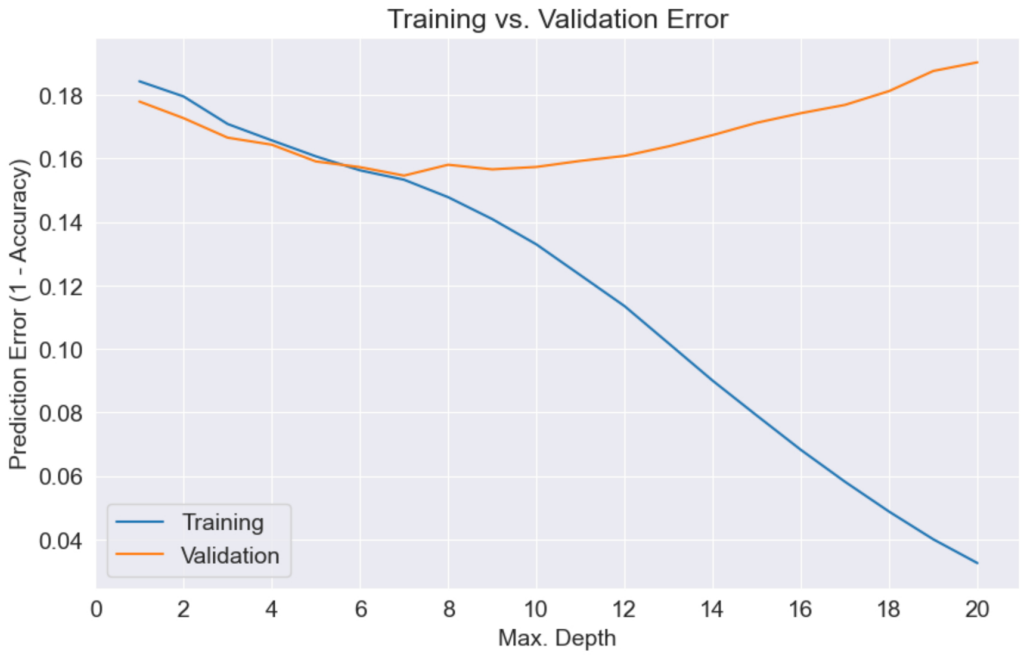
[109]:
```
errors_df
```

[109]:

	Max Depth	Training Error	Validation Error
0	1	0.184315	0.177935
1	2	0.179547	0.172712
2	3	0.170869	0.166560
3	4	0.165707	0.164355
4	5	0.160676	0.159074
5	6	0.156271	0.157275
6	7	0.153312	0.154605
7	8	0.147806	0.158029
8	9	0.140906	0.156578
9	10	0.132945	0.157333
10	11	0.123227	0.159248
11	12	0.113489	0.160815
12	13	0.101750	0.163833
13	14	0.089981	0.167373
14	15	0.078999	0.171261
15	16	0.068180	0.174279
16	17	0.058138	0.176890
17	18	0.048733	0.181243
18	19	0.040025	0.187569
19	20	0.032539	0.190297

[110]:
```
plt.figure()
plt.plot(errors_df['Max Depth'], errors_df['Training Error'])
plt.plot(errors_df['Max Depth'], errors_df['Validation Error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,21, 2))
plt.xlabel('Max. Depth')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])
```

[110]:
```
<matplotlib.legend.Legend at 0x23e8fd81c90>
```



[111]:
```
model = DecisionTreeClassifier(max_depth = 7, random_state = 42)
model.fit(X_train, train_targets)
```

[111]:
```
DecisionTreeClassifier(max_depth=7, random_state=42)
```

[113]:
```
model.score(X_train, train_targets), model.score(X_val,val_targets)
```

[113]:
```
(0.8466884874934335, 0.8453949277465034)
```

MAX_LEAF_NODES:

[114]:
```
model = DecisionTreeClassifier(max_leaf_nodes = 128, random_state = 42)
```

[115]:
```
model.fit(X_train,train_targets)
```

[115]:
```
DecisionTreeClassifier(max_leaf_nodes=128, random_state=42)
```

[116]:
```
model.score(X_train,train_targets)
```

[116]:
```
0.8480421869317493
```

[117]:
```
model.score(X_val, val_targets)
```

[117]:
```
0.8442342290058615
```

[118]:
```
model.tree_.max_depth
```

[118]:
```
12
```

TRAINING A RANDOM FOREST:

[120]:
```
from sklearn.ensemble import RandomForestClassifier
```

[121]:
```
model = RandomForestClassifier(n_jobs = -1, random_state = 42)
```

[122]:
```
model.fit(X_train, train_targets)
```

[122]:
```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

[123]:
```
model.score(X_train, train_targets)
```

[123]:
```
0.9999494888269285
```

[125]:
```
model.score(X_val, val_targets)
```

[125]:
```
0.8566537055307295
```

[126]:
```
train_probs = model.predict_proba(X_train)
train_probs
```

[126]:
```
array([[0.93, 0.07],
       [1.  , 0.  ],
       [0.99, 0.01],
       ...,
       [0.99, 0.01],
       [1.  , 0.  ],
       [0.96, 0.04]])
```

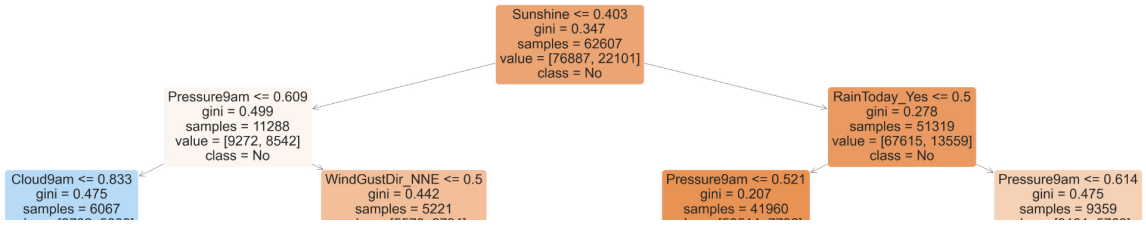
[127]:
```
len(model.estimators_)
```

[127]:
```
100
```

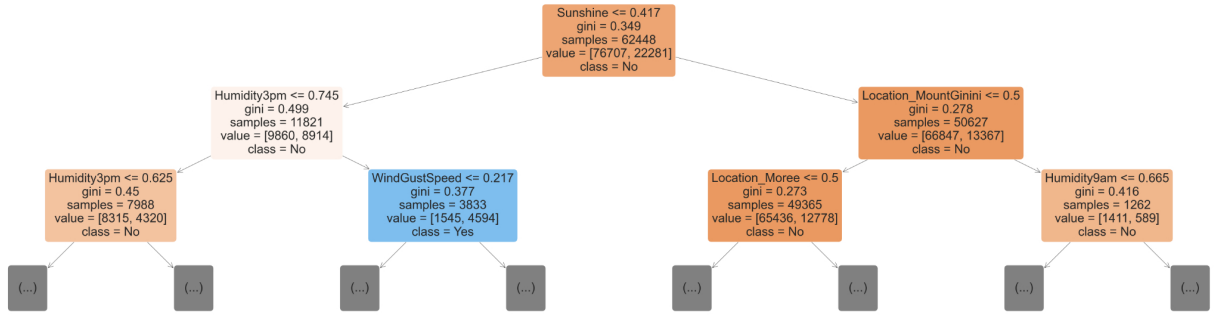
[130]:
```
model.estimators_[0]
```

[130]:
```
DecisionTreeClassifier(max_features='sqrt', random_state=1608637542)
```

[131]:
```
plt.figure(figsize=(80,20))
plot_tree(model.estimators_[0], max_depth=2, feature_names=X_train.columns, filled=True, rounded=True, class_names=model.classes_);
```



[132]:
```
plt.figure(figsize=(80,20))
plot_tree(model.estimators_[15], max_depth=2, feature_names=X_train.columns, filled=True, rounded=True, class_names=model.classes_);
```



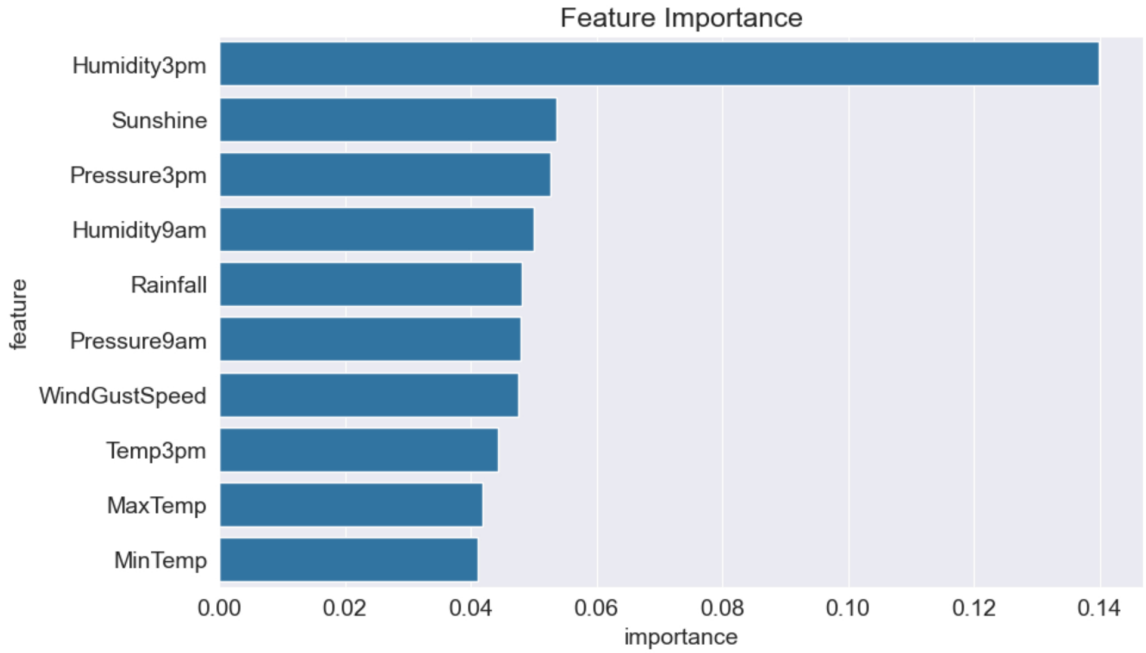
[134]:
```
importance_df = pd.DataFrame({
    'feature':X_train.columns,
    'importance':model.feature_importances_
}).sort_values('importance', ascending = False)
```

[135]:
```
importance_df
```

[135]:

	feature	importance
9	Humidity3pm	0.139904
4	Sunshine	0.053696
11	Pressure3pm	0.052713
8	Humidity9am	0.050051
2	Rainfall	0.048077
10	Pressure9am	0.047944
5	WindGustSpeed	0.047477
15	Temp3pm	0.044379
1	MaxTemp	0.041865

[136]:
```
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(10), x='importance', y='feature');
```



HYPERPARAMETER TUNING WITH RANDOM FORESTS:

[137]:
```
base_model = RandomForestClassifier(n_jobs = -1,random_state = 42).fit(X_train,train_targets)
```

[138]:
```
base_train_acc = base_model.score(X_train, train_targets)
base_val_acc = base_model.score(X_val, val_targets)
```

[139]:
```
base_accs = base_train_acc, base_val_acc
base_accs
```

[139]:
```
(0.9999494888269285, 0.8566537055307295)
```

N_ESTIMATORS:

[143]:
```
model = RandomForestClassifier(n_jobs = -1, random_state = 42, n_estimators = 500)
```

[144]:
```
model.fit(X_train, train_targets)
```

[144]:
```
RandomForestClassifier(n_estimators=500, n_jobs=-1, random_state=42)
```

[145]:
```
model.score(X_train,train_targets), model.score(X_val,val_targets)
```

[145]:
```
(0.9999797955307714, 0.8577563693343393)
```

MAX_DEPTH AND MAX_LEAF_NODES:

[146]:
```
def test_params(**params):
    model = RandomForestClassifier(random_state=42, n_jobs=-1, **params).fit(X_train, train_targets)
    return model.score(X_train, train_targets), model.score(X_val, val_targets)
```

[147]:
```
test_params(max_depth = 5)
```

[147]:
```
(0.8197862367155615, 0.8240961058557251)
```

[148]:
```
test_params(max_depth = 26)
```

[148]:
```
(0.9814826039519942, 0.8572340549010504)
```

[149]:
```
test_params(max_leaf_nodes = 2**5)
```

[149]:
```
(0.8314341132258456, 0.833904010214149)
```

MAX_FEATURES:

[150]:
```
test_params(max_features = 'log2')
```

[150]:
```
(0.9999595910615429, 0.8558992513493123)
```

MIN_SAMPLES_SPLIT AND MIN_SAMPLES_LEAF:

[154]:
```
test_params(min_samples_split = 5, min_samples_leaf = 2)
```

[154]:
```
(0.9573584676930537, 0.855144797167895)
```

MIN_IMPURITY_DECREASE:

[157]:
```
test_params(min_impurity_decrease = 1e-6)
```

[157]:
```
(0.9888168262819735, 0.8561313910974406)
```

BOOTSTRAP, MAX_SAMPLES:

[161]:
```
test_params(bootstrap = True, max_samples = 0.9)
```

[161]:
```
(0.9997676486038711, 0.8565376356566653)
```

[162]:
```
train_targets.value_counts()
```

[162]:
```
RainTomorrow
No     76705
Yes    22283
Name: count, dtype: int64
```

CLASS_WEIGHT:

[163]:
```
test_params(class_weight = {'No' : 1, 'Yes': 2})
```

[163]:
```
(0.9999595910615429, 0.8558412164122802)
```

[164]:
```
base_accs
```

[164]:
```
(0.9999494888269285, 0.8566537055307295)
```

PUTTING IT TOGETHER:

[165]:
```
model = RandomForestClassifier(n_jobs=-1,
                               random_state=42,
                               n_estimators=500,
                               max_features=7,
                               max_depth=30,
                               class_weight={'No': 1, 'Yes': 1.5})
```

[166]:
```
model.fit(X_train, train_targets)
```

[166]:
```
RandomForestClassifier

RandomForestClassifier(class_weight={'No': 1, 'Yes': 1.5}, max_depth=30,
                       max_features=7, n_estimators=500, n_jobs=-1,
                       random_state=42)
```

[167]:
```
model.score(X_train, train_targets), model.score(X_val, val_targets)
```

[167]:
```
(0.9920192346547057, 0.8563054959085369)
```

[168]:
```
model.score(X_test, test_targets)
```

[168]:
```
0.8451913451913452
```

PREDICTIONS ON NEW INPUTS:

[169]:
```
def predict_input(model, single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])
    input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
    X_input = input_df[numeric_cols + encoded_cols]
    pred = model.predict(X_input)[0]
    prob = model.predict_proba(X_input)[0][list(model.classes_).index(pred)]
    return pred, prob
```

[170]:
```
new_input = {'Date': '2021-06-19',
             'Location': 'Launceston',
             'MinTemp': 23.2,
             'MaxTemp': 33.2,
             'Rainfall': 10.2,
             'Evaporation': 4.2,
             'Sunshine': np.nan,
             'WindGustDir': 'NNW',
             'WindGustSpeed': 52.0,
             'WindDir9am': 'NW',
             'WindDir3pm': 'NNE',
             'WindSpeed9am': 13.0,
             'WindSpeed3pm': 20.0,
             'Humidity9am': 89.0,
             'Humidity3pm': 58.0,
             'Pressure9am': 1004.8,
             'Pressure3pm': 1001.5,
             'Cloud9am': 8.0,
             'Cloud3pm': 5.0,
             'Temp9am': 25.7,
             'Temp3pm': 33.0,
             'RainToday': 'Yes'}
```

[171]:
```
predict_input(model, new_input)
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
```

```
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
C:\Users\adity\AppData\Local\Temp\ipykernel_2004\3775630190.py:5: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of callin
g `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented fram
e, use `newframe = frame.copy()`
  input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
```

[171]: ('Yes', 0.7608595348304203)